

A (Really Easy) Introduction to Fault Tree Analysis (FTA) Tutorial

Understanding HOW your system has failed ... or will fail



WORKBOOK

Presented by

Dr Chris Jackson

Reliability Engineer



Copyright © 2024 Christopher Jackson

All rights reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means, including photocopying or other electronic or mechanical methods without the prior written permission of the author (Christopher Jackson), except as permitted under Section 107 or 108 of the 1976 United States Copyright Act and certain other non-commercial uses permitted by copyright law.

For permission requests, write to Christopher Jackson using the details on the following pages.

Limit of Liability/Disclaimer of Warranty: While the author has used his best efforts in preparing this document, he makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. The author shall not be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Reliability happens at the point of **DECISION**

Reliability doesn't just happen. Following standards, doing what worked 10 years ago, and doing what we have always done means we aren't focusing on what your **system is today**. It also means we won't make reliability happen.

Fault trees are great at **modelling system reliability**. They are one of several tools that can help you turn what you know about component or subsystem failure characteristics into an understanding of system reliability characteristics. Which lets you **measure reliability**.

But measuring reliability is one thing. **Improving reliability** is a much bigger thing.

Good reliability decisions are based on knowing **HOW** your system will fail

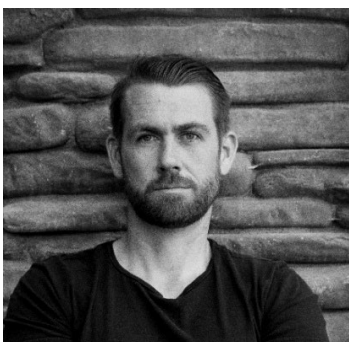
Fault trees are a great **Root Cause Analysis (RCA)** tool. They can really help you and your team identify the potential causes of failure, which then focuses your investigation on what really happened.

But it is much better to prevent failures from occurring. Making your **first design a reliable design** means you need to know how your system will fail ... from the first day of design.

Fault Tree Analysis (FTA) is a great tool for working this out

If you are interested in enrolling, or want to learn more, go to ...

www.acuitas.com/fta-course



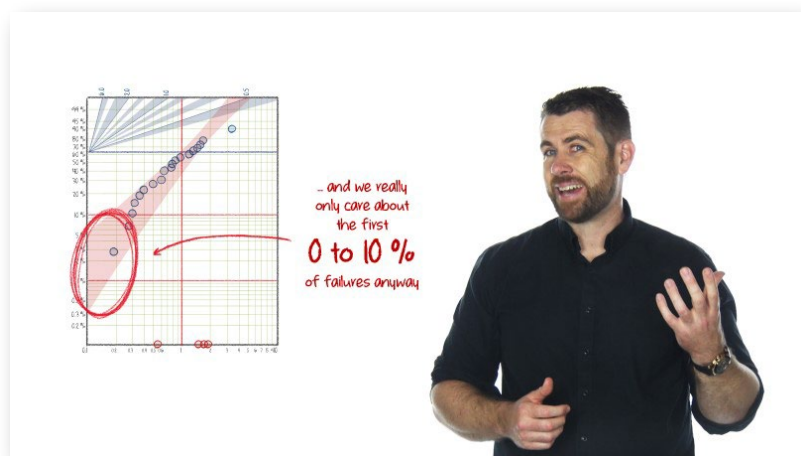
Chris Jackson, PhD, CRE, CPEng

Reliability Engineer
Founder and director of Acuitas Reliability
Co-founder of IS4 education
Author of books on how to do 'reliability stuff'
Former Lieutenant Colonel in the Australian Army
Still learning

Who is your teacher?... Dr Chris Jackson

Dr Jackson holds a Ph.D. and MSc in Reliability Engineering from the University of Maryland's Center of Risk and Reliability. He also holds a BE in Mechanical Engineering, a Masters of Military Science from the Australian National University (ANU) and has substantial experience in the field of leadership, management, risk, reliability and maintainability. He is a Certified Reliability Engineer (CRE) through the American Society of Quality (ASQ) and a Chartered Professional Engineer (CPEng) through Engineers Australia.

Dr Jackson was the inaugural director of the University of California, Los Angeles (UCLA's) Center of Reliability and Resilience Engineering and the founder of its Center for the Safety and Reliability of Autonomous Systems (SARAS). He has had an extensive career as a Senior Reliability Engineer in the Australian Defence Force, and is the Director of [Acuitas Reliability](#).



He has supported many complex and material systems in developing their reliability performance and assisted in providing reliability management frameworks that support business outcomes (that is, make more money). He has been a reliability management consultant to many companies across industries ranging from medical devices to small satellites. He has also led initiatives in complementary fields such as systems engineering and performance-based management frameworks (PBMFs). He is the author of two reliability engineering books, co-author of another, and has authored several journal articles and conference papers.

If you would like to speak to Chris or Acuitas about anything, including what we can do better in the future... please reach out to us at contact@acuitas.com

So what does a FAULT TREE look like?	5
Perspective #1 – analyzing system RELIABILITY	6
Perspective #2 – root cause analysis (RCA).....	7
Perspective #3 – robust, customer-centric DESIGN.....	7
... other things FAULT TREES can help us with.....	7
Focus on the DECISION first and foremost	8
Using fault trees to model system reliability	8
What is RELIABILITY?	8
The RELIABILITY curve.....	10
So... how does a FAULT TREE work?.....	12
Let's start with an 'AND Gate'	14
And now the 'OR Gate'	15
Analyzing a SERIES System.....	17
Analyzing a SERIES System	17
Let's look at VARIATION in TTF	20
But it is much easier to find SERIES SYSTEM RELIABILITY.....	21
Representing basic events with NUMBERS – not LETTERS	21
Analyzing a PARALLEL System	23
...but it is much easier to find PARALLEL SYSTEM RELIABILITY	25
The 'k' Out of 'n' System	26
What does 'k OUT OF n' system reliability look like?	27
Why else do we want to use a 'k OUT OF n' system?	29
Analyzing a 'k OUT OF n' System	30
What does '!' mean? ... (FACTORIALS)	30
System Reliability Analysis Summary	31
Modelling Complex System Reliability	32
Let's look at a REALLY SIMPLE (but still complex) nuclear power plant... ..	32
Complex System Reliability Analysis.....	36
Learning HOW Things Fail from RELIABILITY CURVES.....	41
...and that's it for SYSTEM RELIABILITY ANALYSIS	42
All the FAULT TREE SYMBOLS Together	43

Fault Trees and Root Cause Analysis (RCA)	43
So What is a ROOT CAUSE?.....	44
Let's Do Some RCA on a Smart Lock... ..	44
The TREE OF FAILURE	48
Then There Are IMMEDIATE ACTIONS.....	48
ROOT CAUSES Are All About What 'WE' Can Do.....	49
The Reliability Mindset	52
So How Do We DO THIS RIGHT (...Not WRONG)?.....	53
Root Cause Analysis of Our Smart Lock	54
You often waste your time trying to find the 'TRUE' root cause.....	54
The SMART LOCK.....	55
Do we need the 'TRUE' ROOT CAUSE?	62
Robust, Customer-Centric Design	63
... and the final WORDS	64

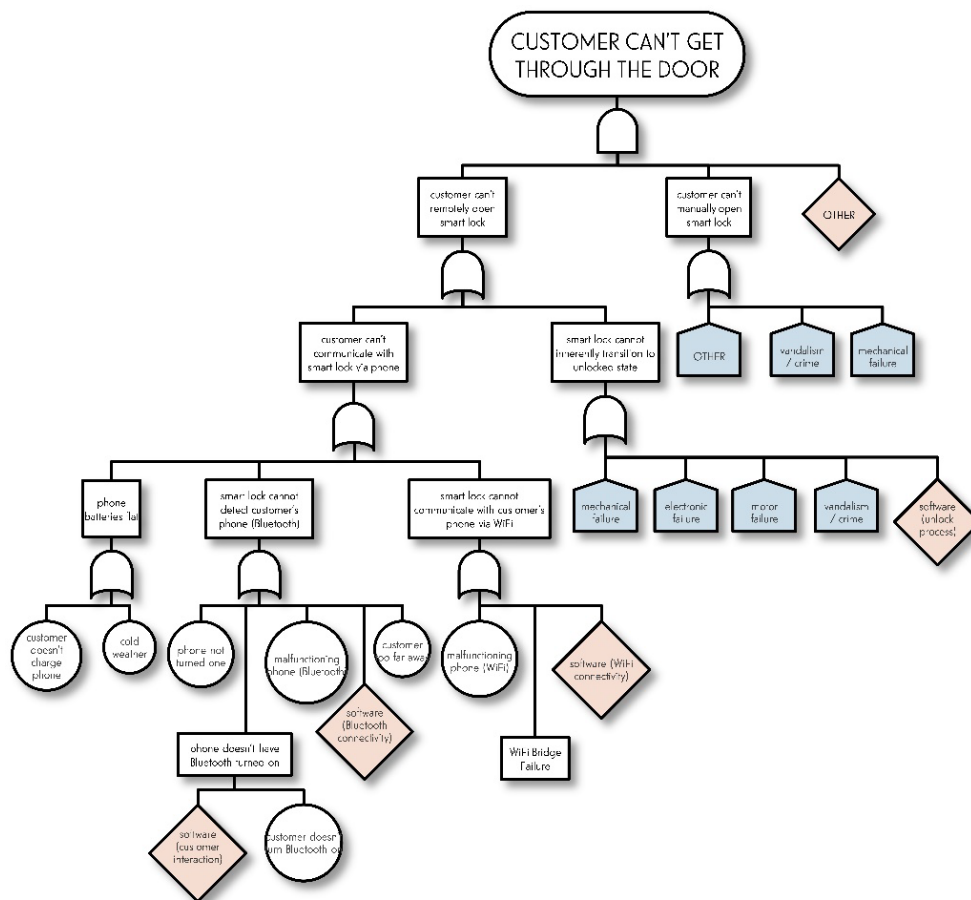
Fault tree analysis or **FTA** can be used in many **reliability engineering**, **risk management** and **innovative design** scenarios. A common aim of **fault tree analysis** is to **analyze system reliability** to help determine new product **warranty periods**. Another (and entirely different) aim is to identify the **root cause** of some undesirable event – like equipment **failure**. And yet another aim of **FTA** is to add structure to brainstorming activities such as those trying to identify emerging **design features** and **characteristics** that customers will desire (even if they don't know it yet). There are many more applications of **FTA** beyond these examples.

A key 'binding' characteristic of any worthwhile **FTA** activity is that they always be linked with **DECISIONS**. You need to understand the **decision** that **FTA** is supposed to inform in order to structure your **FTA strategy**.

This guidebook is an editable PDF that allows you to enter answers and notes directly into it. The grey boxes are for notes, and the orange boxes are for your answers to specific questions in worked exercises.

So what does a FAULT TREE look like?

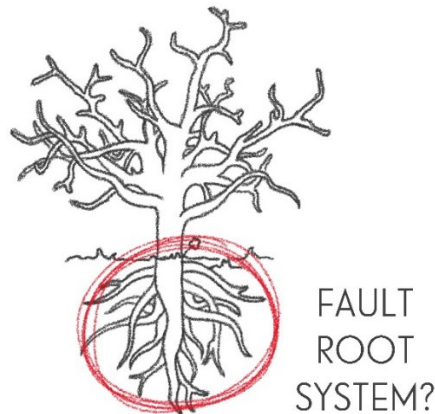
An example **fault tree** is illustrated below.



It contains different shapes and lines that start or branch down from a single shape at the top which usually represents some 'undesirable' scenario (called the **top event**). This **top event** must

relate to the decision that **FTA** is intended to inform. So, you need to understand the **decision** your **FTA** is informing in order to define your **top event**, which must occur before you construct the remainder of your **fault tree**.

Fault trees don't have a typical 'tree structure' where there is a single shape at the bottom (trunk) that branches upwards. In practice, **fault trees** look a lot more like the **root system of a tree**.



To get **fault tree** construction 'right,' it helps to understand the three different **FTA perspectives**. You need to adopt one of these **perspectives** depending on your **decision**. Most references (textbooks, standards, professors and so on) tend to focus on only one of these **perspectives** (which is usually their favourite). Not understanding these **perspectives** can result in your **FTA** being either partially or completely ineffective.

Perspective #1 – analyzing system RELIABILITY

Systems can be made up of lots of **components**. This means **system reliability** is based on **component reliability** (**reliability** being the probability that an item has not **failed** within a certain interval when used in specified conditions). **Analyzing system reliability** often starts with creating a **system reliability model** that allows you to convert what you know about **component** or **subsystem reliability** into something meaningful about **system reliability**.

So, before you use **fault trees** (or anything else) to analyse **system reliability**, you must already understand **component reliability**. This allows you to (among other things) see if your **system** 'is **reliable** enough,' and work out which of your **components** or **subsystems** is having the biggest impact on **(un)reliability**. You can also use a **system reliability model** to work out how long your **warranty period** should be by specifying an allowable '**warranty failure probability**' and then finding the usage with the corresponding **reliability** (**reliability** = 1 - **failure probability**.)

Perspective #2 – root cause analysis (RCA)

Root Cause Analysis (RCA) is all about trying to find the reason (or the likely reasons) behind **failure**. We conduct **RCA** after a **failure** has occurred. **RCA** is often defined as:

... a method for identifying the root causes of failure, faults, failure modes, defects, errors or any other event associated with a system not performing as desired.

A **fault tree** constructed for **RCA** looks very different to a **fault tree** constructed for **system reliability analysis**. The aim of **RCA** is to identify *why* an undesirable event or scenario occurred to stop it from happening again (through redesign, preventive maintenance, operational changes, new components, and plenty of other **corrective actions**).

Perspective #3 – robust, customer-centric DESIGN

Robust, customer-centric design involves pre-empting hypothetical **undesirable events** before they occur so that their **root causes** can be prevented from occurring through **robust design**, manufacturing, maintenance or operations. This perspective is like **RCA** for this reason. However, **robust, customer-centric design** also requires the **undesirable event** (like system **failure**) to be predicted.

This approach is used by many 'industry leaders' who pre-emptively design out not only system or product **failures**, but **failure** to meet customer expectations. This results in making a **first design an amazing & reliable design**, saving time and money during production because we also prevent crises and problems that cost money and cause delays.

... other things FAULT TREES can help us with

Fault trees can help us identify things like **cut sets** that help us work out what combinations of components need to **fail** for the system to **fail**. This helps us with things like **system reliability analysis** but can also help us work out **single point failures** where one event can lead to system **failure**.

Fault trees can help us work out what customers want, and what **failures** mean. For example, a military vehicle design team will need to know if a system **failure** that reduces top speed by 15 km per hour is 'worse' than a system **failure** that reduces payload by 500 kg. This informs key **design** prioritization **decisions**.

If every failure is IMPORTANT, then no failure is IMPORTANT

FTA often involves a group of people. This book will provide guidance on how to conduct **FTA**, facilitate the group, and ensure everyone has a positive experience.

Focus on the DECISION first and foremost

Some decisions are informed by 'measuring' **reliability** versus understanding the **root cause of failure**. Others still are informed by understanding **CUSTOMER EXPECTATIONS**. It is important to understand what your **decision** is, and what you need to make it right.

Using fault trees to model system reliability

The term **system reliability** is usually used to separate 'overall product or process' **reliability** from the **component reliability** of its individual elements. We typically first study and understand **component reliability** to understand **system reliability**.

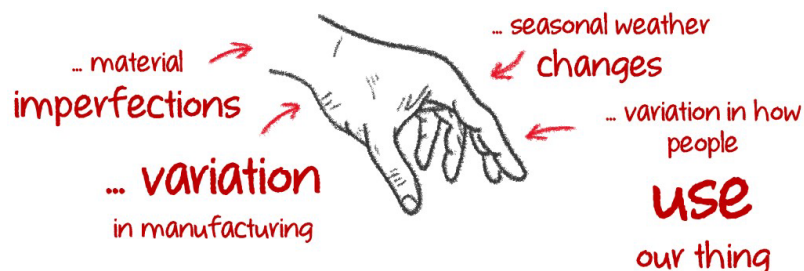
A good understanding of **system reliability** can really help **decision-making**. For example, we can use **system reliability** to determine the optimal **warranty period** for a product we are manufacturing. We don't want too many **failures** within the **warranty period** (this will cost us money and erode profit). We don't want too few **failures** within the **warranty period** (this means we missed an opportunity to advertise a longer **warranty period** to increase market share).

Reliability is one way of characterizing the uncertainty of the **failure** process. But just because **failure** is random, doesn't mean it is unpredictable and cannot be used to help **decisions** (such as the **warranty period** decision in the previous paragraph).

What is RELIABILITY?

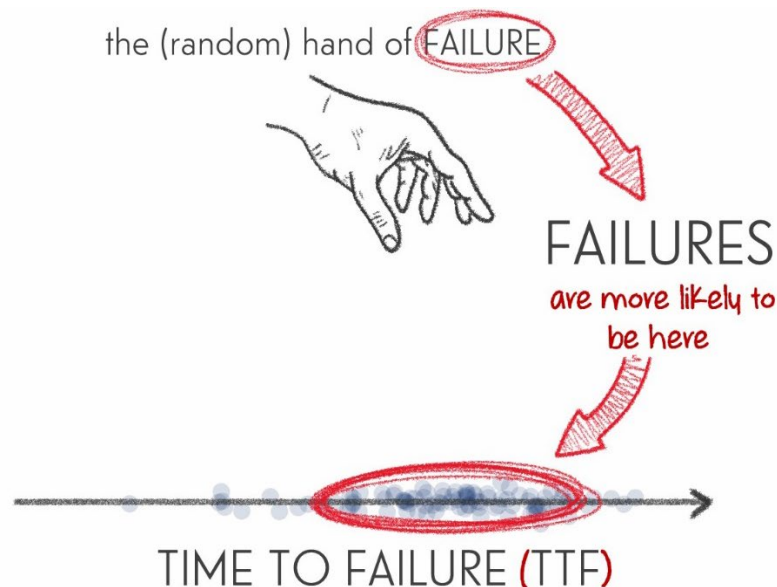
Random processes with seemingly identical **inputs** will create different **outputs**. The **random failure process** means that seemingly identical products or systems will **fail** at different times. We might theoretically know that there is underlying microscopic differences from product to product, or that users will use the products in different ways. If these differences are something we cannot discern during production, then each product or system is practically identical to our eyes.

Let's represent some of the factors that introduce uncertainty into the **Time to Failure (TTF)** of a product or system with the **(random) hand of FAILURE** (below). There are hundreds more factors than the four listed below.



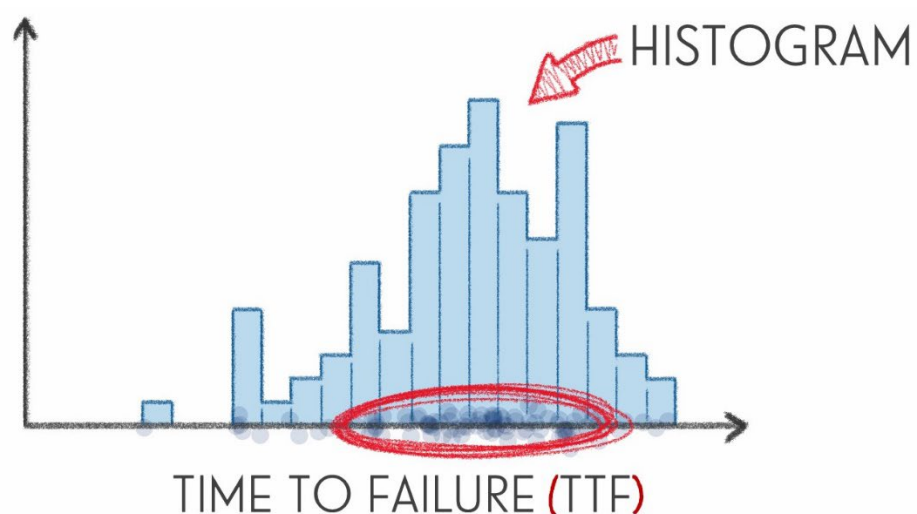
The 'time' in **TTF** could be measured in terms of distance (mile or kilometers), operating hours, cycles, or any other metric that best captures the age of the system or product. This is often not calendar time.

Even though seemingly identical products might **fail** at different times or ages, there is usually some 'consistency' or 'shared characteristics' in their **TTFs**. Identical models of a particular product might all **fail** around the 'same' time or age, as shown below (each blue circle represents the **TTF** of an individual product.)



Products or systems that tend to **fail** at around the same age or time are **wearing out**. They are slowly accumulating **damage** throughout usage that makes them increasingly likely to **fail**. **Damage** can be many things (like crack length, amount of contaminants in lubrication, diffusion in a semiconductor or corrosion).

While we can see that the blue circles above cluster around a central area creating regions of higher **failure density**, it is not easy to visualize 'how dense' **TTFs** are. This is where a **histogram** can help.



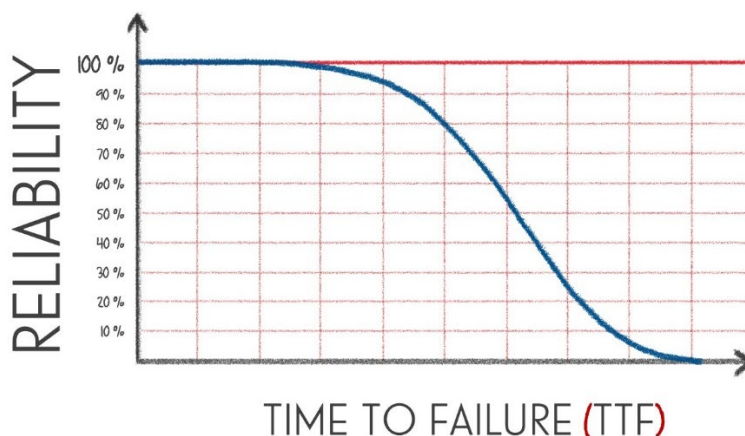
The height of each **bar** in a **histogram** represents the amount or '**density**' of the **TTFs** that fall within the base (or width) of each **bar**. The higher the **bar**, the denser the **TTFs**. **Histograms** really help us visualize the **random** nature of things like **failure**.

Histograms help us visualize the **random** nature of **failure**. But a **histogram** doesn't visualize **reliability**. Visualizing **reliability** involves introducing a vertical axis that represents the fraction or percentage of products that have not **failed**. Plotting the **TTFs** on this chart now creates a relatively smooth line.



The RELIABILITY curve

The chart above visualizes the same characteristics of the random failure process, albeit in a different way when compared to the **histogram**. If we were to (hypothetically) observe an infinite number of data points, then a perfectly smooth line will eventually be traced out. This curve is **reliability**.

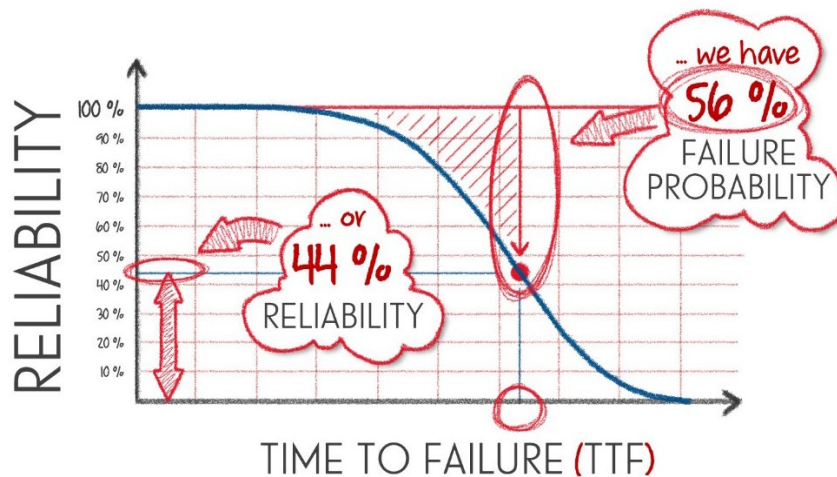


RELIABILITY is the PROBABILITY that our product HAS NOT FAILED at a particular DURATION in SPECIFIED CONDITIONS

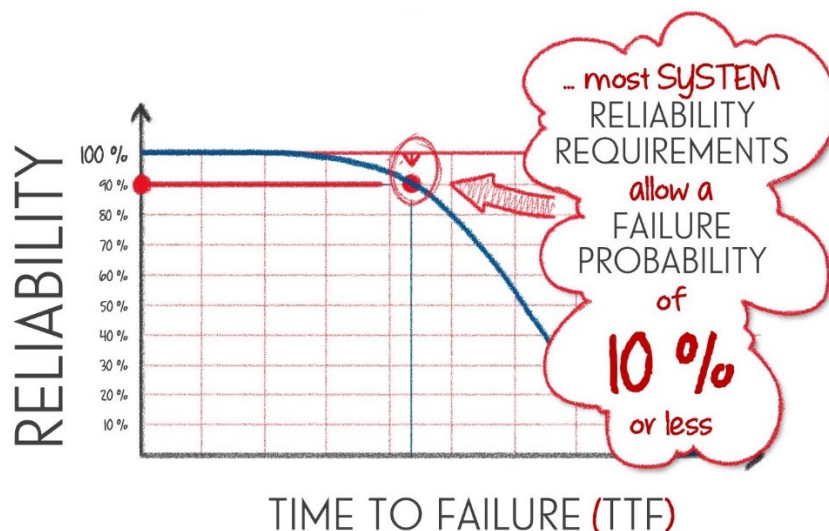
The **reliability curve** contains a tremendous amount of information regarding the **failure** of our product. Importantly, the curve shows that **reliability** can never be characterized by a single

number as it changes over time or usage. The curve above starts at 100 % which implies that a product or system is working at the start of its useful life. The curve that decreases to 0 % also shows that a product or system will always eventually **fail**.

We can use our **reliability curve** to generate a lot of useful information that might help us make better decisions. We can pick any **TTF** and then identify the **reliability** (or **failure probability**) at that point in usage, as illustrated below.

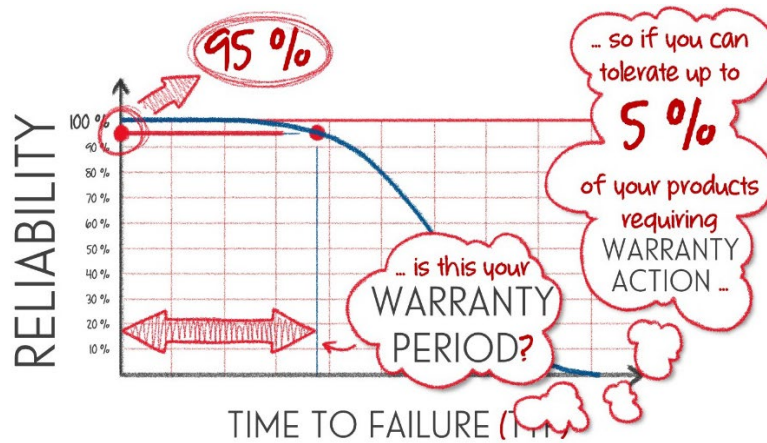


We are usually not interested in the point in time where we expect **reliability** to be anywhere near 50 %. By the time 50 % (or half) of our products have **failed**, they are too old to economically maintain. In other words, they are **obsolete**. We are usually more interested in focusing on the point in usage where a small minority of products or systems have **failed**. Most **reliability requirements** allow for **failure probabilities** of 10 % or less.



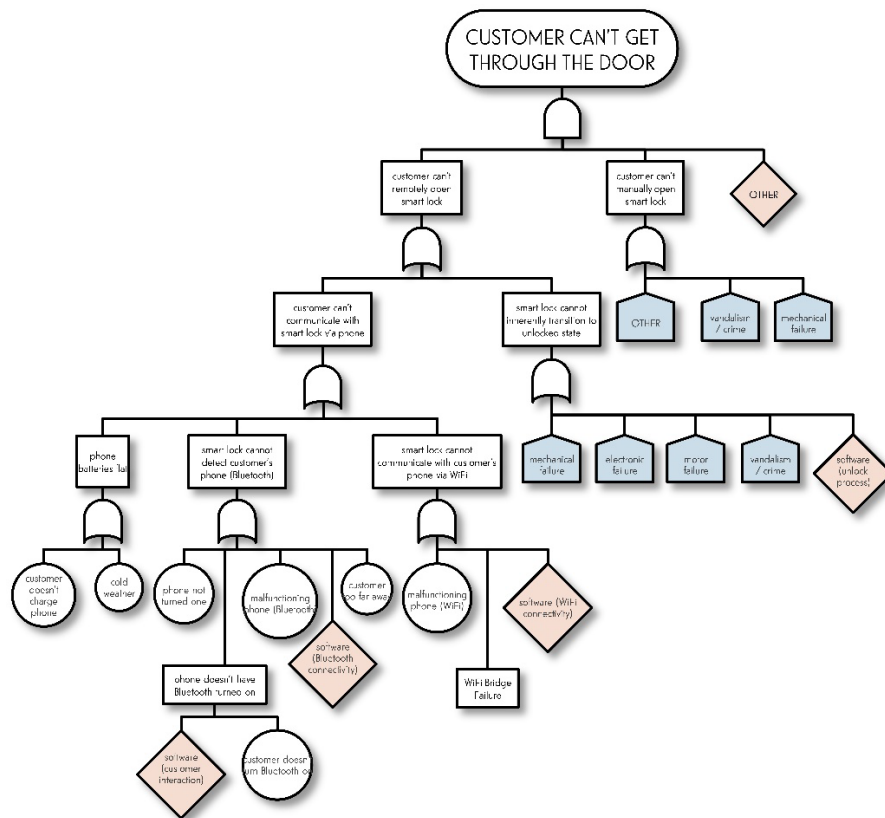
For example, say you were able to work out that for your amazing new product, you can tolerate up to 5 % of your entire sales failing during the **warranty period**. You don't want any more than 5 % to fail otherwise you will start losing too much money through **warranty costs**. You ALSO don't want your **warranty period** to be too short because this won't be attractive to your customers.

So, we can use our **reliability curve** to help us find the 'best' **warranty period** ...

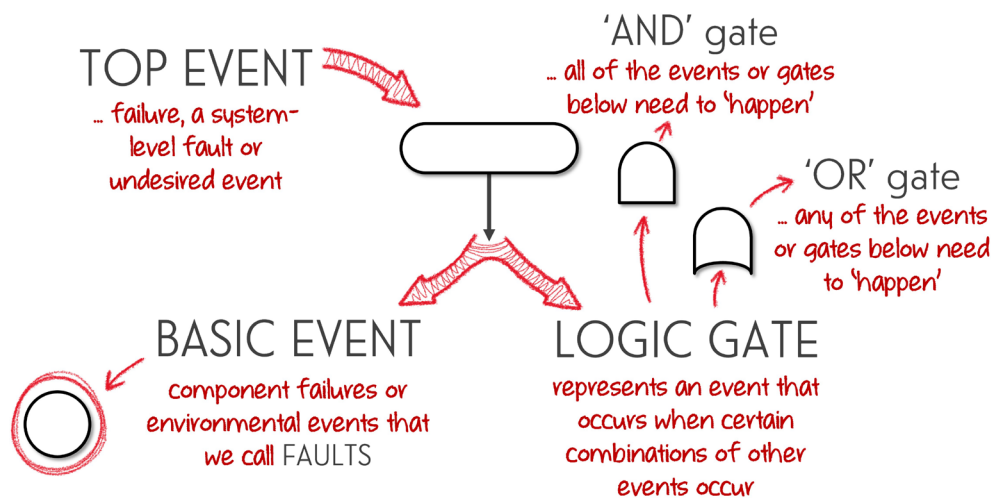


So... how does a **FAULT TREE** work?

The example **fault tree** from Chapter 1 is illustrated below with the shapes (and the small writing inside each of them) being explained later in this book.



Complicated shapes (like the one above) can be relatively easily generated 'one step at a time.'
 The most common shapes required to create a **fault tree** are illustrated below.



Most **fault trees** are built by starting with the **top event** which is depicted by a thin horizontal shape with round ends. This represents the **undesirable event** around which is the focus of the **fault tree**. A line is drawn down from the **top event** to either a single **basic event** or a **logic gate**.

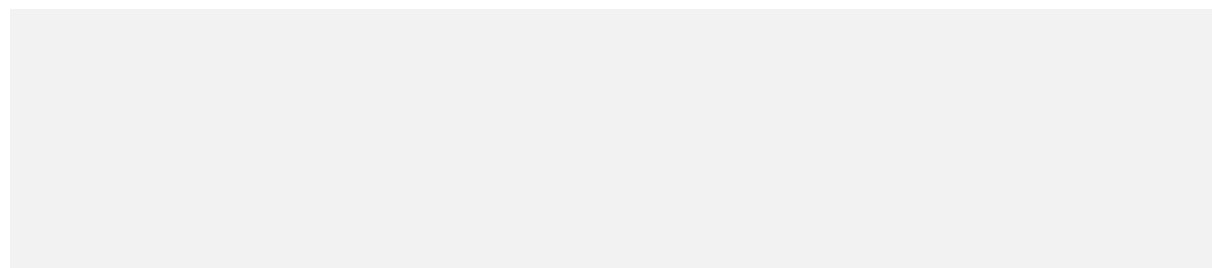
A **basic event** is often referred to as a **system fault**, which is where the **fault tree** gets its name. **Basic events** are depicted by circles or ovals. A **basic event** in a **system reliability model fault tree** typically represents **component failure**. In other types of **fault trees**, **basic events** can be design flaws, environmental events, human errors, or any other scenario that might contribute to **failure**.

Logic gates represent an **event** which is the **combination of other events**. This combination can include **basic events** and other **logic gates**.

The **logic gate** depicted by the shape with a round top and a flat bottom is an '**AND gate**.' An '**AND gate**' represents a combination where ALL the **events** or **gates** 'beneath' it need to occur for the **fault** to propagate to the **top event**. The **events** and **gates** 'beneath' the '**AND gate**' are called **input events**.

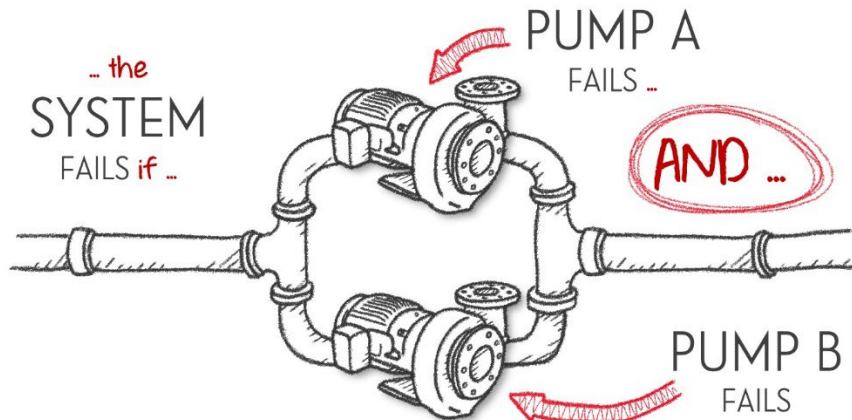
The **logic gate** depicted by the shape with a round top and round (inverted) bottom is an '**OR gate**.' An '**OR gate**' represents a combination where AT LEAST ONE **input event** needs to occur for the **fault** to propagate to the **top event**.

The '**AND gate**' and '**OR gate**' are the two most common **logic gates** used in **FTA**. Other **logic gates** used to create **fault trees** are examined later.



Let's start with an 'AND Gate'

Consider the simple, two pump system below.



The system **fails** if Pump A **fails** **AND** Pump B **fails**. In other words, we need at least one pump to **function** for the system to **function**. This is called a **parallel system** as the **components** are often configured on 'parallel operating lines' (like pumps) to allow the system to **function** if at least one component is **functioning**.

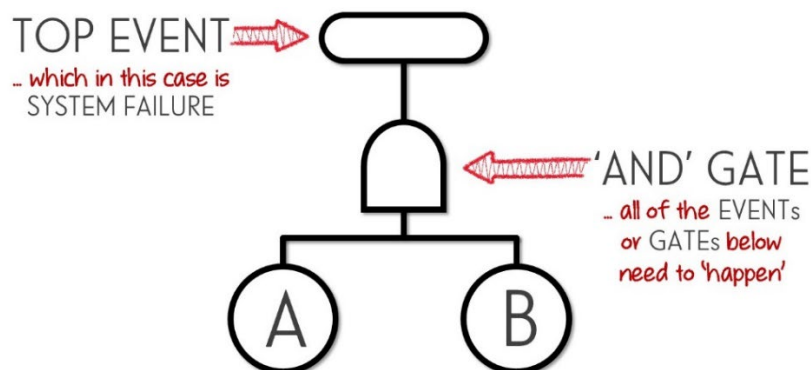
The most important word for this system is 'AND.'

To create a **fault tree** that models **two-pump parallel system reliability**, we start with defining the **basic events**.

Basic event 'A' is the event where Pump A fails.

Basic event 'B' is the event where Pump B fails.

Because the system **fails** if Pump A **fails** **AND** Pump B **fails**, we need use an '**AND Gate**' to create the **two-pump parallel system reliability fault tree**.

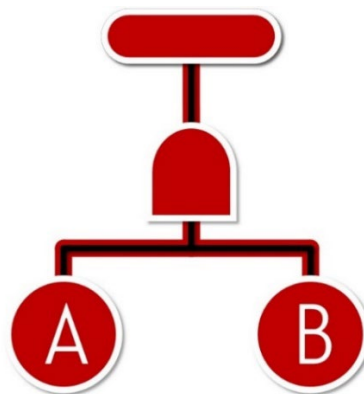


There is also an alternate way of drawing an 'AND Gate' as shown below.



Some people and software packages add a circle or dot inside their 'AND Gate' to help visually differentiate it from the 'OR Gate.' It doesn't change the meaning or logic in any way.

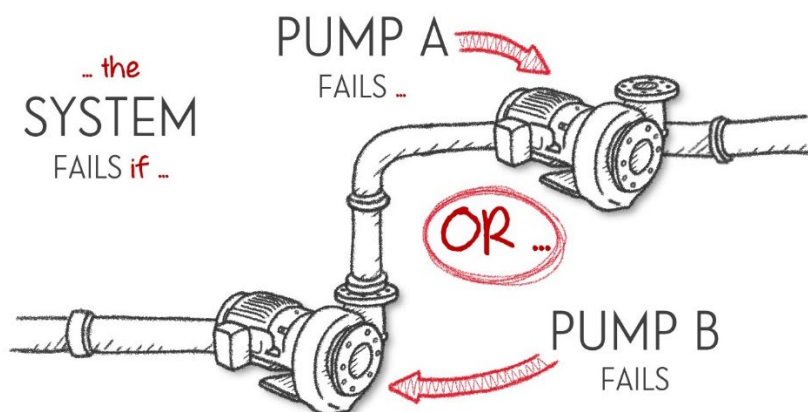
To simplify the **fault tree** illustration, we use **component identifiers** when depicting **basic events**. The letters 'A' and 'B' in the **fault tree** above mean the **basic events** represent the **failures** of Pump A and Pump B respectively. So, for the **two component parallel system, failure** looks like this (where red represents **failure**), meaning both pumps have **failed** ...



Parallel systems involve **redundant components** that can ensure the system remains **functional** when one of the other (usually identical) **components** fail.

And now the 'OR Gate'

Let's look at a different two-pump system. This system needs one pump to pump fluid 'up' to a level that is higher off the ground. The second pump then pumps that fluid from the 'higher' place somewhere else.



The system **fails** if Pump A **fails OR** Pump B **fails**. This is called a **series system** (mainly because our pumps are arranged one after the other in **series**).

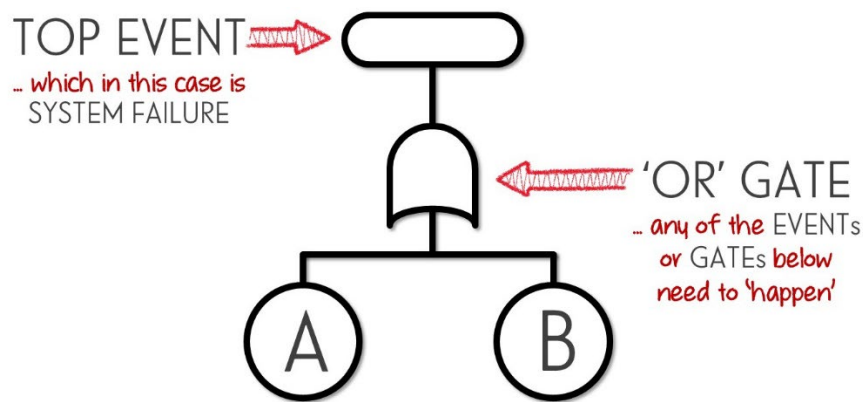
The most important word for this system is 'OR.'

The two **basic events** are the same as those for the two-pump parallel system as the components have not changed.

Basic event 'A' is the event where Pump A fails.

Basic event 'B' is the event where Pump B fails.

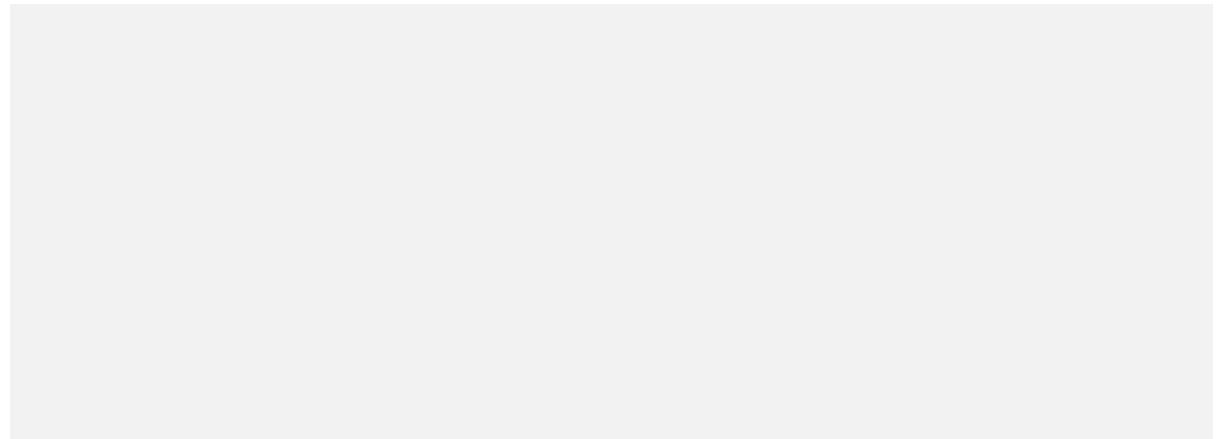
This allows us to use an **'OR gate'** to create the following **two-pump series system reliability fault tree**.



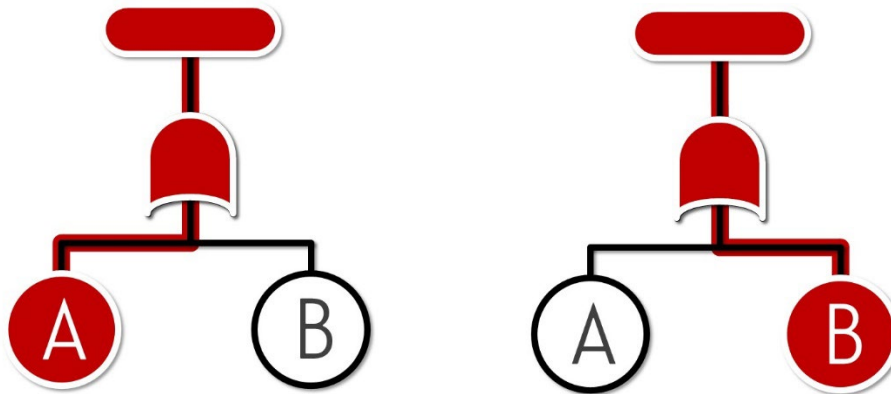
And just as was the case for the **'AND Gate,'** the **'OR Gate'** can be represented with different shapes.



Some people or software use the more 'pointed' shape on the left, or add an addition symbol on the right, to help visually differentiate the **'OR Gate'** from the **'AND Gate.'** Again, this in no way changes its logic.



So, for the **two-pump series system fault tree**, the two **failure** scenarios are represented below (where red represents **failure**) ...



Analyzing a SERIES System

While **fault trees** can visualize a **system reliability model**, we need to incorporate equations and calculations to relate component reliability characteristics to system level reliability characteristics. The **logic gates** are used to identify which equations we need to use.

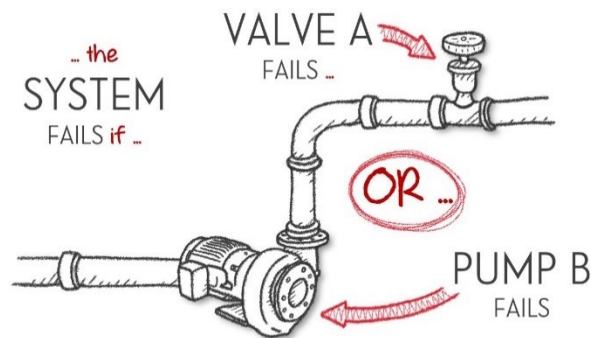
This is where we need to start talking about **probability**. Specifically, the **probability** of specific events occurring. We know that **basic events** correspond to specific components failing when our **fault tree** models **system reliability**. The **fault tree** then helps us work out which combinations of **basic events** lead to the **top event** or **system failure**. Once we have this understanding, we then need to work out how to combine **basic event probabilities** to estimate **top event probability**. This is **system reliability analysis**.

Analyzing a SERIES System

While **fault trees** can visualize a **system reliability model**, we need to incorporate equations and calculations to relate component reliability characteristics to system level reliability characteristics. The **logic gates** are used to identify which equations we need to use.

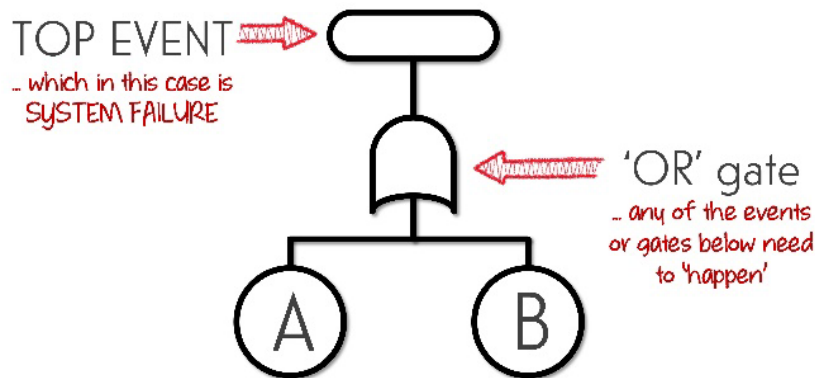
This is where we need to start talking about **probability**. Specifically, the **probability** of specific events occurring. We know that **basic events** correspond to specific components failing when our **fault tree** models **system reliability**. The **fault tree** then helps us work out which combinations of **basic events** lead to the **top event** or **system failure**. Once we have this understanding, we then need to work out how to combine **basic event probabilities** to estimate **top event probability**. This is **system reliability analysis**.

Let's go back to our **two-pump series system** whose **reliability** we can model with a simple **fault tree** based on an 'OR Gate.'



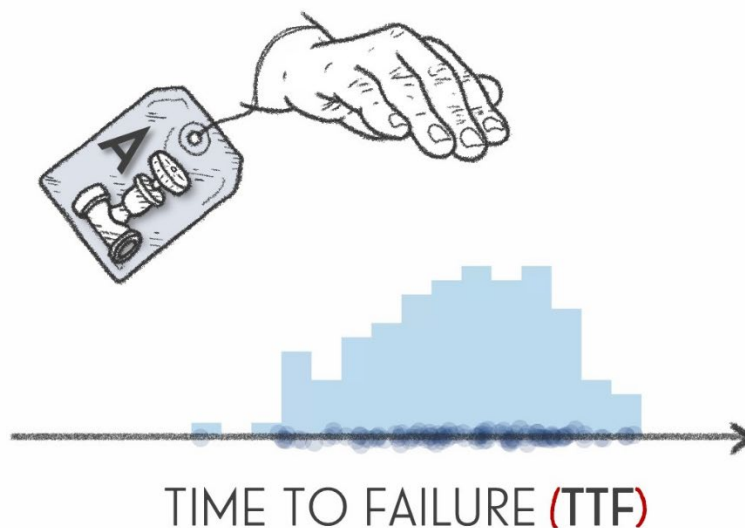
... we call this a **SERIES SYSTEM**

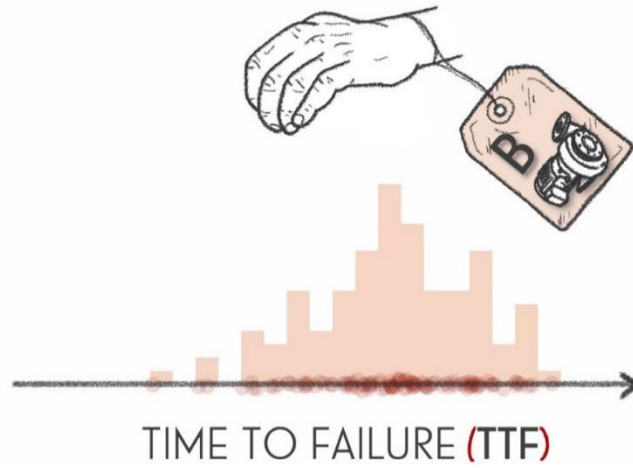
Most **systems** do not have the same **component** arranged in **series**. Different **components** are needed to provide different **functions**. So, for the series **system** above, we have a pump and a valve.



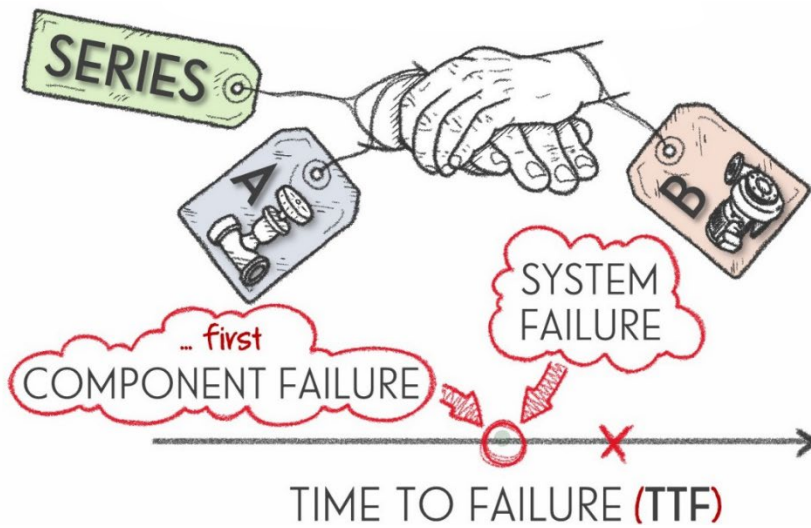
Each component will have its own **reliability** characteristics.

the (RANDOM) hand of FAILURE

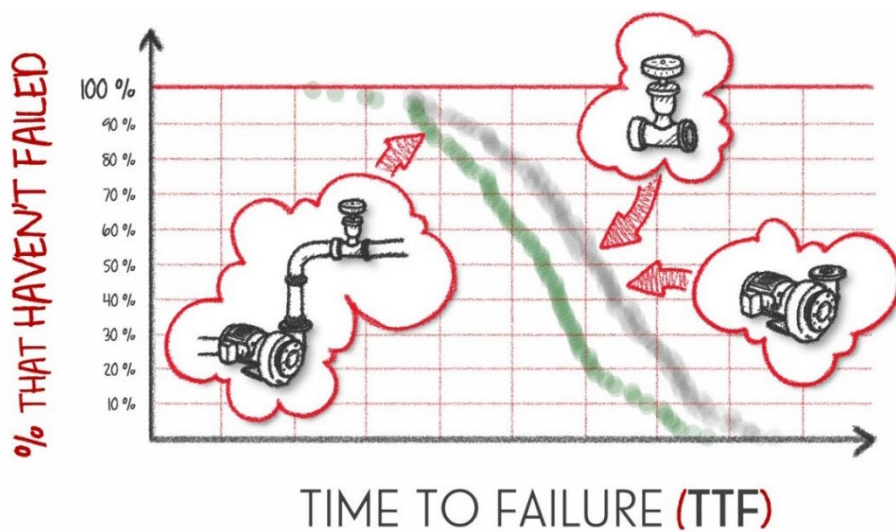




If we look at two of these **components** performing within the same two **component series system**.



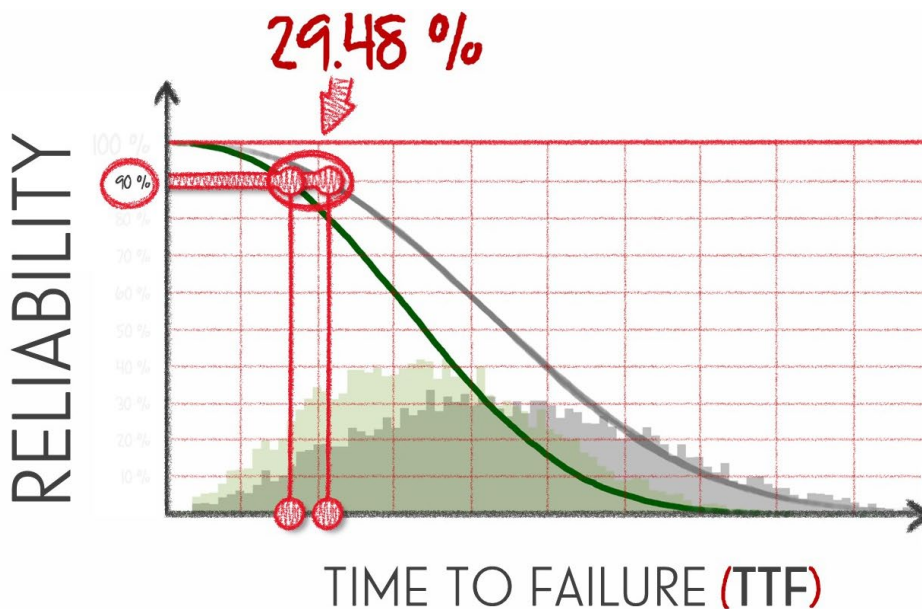
If we now look at the percentage of **series systems** (and **components**) that have **failed**.



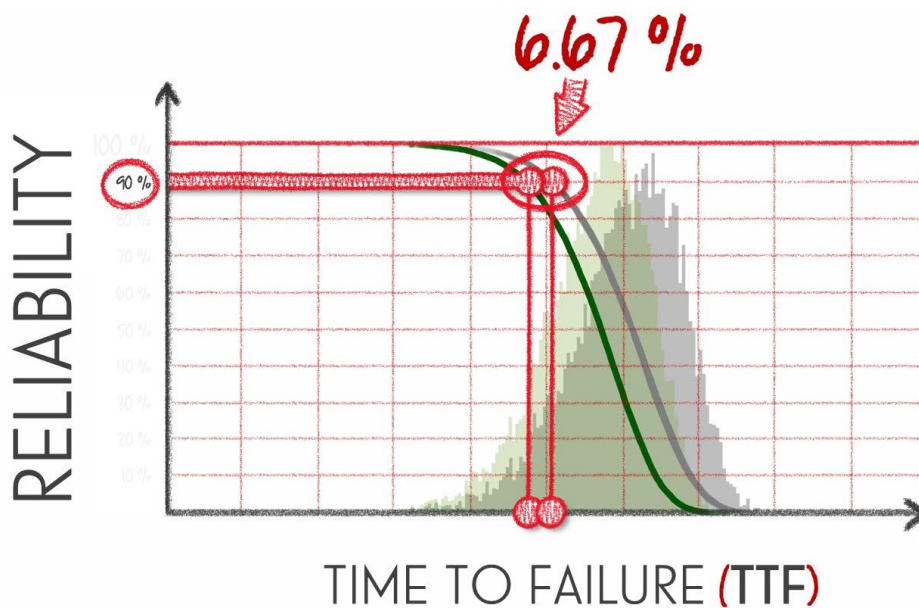
Because the **series system fails** when the first **component fails**, it is always more likely to **fail** than any single **component failing**.

Let's look at VARIATION in TTF ...

In the example below, the two **components** that in our **series system** have the same **reliability characteristics** (which is unusual for **series systems** but helps illustrate a point). You can see that the system L_{10} **life** (the time by which we expect 10 % of our **series systems** to have **failed**) is 29.48 % less than the **component** L_{10} **lives**.

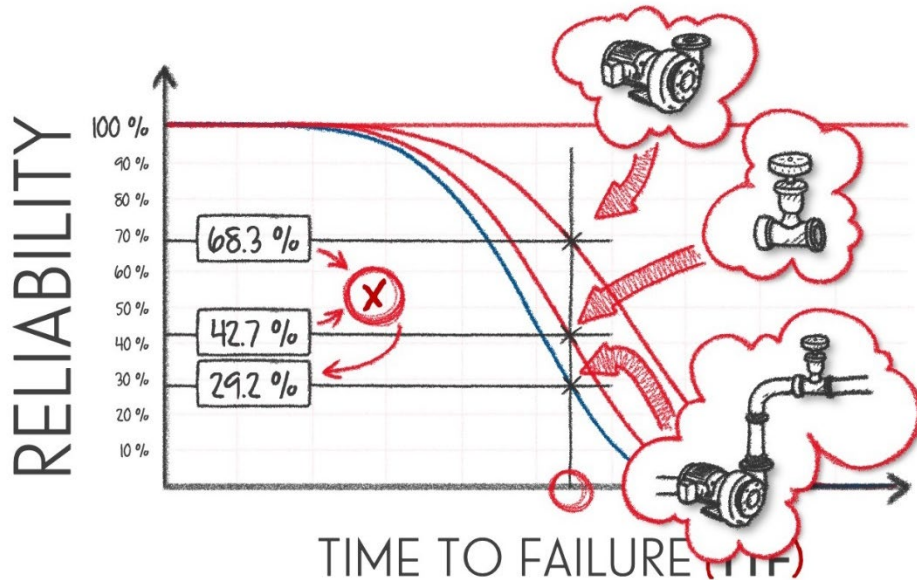


Let's look at another **two component series systems** where the **TTFs** for each **component** are on average the same duration but vary less. This makes the drop in **system** L_{10} **life** much less significant.



But it is much easier to find SERIES SYSTEM RELIABILITY

Series system reliability is the product of component reliability.



Reliability is often represented with the following notation.

RELIABILITY of the
SYSTEM at time
(or usage) 't'

$$R_{System}(t)$$

Representing basic events with NUMBERS – not LETTERS

The **fault trees** we have created up until now tend to use 'letters' as **component identifiers** for **basic events**. For example, 'A' represents the event where '**component A fails**.' We have also represented the **reliability** of component 'A' with the function $R_A(t)$.

But... we are now going to use numbers and not letters to represent components. For example, we will now represent the **reliability** of component 'A' not with $R_A(t)$, but with $R_1(t)$. The reason for this will become clear when we develop the equations that help us calculate **system reliability**. So, for our **two component series system**, **system reliability** now becomes ...

$$R_{System}(t) = R_1(t) \times R_2(t)$$

Series systems can have more than two components, and **system reliability** is the product of all their **reliabilities**. We often use the letter n to represent how many **components** are in our system.

$$R_{System}(t) = R_1(t) \times R_2(t) \times R_3(t) \times R_4(t) \times R_5(t)$$

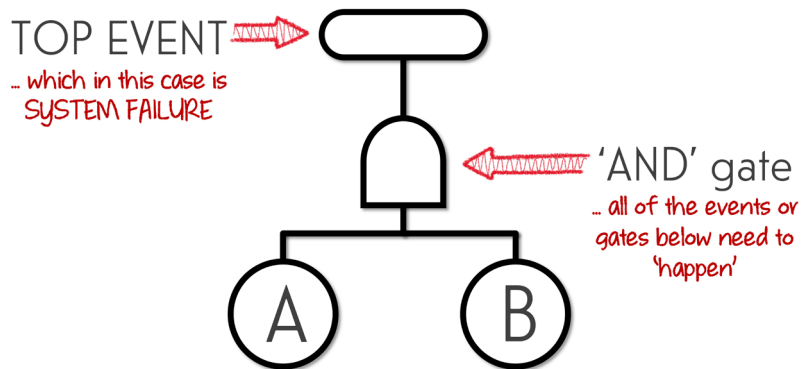
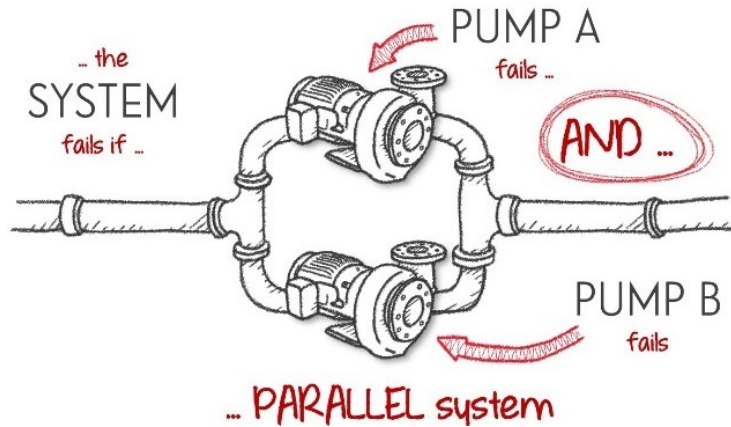
The general equation for **series system reliability** is represented by the equation below. The Greek letter Π (π) represents the multiplication of component **reliability functions**.

$$R_{System}(t) = \prod_{i=1}^n R_i(t) \dots \text{or} \dots = R_1(t) \times R_2(t) \times \dots \times R_n(t)$$

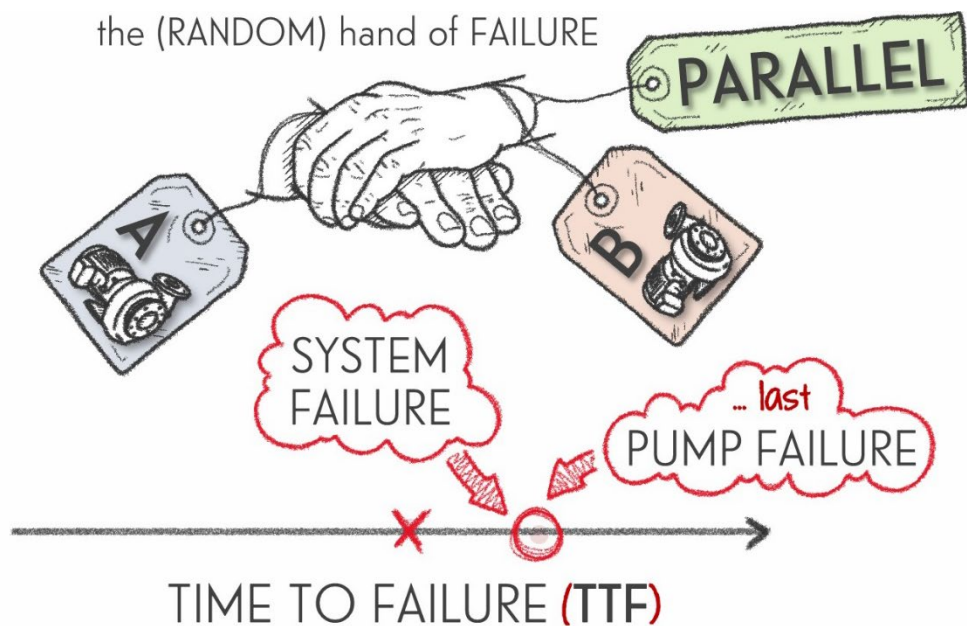
This is why we use numbers to represent **component reliability functions**. It allows us to represent any component's **reliability function** with the general term $R_i(t)$ where i can be 1, 2, 3, 4, or any other number that corresponds with components 'A,' 'B,' 'C,' 'D,' or any other component identifier.

Analyzing a PARALLEL System

Let's now look at our **two-pump parallel system**.



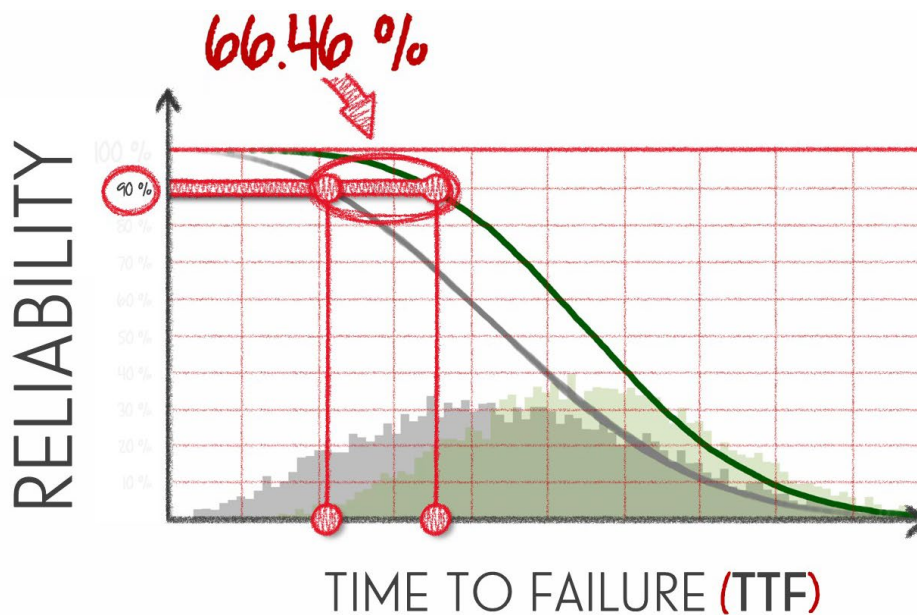
If we look at two of these **components** performing within the same two **component parallel system** ...



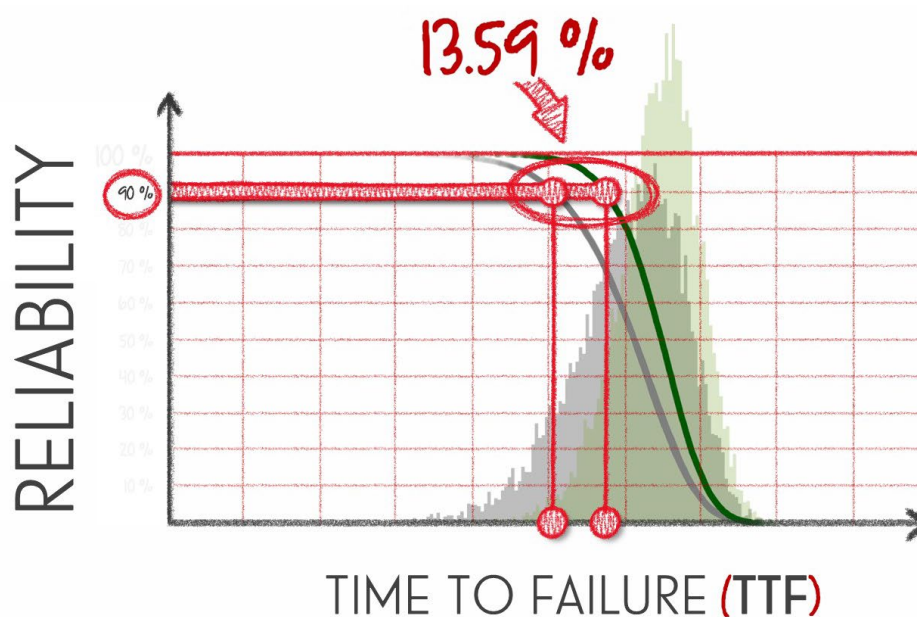
Parallel systems involve **redundant components** which are more likely to be identical to the main or primary **component**. This means the **components** of a **parallel system** (because they are likely to be identical) are therefore likely to have similar **reliability characteristics** (unlike a **series system**).

And because the **system** fails when the last **component fails**, its **reliability curve** will always be higher than the **reliability curve** of any **component**.

Parallel system reliability is also influenced by the **variation** in **component TTF**. For example, the **parallel system L_{10} life** (or the time by which we expect 10 % of our **parallel systems** to fail) is 66.46 % longer the L_{10} life of one of our two **parallel components**.

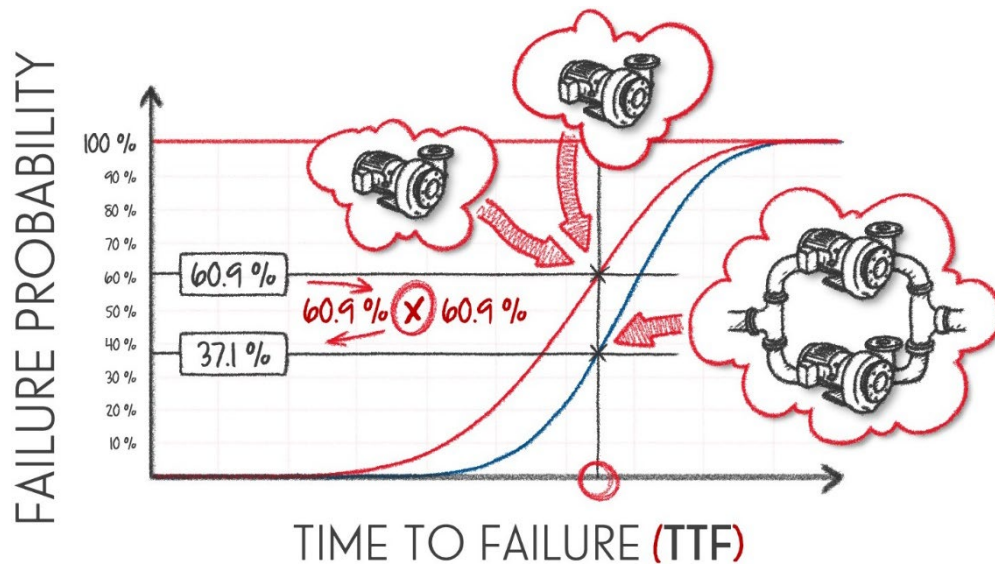


And when the variation in component TTF decrease ...

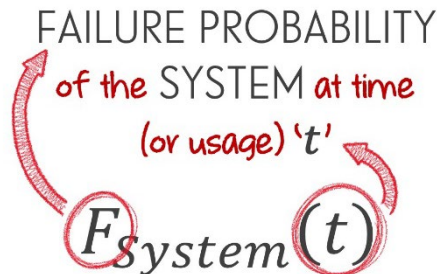


...but it is much easier to find PARALLEL SYSTEM RELIABILITY

Parallel system failure probability is simply the product of all parallel components' failure probabilities.



Modelling **parallel system** reliability characteristics is most 'simply' viewed from the perspective of **failure**.



Once we know system failure probability, we can easily calculate system reliability (and vice versa.)

$$R_{System}(t) = 1 - F_{System}(t) \quad \dots \quad F_{System}(t) = 1 - R_{System}(t)$$

For **parallel systems**, system failure probability is simply the product of all the component failure probabilities IF COMPONENT FAILURES ARE INDEPENDENT. For our **two-pump parallel system**:

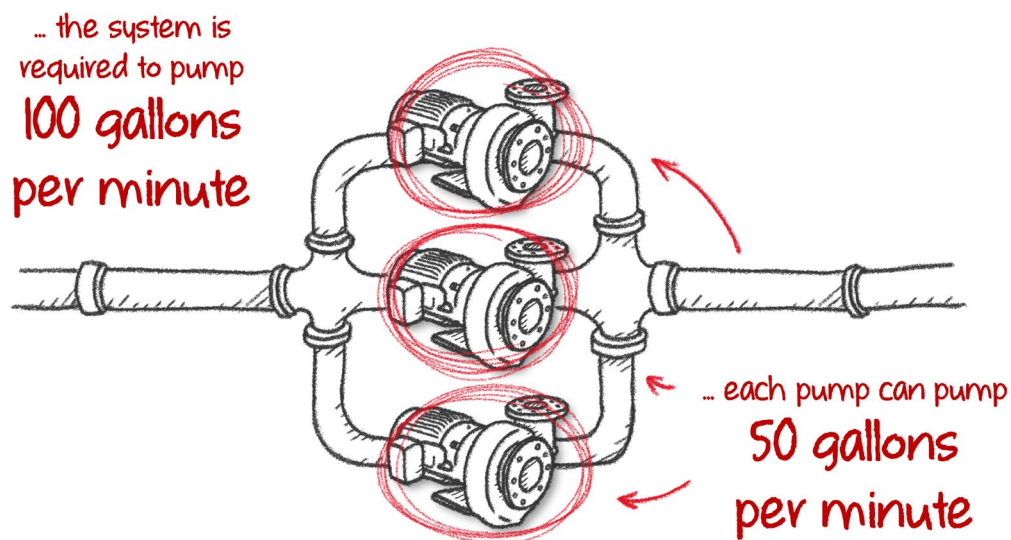
$$F_{System}(t) = F_1(t) \times F_2(t)$$

Parallel systems can have more than two components. So, the general expression for the **failure probability** of any **parallel system** is represented below.

$$\begin{aligned} F_{System}(t) &= \prod_{i=1}^n F_i(t) \text{ ... or ...} \\ &= F_1(t) \times F_2(t) \times \dots \times F_n(t) \end{aligned}$$

The 'k' Out of 'n' System

Many systems require logic beyond simple '**AND**' and '**OR Gates**' to fully represent how they work. Look at the three-pump system below.



With each pump being able to pump 50 gallons per minute, and the system requiring 100 gallons per minute to be pumped, we need two pumps to be functional. This means we can tolerate one pump **failing** (in other words, one pump is **redundant**). So, the system is not a **series system**.

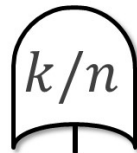
However, we need more than one pump to work for the system to meet its 100 gallons per minute requirement. So, the system is not a **parallel system**.

This is what we call a '**k out of n**' system. '**k**' refers to the minimum number of components that need to be working for the system to work. '**n**' refers to the total number of components that have the 'ability' to function.

Our three-pump system above is a '**2 out of 3**' system.

'**k out of n**' systems are represented using a special logic gate symbol that resembles an '**OR Gate**' symbol, but contains the number of functional systems for it the gate to NOT be true.

In other words, we need at least 'k' things to be functional for our subsystem to be functional.

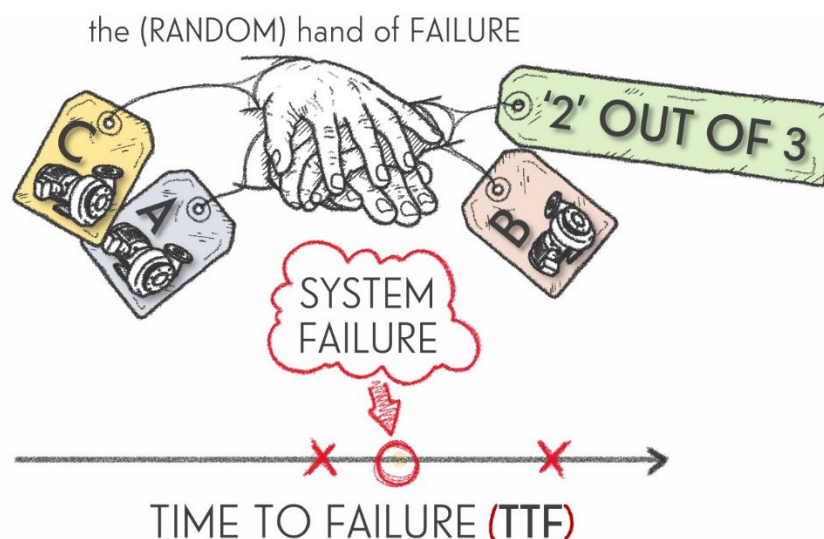


'k' OUT OF 'n'
... where we need at least 'k'
events to occur or happen

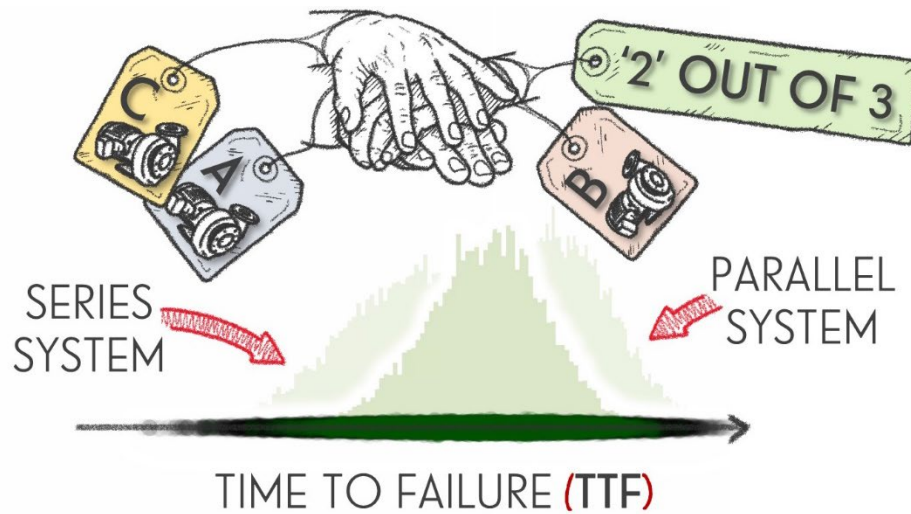
It also turns out that both series and parallel systems are forms of '**k out of n**' systems. For example, a parallel system only needs one component to work, so it is a '**1 out of n**' system. And a series system needs all components, so it is a '**n out of n**' system.

What does 'k OUT OF n' system reliability look like?

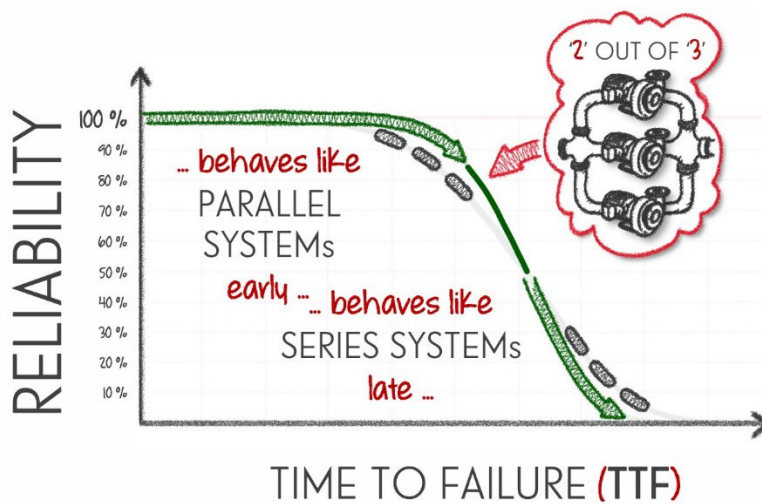
Let's combine the three random hands of failure that represent the failure processes of three pumps in a '**2 out of 3**' system. Because our system needs two pumps to work, the system will fail when the second to last pump fails. The two red crosses below represent the times to failure of the first and last pump to fail.



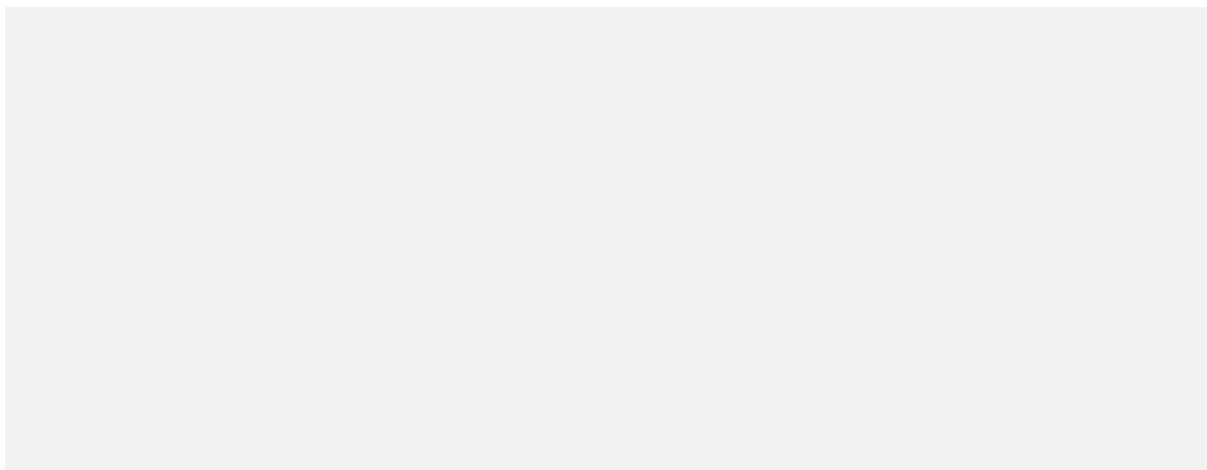
If we were to test lots of '2 out of 3' systems, we can create a histogram like the one below. You can see how it sits 'between' the histogram of a series system TTFs, and the parallel system TTFs.



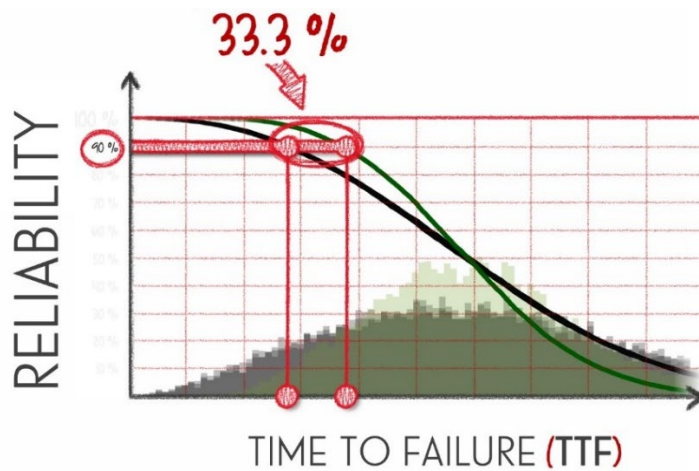
So, the histograms visualize the reliability of '2 out of 3' systems in one way. But the reliability curves illustrate something else.



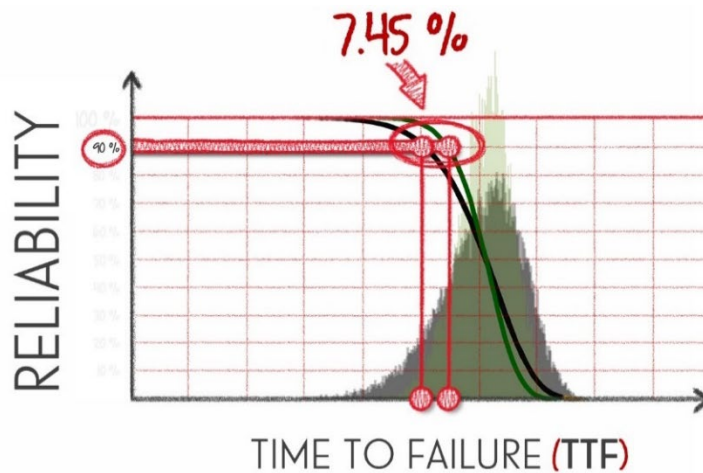
And just like for series and parallel systems, how far 'spread out' the TTFs are affect system reliability characteristics.



For example, with pump TTFs spread out to the extent they are below, the L_{10} life of the '2 out of 3' system is 33.3 percent longer than the L_{10} life of a single pump.



But this percentage decreases when the pump TTFs are less spread out.

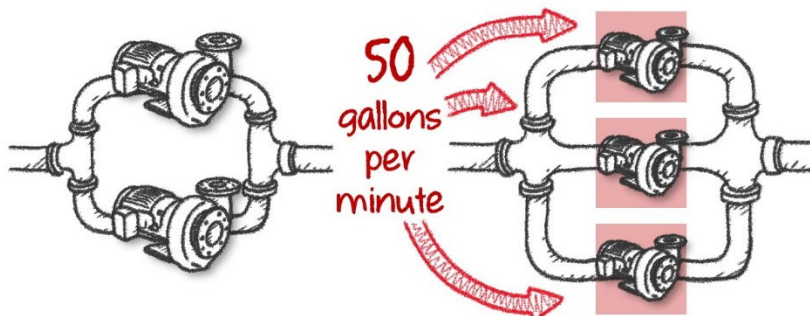


Why else do we want to use a 'k OUT OF n' system?

Using a '*k* out of *n*' system allows us to use smaller components.

PARALLEL SYSTEM

'2' OUT OF '3' SYSTEM



... the SYSTEM is required to pump 100 gallons per minute

This might be very helpful where space is limited. It also allows degraded performance when two pumps fail. Even though the system will have failed when there is only one functional pump, being able to pump at a lower rate does offer some ongoing functionality.

Analyzing a 'k OUT OF n' System

The equation to calculate '**k out of n**' system reliability is a little more complicated. If the reliability of each component is the same (or in other words, all components are the same), then the 'k out of n' system reliability becomes:

$$R_{System}(t) = \sum_{i=k}^n \frac{n! [R(t)]^i [1 - R(t)]^{n-i}}{i! (n - i)!}$$

The symbol ' Σ ' is the Greek uppercase letter 'sigma.' It represents addition. The indices allow us to simplify notation further. For example:

$$\sum_{i=1}^n x_i = x_1 + x_2 + \dots + x_n$$

This equation would usually need software to solve it. And if the components have different **reliabilities**, then the expression above needs to be derived 'by hand' using potentially complicated **Boolean algebra**. Fortunately, most '**k out of n**' systems involve the same components, so they have the same **reliabilities**, which then allows us to use the equation above.

What does '!' mean? ... (FACTORIALS)

You might have noticed that there was a strange symbol in the '**k out of n** system reliability equation' on the previous page. Some of you might know what this symbol means (in a mathematical context). Some of you might not. So which symbol are we talking about?

$$R_{System}(t) = \sum_{i=k}^n \frac{[R(t)]^i [1 - R(t)]^{n-i}}{i! (n - i)!}$$

We are talking about the '!' which some people call the **exclamation point** or **exclamation mark** when used in a sentence. But when we see the '!' symbol in a mathematical formula, it is called the **factorial**.

So, what is a **factorial**? It is simply the product of all the integers starting from 1 up to a certain number. For example, '**n factorial**' or '**n!**' is defined as ...

$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 3 \times 2 \times 1$$

For (another) example, '**2 factorial**' or '**2!**' is ...

$$2! = 2 \times 1 = 2$$

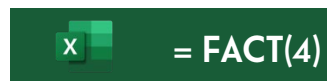
And some more examples:

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

The **factorial** is very useful when we need to deal with **combinations** of things. In a '**k out of n**' system, we need to deal with a combination of '**k**' functional components out of a total of '**n**' components. That is why we see '!' in the reliability function of a '**k out of n**' system.

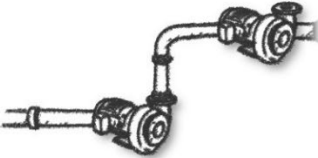

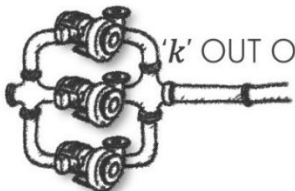
Factorials can become very difficult to calculate once the integers involved start getting bigger. It's important to use a software tool like Microsoft Excel to help you out. For example, you can use the following formula in Microsoft Excel to calculate '**4 factorial**' or '**4!**':



This allows us to evaluate our '**k out of n**' system reliability. The summation sign allows us to add all the probabilities of different numbers of components working to determine system reliability.

System Reliability Analysis Summary

We have now worked out how to calculate the **reliability** and **failure probability** of **series**, **parallel** and '**k out of n**' systems. The corresponding formulae are summarized below.

		 'k' OUT OF 'n'
$R_{System}(t) = \prod_{i=1}^n R_i(t)$	$1 - \prod_{i=1}^n 1 - R_i(t)$	$\sum_{i=k}^n \frac{n! [R(t)]^i \times [1 - R(t)]^{n-i}}{i! (n - i)!}$
$F_{System}(t) = 1 - \prod_{i=1}^n 1 - F_i(t)$	$\prod_{i=1}^n F_i(t)$	$\sum_{i=k}^n \frac{n! [R(t)]^i \times [1 - R(t)]^{n-i}}{i! (n - i)!}$

It starts with being able to recognize what characteristics of a **system** make it '**series**,' '**parallel**,' or '**k out of n**.' This allows you to create the right **system reliability model** using a tool like a **fault tree**. The next two chapters involve us practicing how to use these formulae for the systems we have examined so far. By the end of chapter 7, you should be an expert in analyzing system reliability!

Modelling Complex System Reliability

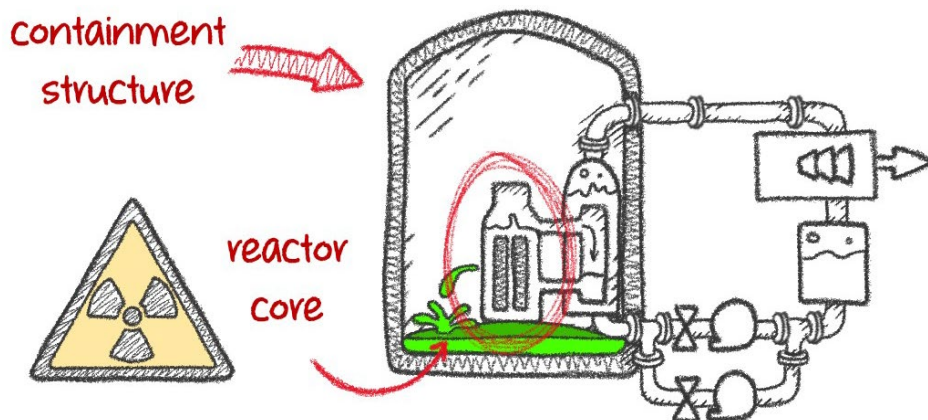
So far, we have developed **fault trees** to model the **reliability** of simple, two-component systems. The first was the **parallel system** that **fails** when all components **fail**. The corresponding **system reliability model fault tree** is based on the '**AND Gate**.' The second was the **series system** that **fails** when any component **fails**. The **corresponding system reliability model fault tree** is based on the '**OR Gate**.' And the third was a '**k** out of '**n**' system that will fail when less than '**k**' components are still working.

The **parallel** and '**k** out of '**n**' systems involve **redundancy**, or **redundant components** that ensure the system remains **functional** even if another (usually identical) component **fails**. The **series system** has no **redundancy** and will **fail** when any component **fails**.

Most systems have more than two components. **System reliability model fault trees** are usually more complicated than the simple **series, parallel** and '**k** out of '**n**' examples above. However, the approach to creating a more complicated **fault tree** is still relatively straightforward if approached in a methodical way.

Let's look at a REALLY SIMPLE (but still complex) nuclear power plant...

A nuclear power is very complex. Many of its subsystems are themselves complex systems. A very simplified illustration of a nuclear power plant is shown below.



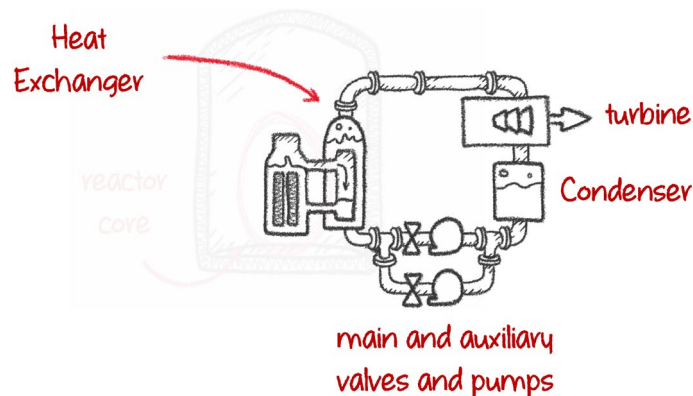
The most 'important' part of a nuclear power plant is the reactor core. This is where the uranium fuel undergoes nuclear fission to generate heat. This heat is imparted into water that circulates around the fuel, which means it is exposed to many (very dangerous) radioactive by-products. For this reason, reactor cores are always built within a containment structure that is designed to capture any leaks of this coolant water.

EXERCISE

To generate electricity, the thermal energy that is passed into the coolant water that circulates around the uranium fuel needs leave the containment structure, without any of the coolant water itself leaving the containment structure.

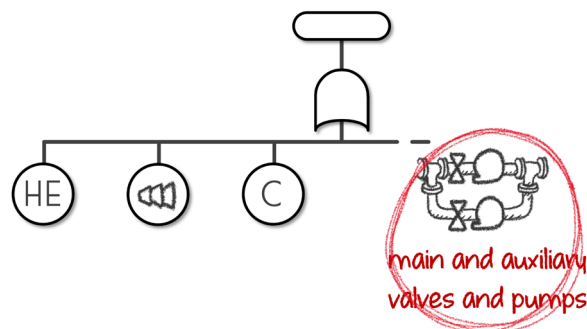
This is where the **SECONDARY COOLING LOOP** becomes critical.

The (very simplified) version of the secondary cooling loop is illustrated below. Heat is transferred to the water in the secondary cooling loop via the heat exchanger that removes heat from the water circulating in the reactor core.



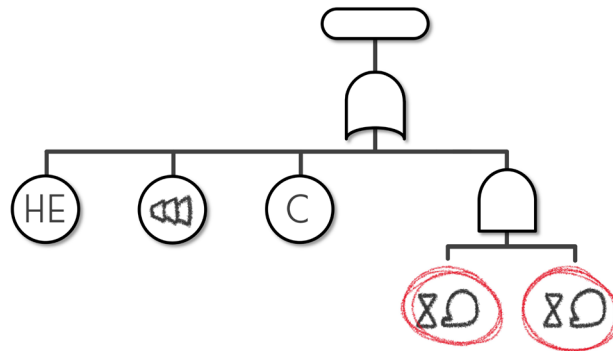
The water in the secondary cooling loop vaporizes and turns into steam. It is then transported to a turbine that generates electricity. After it leaves the turbine, it passes through a condenser that removes residual heat and condenses the steam back into water. And then it is finally driven back into the heat exchanger using a combination of valves and pumps. In the system above, each valve-pump pair is arranged in parallel.

The first thing you should try is arranging the secondary cooling loop from a high-level, trying to create a series or parallel structure. In this case, we can create a high-level series system based on an 'OR gate' if we consider the valves and pumps to be a single subsystem. For clarity, we are going to use component symbols and not just letters as basic event identifiers. This is only to help us remember what each basic event represents **IN THIS EXERCISE ...** in practice we would use letters or other identifiers.

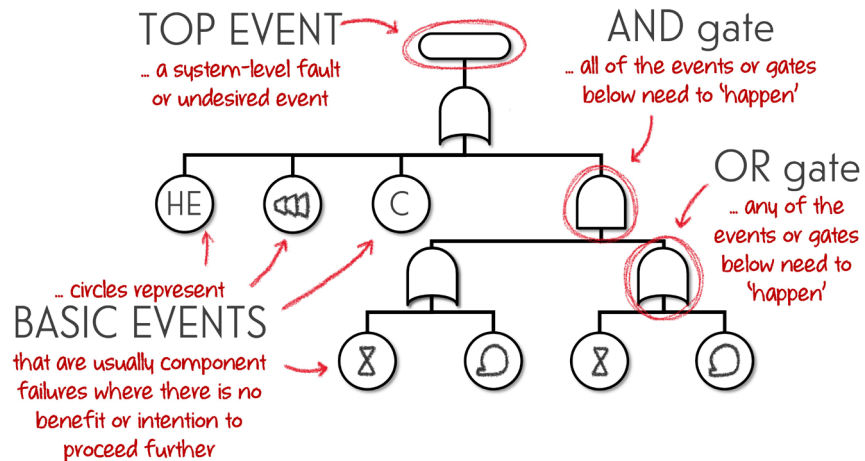


EXERCISE

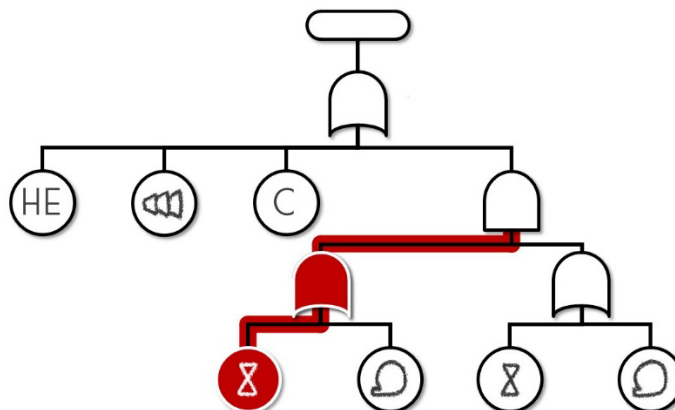
The next step is to focus on the main and auxiliary valves and pumps. We can see that each valve-pump pair is arranged in parallel. So, we now create a parallel subsystem that is based on an 'AND Gate.'



The last step is to look at each valve-pump pair. They are arranged in series. So, we replace each valve-pump pair above with a series valve-pump subsystem that is based on an 'OR Gate.'

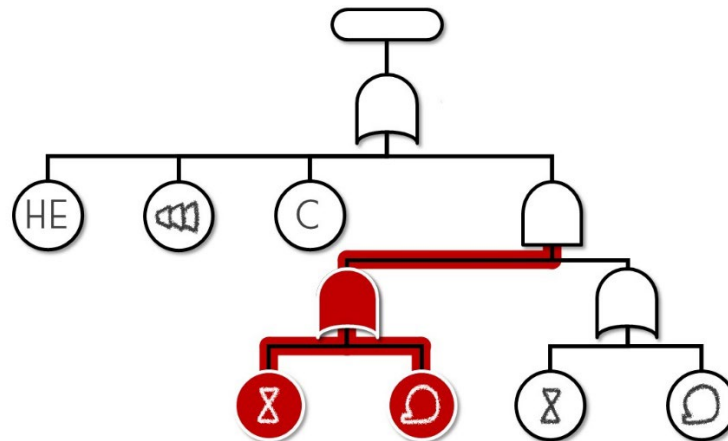


Let's use the nuclear power plant secondary cooling loop system reliability fault tree to work out what happens if the primary (or first) auxiliary valve fails. Let's represent this event by shading the basic event that corresponds with the primary auxiliary valve failure red, and trace how this system fault propagates up through our fault tree.

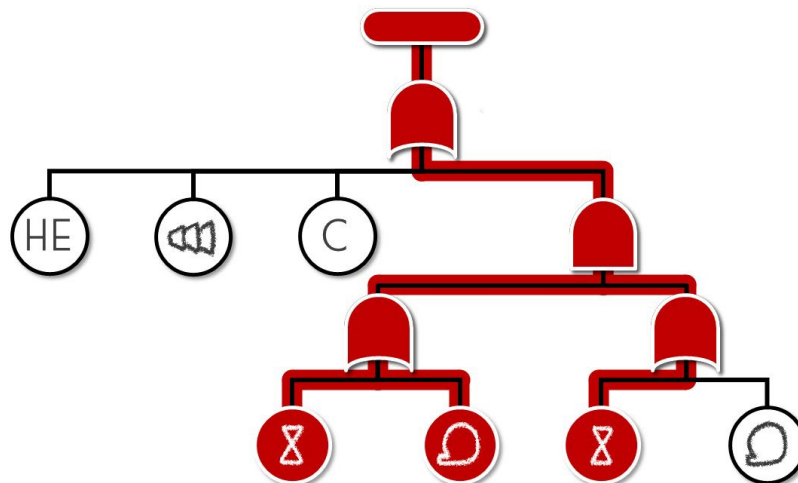


EXERCISE

We can see that the system fault passes through the first 'OR Gate' because it only requires one (any) input event to occur for an output to occur. But our system fault stops when it gets to the next 'AND Gate' because it requires ALL input events to occur for an output to occur. So, when our primary auxiliary valve fails, the system still works. This system will still work even if the primary pump fails after the primary valve fails (see below).



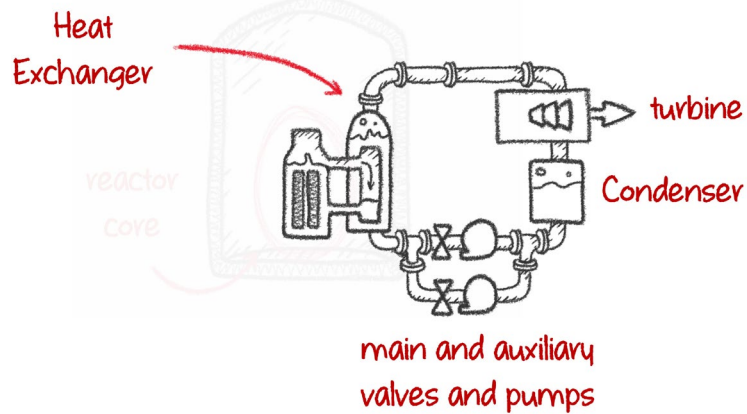
But if the secondary auxiliary valve fails (in addition to the primary auxiliary valve and pump), then our secondary cooling loop will fail. This is because the 'AND Gate' that was 'stopping' the system fault from propagating up through the fault tree now has ALL input events occurring. So, it now creates an output event that allows the system fault to move up to the next 'OR Gate' and pass through it, not stopping until it reaches the top event.



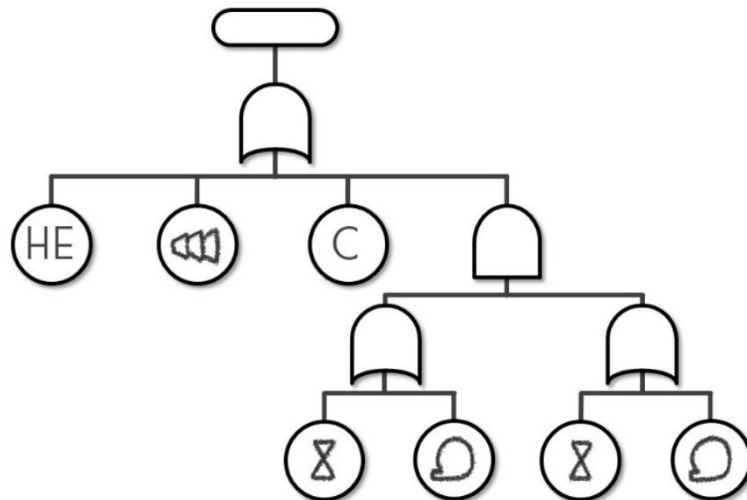
Complex System Reliability Analysis

EXERCISE

Let's go back to the secondary cooling loop of our nuclear power plant.



The secondary cooling loop fault tree is illustrated below.





We are going to estimate the secondary cooling loop reliability at 200 years using the following subsystem reliability estimates.

	reliability @ 200 years
HE	0.95
	0.92
C	0.89
	0.75
	0.72

EXERCISE

While we create fault trees from the 'top down,' we estimate system reliability working from the 'bottom up.' In this case, the 'bottom' parts of our fault tree are the valve-pump series subsystems.

	reliability @ 200 years
HE	0.95
	0.92
C	0.89
	0.75
D	0.72



We know that because each valve-pump series subsystem is in series, the reliability of each subsystem is product of the valve reliability and pump reliability.

We know for a SERIES system that reliability is ...

$$R_{System}(t) = \prod_{i=1}^n R_i(t)$$

Fill in the missing numbers below to work out what the valve-pump series subsystem reliability is.

$$R(200) = \prod_{i=1}^{\square} R_i(200) = \square \times \square = \square$$

The figure above is the PROBABILITY OF A SYSTEM FAULT NOT PROPAGATING THROUGH THE CORRESPONDING 'OR GATE.'

We can also calculate the PROBABILITY OF A SYSTEM FAULT PROPAGATING THROUGH THE CORRESPONDING 'OR GATE.'

$$F(200) = 1 - R(200) = 1 - \square$$

$$= \square$$

This allows us to know the reliability and failure probability of each valve-pump series system at 200 years.

EXERCISE

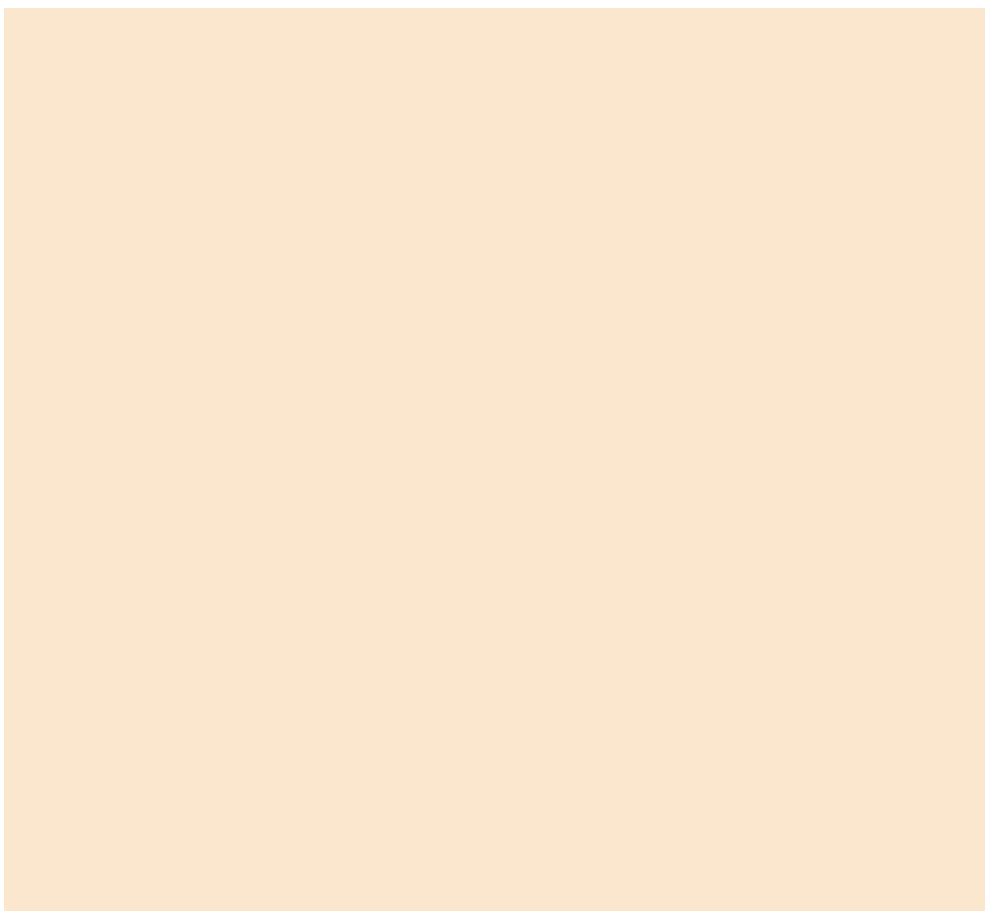
The next part of the system we will look at is the other valve-pump series subsystem.



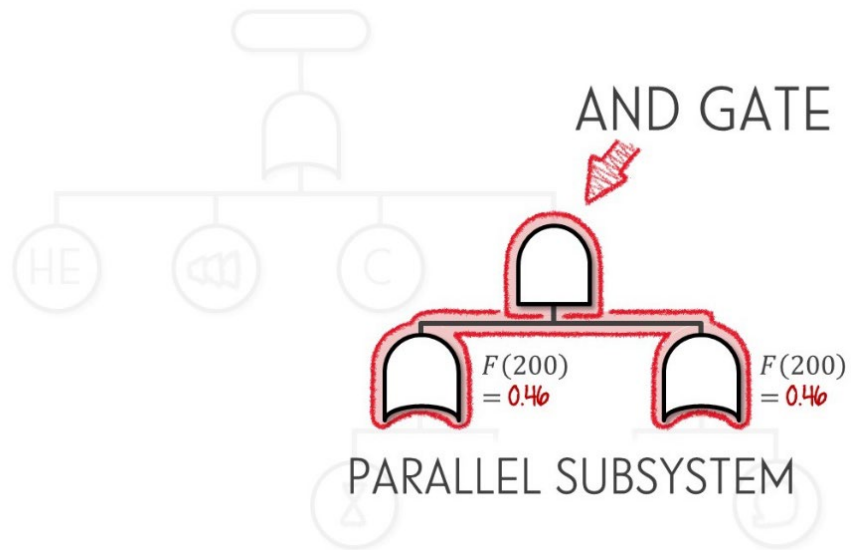
This series subsystem is exactly the same (in terms of reliability characteristics) as the valve-pump series subsystem on the previous page.

So we don't need to analyze this further.

Now that we have analyzed these two valve-pump series subsystems,
WHAT IS THE NEXT 'LITTLE CHUNK' WE SHOULD ANALYZE?



We now need to keep moving up our fault tree. This means we now focus on the next 'AND Gate' up, which models a parallel subsystem.



We know for a PARALLEL system that failure probability is ...

$$F_{System}(t) = \prod_{i=1}^n F_i(t)$$

Fill in the missing numbers to work out what the ENTIRE main and auxiliary valve/pump subsystem failure probability is ASSUMING INDEPENDENCE.

$$F(200) = \prod_{i=1}^{\square} F_i(200) = \square \times \square = \square$$

The figure above is the PROBABILITY OF A SYSTEM FAULT PROPAGATING THROUGH THE CORRESPONDING 'AND GATE.'

We can also calculate the PROBABILITY OF A SYSTEM FAULT NOT PROPAGATING THROUGH THE CORRESPONDING 'AND GATE.'

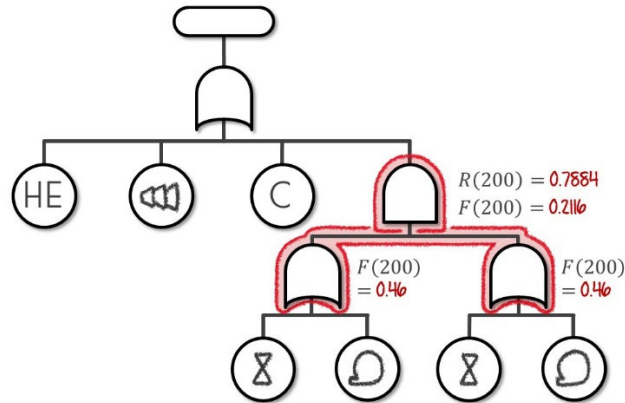
$$R(200) = 1 - F(200) = 1 - \square$$

$$= \square$$

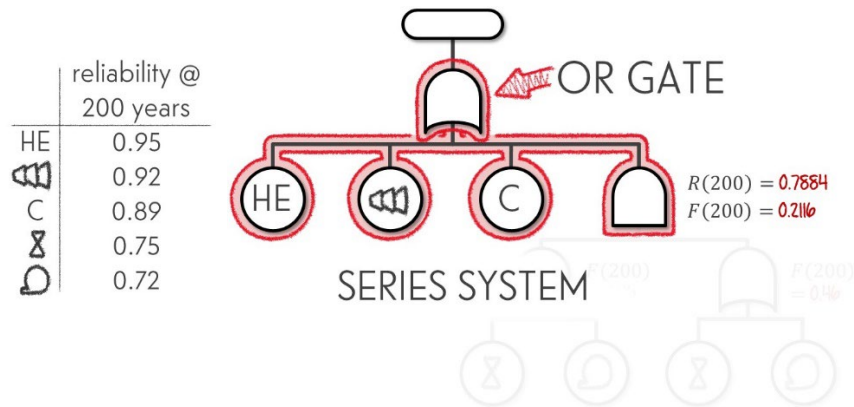
This allows us to know the reliability and failure probability of the ENTIRE main and auxiliary valve/pump subsystem at 200 years.

EXERCISE

We can now take the next step to work out **WHAT IS THE RELIABILITY OF OUR SECONDARY COOLING LOOP AT 200 YEARS.**



We can now keep moving up the fault tree, to focus on the top 'OR gate' ...



We know for a **SERIES** system that reliability is ...

$$R_{System}(t) = \prod_{i=1}^n R_i(t)$$

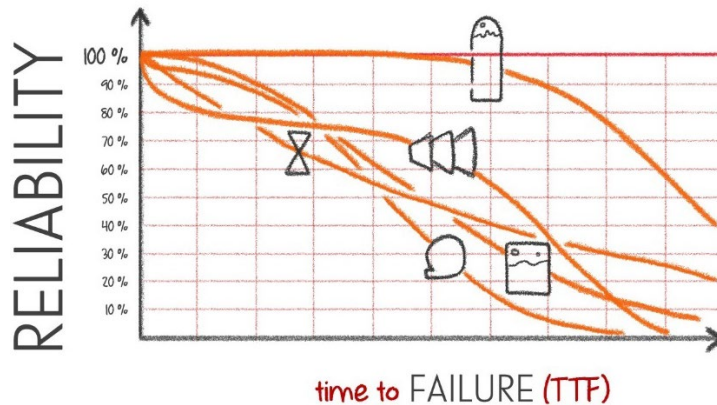
Fill in the missing numbers below to work out what the valve-pump series subsystem reliability is.

$$\begin{aligned}
 R(200) &= \prod_{i=1}^n R_i(200) \\
 &= \square \times \square \times \square \times \square \\
 &= \square
 \end{aligned}$$

This is the **SECONDARY COOLING LOOP RELIABILITY** at 200 years.

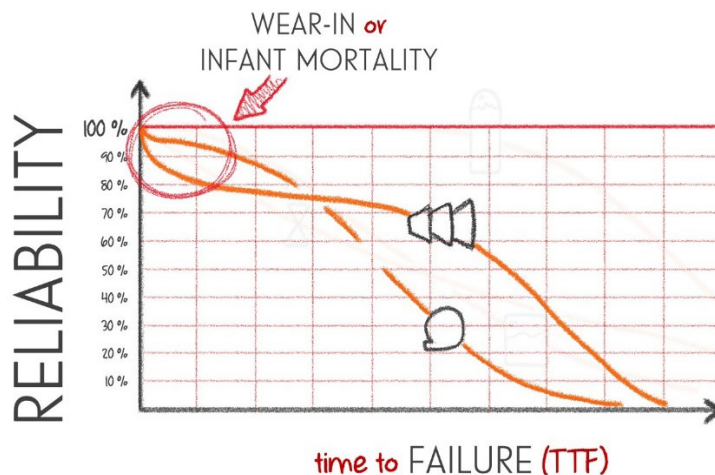
Learning HOW Things Fail from RELIABILITY CURVES

We can use this approach for any time throughout our nuclear power plant's life to turn **component reliability** into **system reliability**. Let's say the following curves represent our estimates about the SECONDARY COOLING LOOP's **component reliabilities** over time.



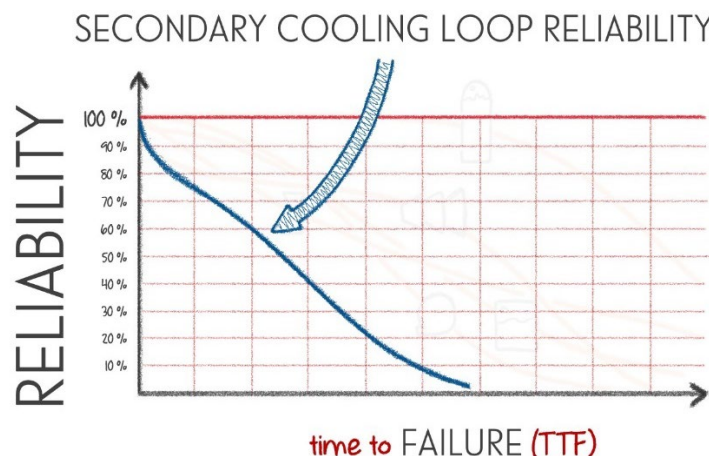
As is the case for every **reliability curve**, the curves start at 100 % and then decrease over time toward 0 %. This reflects the fact that as components get older, the chance of them having **failed** increases.

But the shape of each **reliability curve** contains a lot more information. For example, some of the **component reliability curves** start with an initial 'dip.' This represents **wear-in** or **infant mortality**.



This part of the **reliability curves** for the turbine and pump describes the nature of early **failures** that are typically caused by things like manufacturing defects. As the fraction of components that have these defects have all failed, the apparent **reliability** of the remaining 'defect-free' components does not drop as quickly.

We can see this in the secondary cooling loop **reliability curve**. Using the approach we used in the worked exercise; we can create the following **curve** that represents **system reliability**.



...and that's it for SYSTEM RELIABILITY ANALYSIS

You have now learned how to complete **fault tree SYSTEM RELIABILITY ANALYSIS!** Well done! Of course, there are much more complex systems out there, but the principles you have learned about are exactly the same. You may just need to take more steps for a more complex system!" Or something similar.

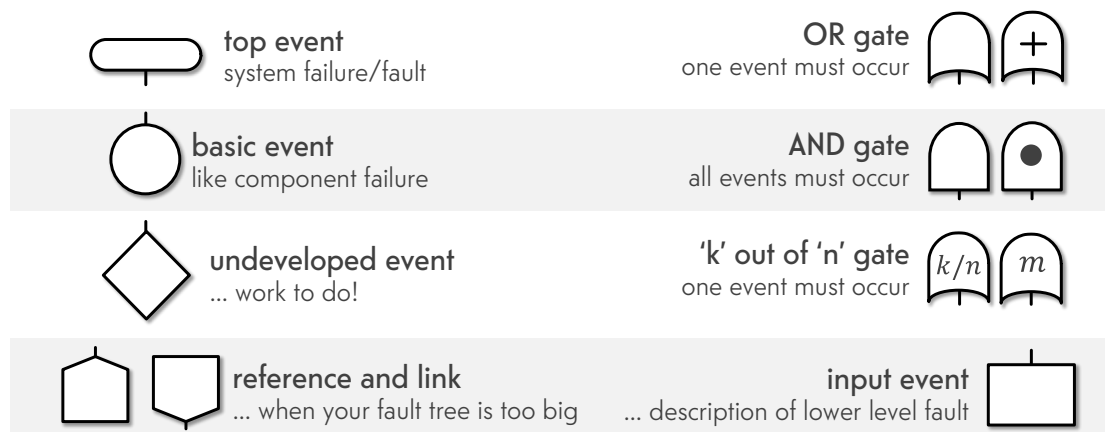
There are lots of software packages out there that can help you do this. But it is always important to understand how software does things like **fault tree analysis**. If you blindly trust software to do things you don't understand, you will inevitably do something you never intended and get results that might cost (perhaps) millions of dollars. Software is only as sophisticated as the person using it. And because you now know what the software is doing, you can get really sophisticated results. Each software package should have its own training package or manual to teach you how to create **fault tree models**, but they will never be as thorough as courses like this one when it comes to the theory.

But don't forget, **fault tree analysis** always starts with understanding the decision you are trying to inform. Using **fault trees** to conduct **system reliability analysis** will help you make decisions where it is important to know things like **warranty reliability** or work out what are the VITAL FEW components driving **system reliability performance**. In short - **MEASURING reliability**.

The next chapter starts focusing on how to **IMPROVE reliability** through **root cause analysis (RCA)**. **RCA** doesn't focus on **measuring reliability**. Instead, we want to identify the VITAL FEW **Corrective Actions (CAs)** that we need to implement in order to get the biggest increase in **reliability** for the most efficient use of time or resources. And, as a rule: **IMPROVING reliability** is way more important than **MEASURING reliability**.

All the FAULT TREE SYMBOLS Together

A summary of typical **fault tree** symbols is listed below. Note that the '**reference** and **link**' symbols are sometimes replaced with triangles. And, sometimes the '**k out of n**' gate is simply represented with a single letter '**m**' which is equivalent to '**k**' (the number of functional components required for the subsystem to be functional).



Fault Trees and Root Cause Analysis (RCA)

Chapters 2 – 10 looked at **fault trees** from the perspective of analyzing system reliability. But **fault trees** are also very useful for **Root Cause Analysis (RCA.)** Remember that **RCA** is all about trying to find the reason a failure has occurred. **RCA**'s more 'textbook' definition is:

... a method for identifying the root causes of failure, faults, failure modes, defects, errors or any other event associated with a system not performing as desired.

But perhaps a simpler understanding of what **RCA** nominally does is to help us find out ...

WHY THINGS FAIL?

But, this is not the ultimate aim of **RCA**. If we stop at understanding *why* things fail but then don't use this information to improve safety or reliability, then **RCA** has been a waste of time. So, in practice, **RCA** is all about working out:

what can we do to STOP THINGS FAILing?

So What is a ROOT CAUSE?

Many organizations and 'experts' really don't understand what a **root cause** is. Many textbooks and standards say things like:

... a ROOT CAUSE is the initiating cause of failure.

But this is not helpful, as the 'initiating' cause of failure is completely subjective. Is it the mistake the manufacturing engineer made that resulted in a defect being included in a component that led to premature failure, OR inadequate training the manufacturing engineer received that led to this mistake, OR the floor manager who is over-burdening his or her staff in a way that is forcing them to cut corners? A much more useful way of looking at a **root cause** is

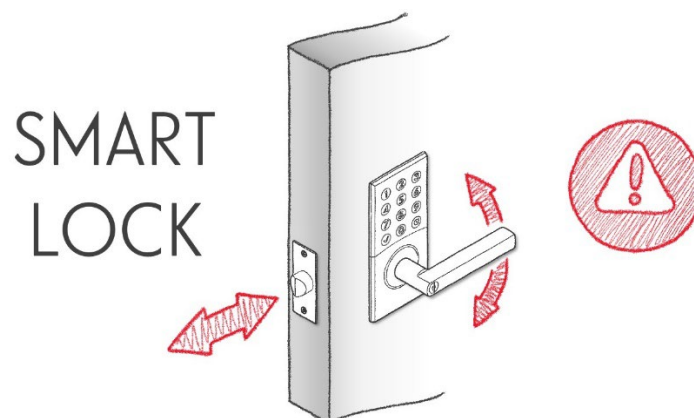
... a ROOT CAUSE is the THING we can change to make sure 'that' failure hardly happens again ...

This means a **root cause** is a (potentially embarrassing) behaviour YOU EXHIBITED that you have the ability to change. You can't change the weather, school system, or customer behaviour. So, these aren't **root causes**.

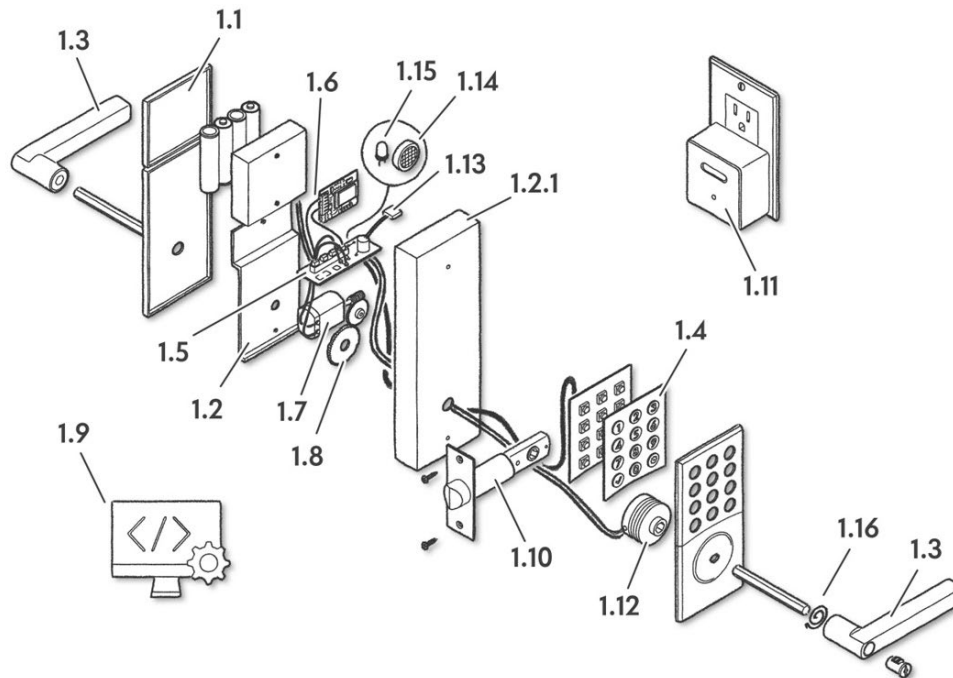
Let's Do Some RCA on a Smart Lock...

Below is an illustration of a 'smart lock,' with a problem. A smart lock is a great example of a modern fusion of traditional mechanical lock mechanics with emerging 'smart' technology. A smart lock can communicate with smart phones and computers in a way that allows users to remotely unlock and lock a door. You can create temporary pin codes to allow cleaners, tradespeople or guests access through your door for certain periods of time. It can also detect how close you (and your smart phone) are, to automatically unlock when you are carrying heavy groceries.

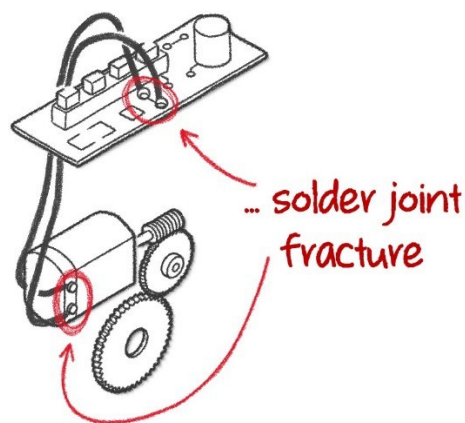
But, the production prototype below stopped working during a user demonstration. Now we need to work out why.



One of the first things we do is pull our failed system apart to try and see if there is anything visibly wrong. You can see that our smart lock is made up of several very different elements that include mechanical components, Printable Circuit Boards (PCBs), batteries, electric motors and gears, software, and electronic components.



When we fully disassemble the smart lock, we find that the solder for the cable that connects the electric motor to the PCB has fractured.



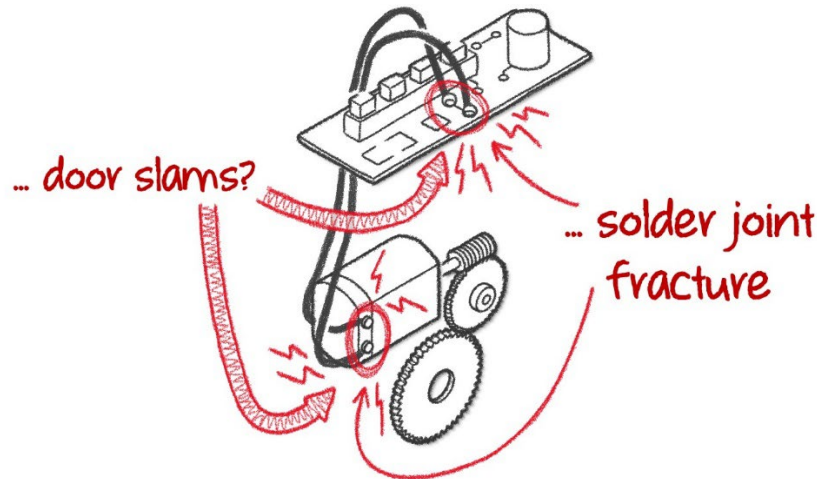
Is this the ROOT CAUSE of failure?

If fractured solder was the **root cause** of **failure**, then we should simply be able to correct the design flaw that led to this failure by removing the 'fractured solder joints.' But this is not a **Corrective Action (CA)** – this is a **repair** or **Corrective Maintenance (CM)**. Resoldering the cables would ensure that *this* smart lock would work again. But it would do nothing to remedy the underlying factor that led to solder joint fracture in not only this smart lock, but potentially every single smart lock the manufacturer produces. Perhaps a 'soldering expert' would be able to look at the fractured solder joint and deduce what needs to change – but this is only because that

expert is conducting his or her own **RCA** in his head based on his experience to come up with the 'true' **root cause**.

So, we need to keep asking questions.

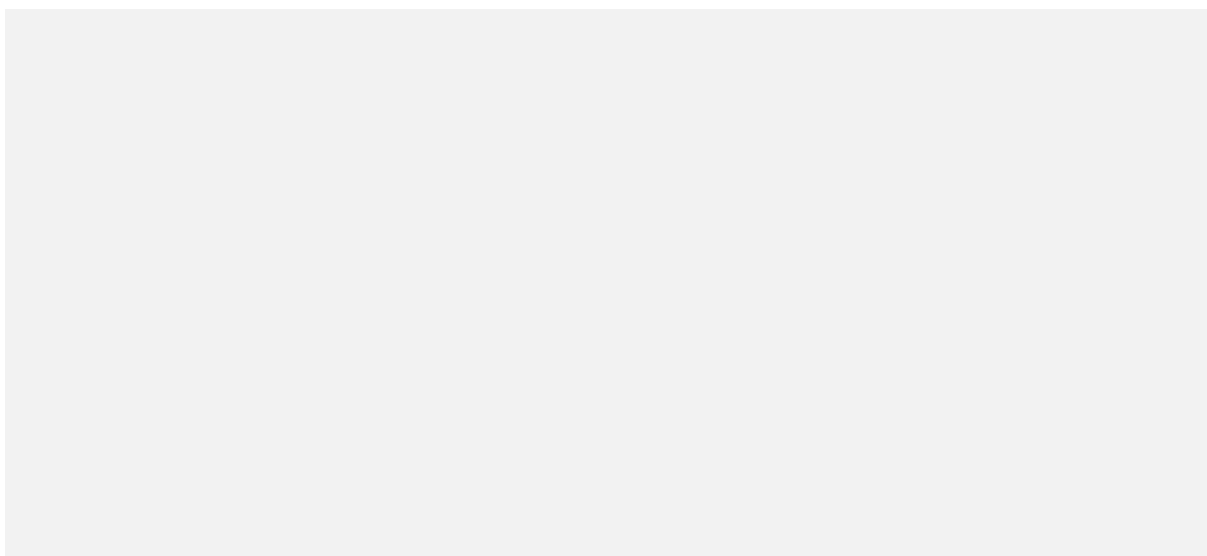
When we review the test profile for the smart lock, we see that some of the test serials involved installing the smart lock onto a test door that was repeatedly exposed to 'door slams.' That is, the door was forcibly shut with a special machine to replicate the type of door slams that at least some of our future customers would subject our smart lock to.



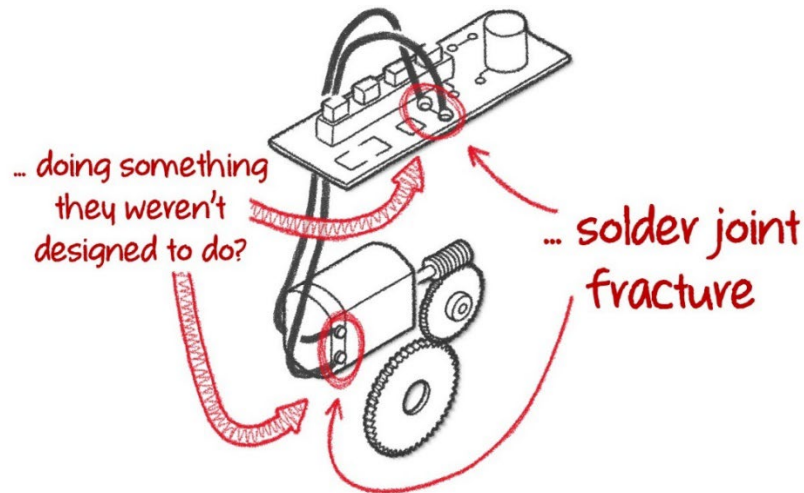
Are DOOR SLAMS the ROOT CAUSE of failure?

If door slams are the **root cause** of failure, then the corresponding **CA** would be to 'eliminate door slams.' But how can we do this? Can we ask customers to ensure that they don't slam the door with our smart lock fitted to it? Will this inspire confidence in an increasingly competitive marketplace? If a customer does indeed slam a door in spite of our clearly stated requirements to not slam the door, can we somehow reject their warranty claim? And how can we find out if they slammed the door when they claim they didn't? So, door slams are not the **root cause**.

We need to keep asking more questions.



Just because door slams are not the **root cause**, it doesn't mean that we are not on the right track. Slamming the door will transfer forces to the solder joints within the smart lock. The circuit board and motor are firmly secured within the housing. But the cables are not. This means that the solder joints are the only thing holding the cables in place. This is often not a problem ... provided there is not a lot of shock loads being applied to the cable. But this is not the case when a door is slammed. And solder joints are not designed to be able to secure cables in this way.

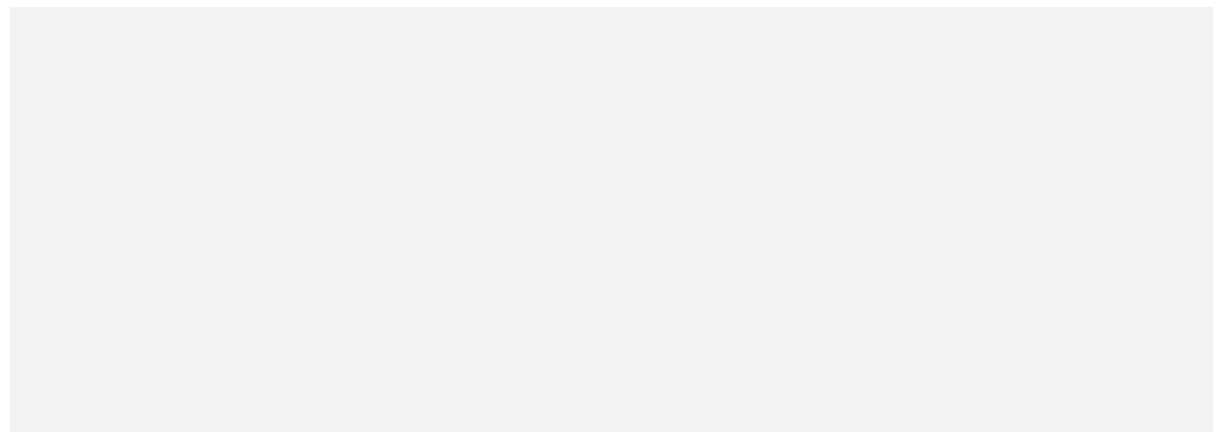


Is the SOLDER JOINTS being 'asked' to do something they were NOT DESIGNED TO DO the ROOT CAUSE of failure?

If the answer is yes, then we can perhaps reword the root cause as

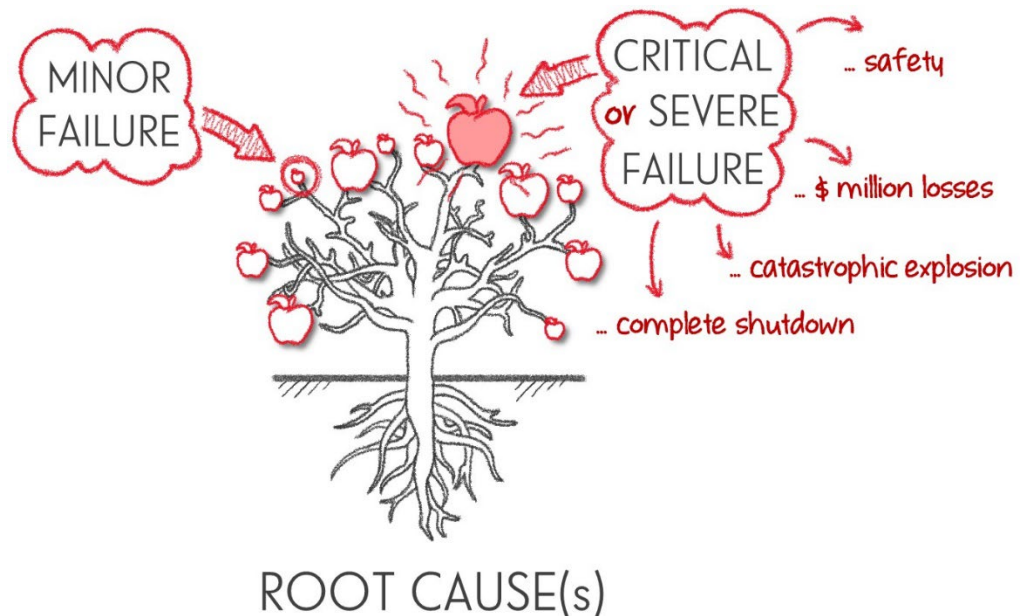
... design being INCAPABLE of tolerating shock loads at cable connections

This IS something we can address. We 'own' the design. We can do what we want with the design. We can't individually repair every fractured solder joint that fails on our customers doors, nor can we ask our customers to not slam the doors. So, this IS something we can address. Some organizations may want to keep going and investigate WHY the design was 'allowed' to be incapable of withstanding the shock loads. Some other organizations might stop investigating here and start coming up with **CA**s. As long as they start 'talking' about things we are able to address, then each organization will likely come up with fantastic **CA**s that will quickly, efficiently and cheaply eliminate this **root cause** of failure.



The TREE OF FAILURE

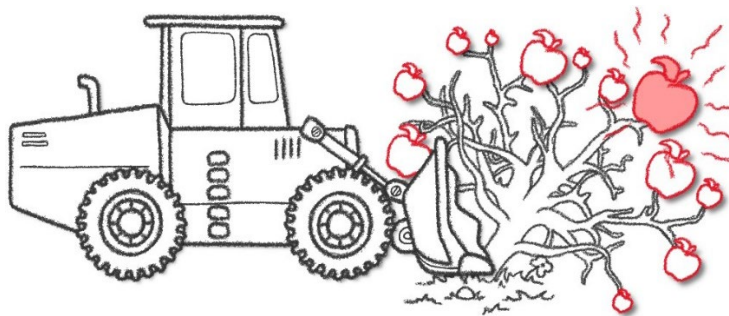
A **root cause** is something that you own, or something you can do something about. The term 'root' comes from the popular way we visualize 'how bad things happen,' derived from the basic structure of a tree. And that way we visualize comes from a tree.



Most trees have more than one root. But the term '**root cause**' is a singular reference that is written as if there is only one, 'true' **root cause**. In reality, there are often lots of different **root causes** that combine to result in the undesirable event we are analyzing. And by identifying lots of potential **root causes**, we might be able to choose the **CAs** that are the cheapest, easiest and fastest to implement.

Then There Are IMMEDIATE ACTIONS

Whenever an undesirable event (like **failure**) occurs, we usually need to do something quickly to contain its consequences. This is like removing the 'visible' part of the tree above the ground.



The problem is that if we do not address the **root cause(s)** of **failure**, the tree (and the fruits that represent failures) will simply come straight back. This doesn't mean that we shouldn't do

something quickly to contain those consequences. It's just that these will be a short-term solution. And we call these short-term solutions **'immediate actions.'** Examples include ...

... *CORRECTIVE MAINTENANCE (CM) or REPAIR* where we address the physical and functional issues that result in system or equipment failure ...

... *ISOLATION* where the failed system or equipment is removed or separated from the rest of the assets to ensure that its failed state doesn't affect operations ...

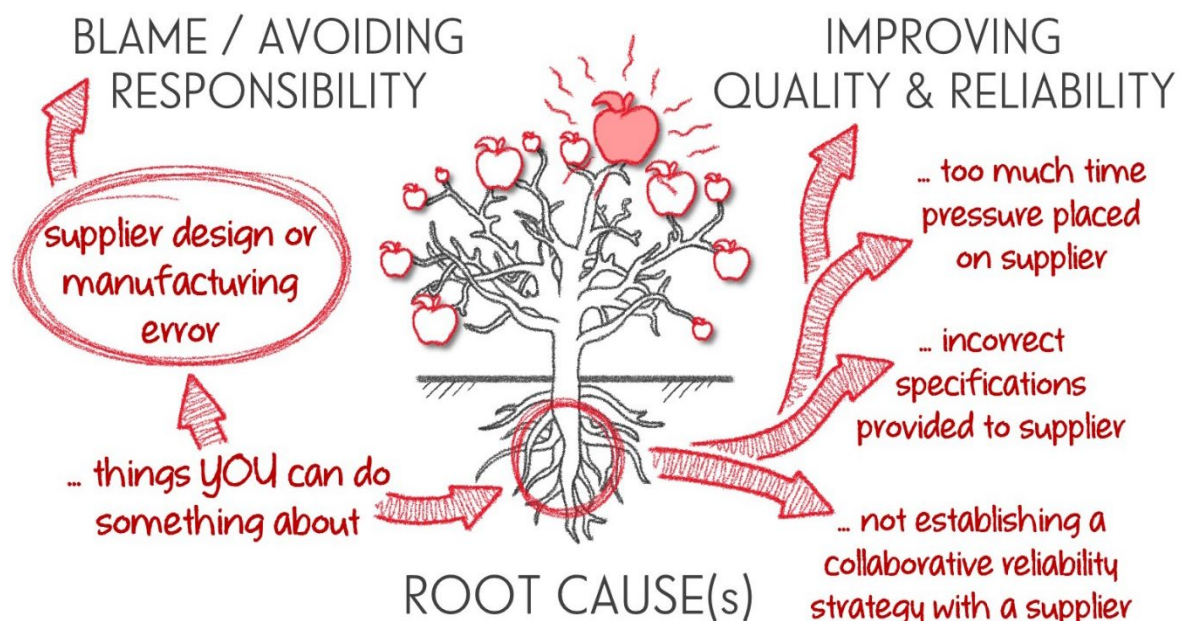
... *CONTAINMENT* where mechanisms or structures are placed to ensure the consequence of failure doesn't cause further damage or harm ...

... *RECALL* where a manufacturer or supplier takes the systems or equipment back from customers in order to implement actions that will mitigate further failures ...

There are lots of others that are specific to each scenario and are usually necessary steps before more detailed **RCA**.

ROOT CAUSES Are All About What 'WE' Can Do

Truly wanting to find **root causes** of **failure** usually means we need to be humble about what went wrong.



And many **root causes** emanate from a set of behaviours we can associate with 'characters' that we label the 'Enemies of Quality and Reliability.' They are not specific people, but help us understand the sort of behaviours we want to avoid.

The first enemy is the PONDEROUS PROFESSOR.

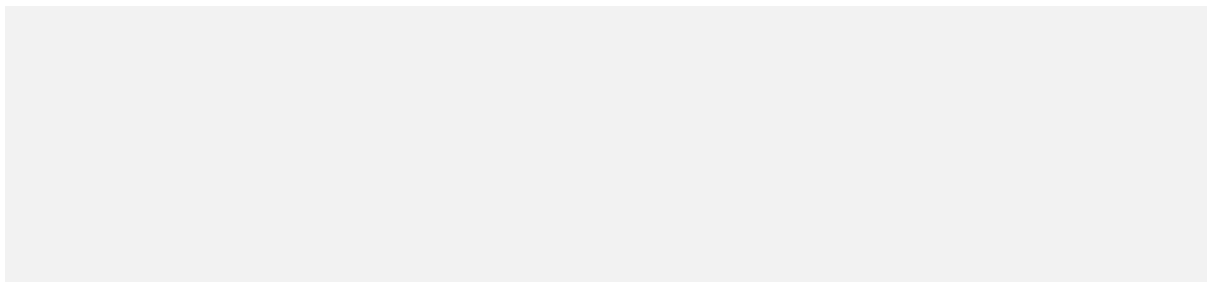


He prefers to measure **quality** and **reliability** instead of improving them. He tends to not be 'clever' in terms of design or manufacturing, and ensuring **root causes** are never allowed to exist. Instead, he wants to make decisions based on a preponderance of data and is perhaps skilled at analysing this data. But because he is not comfortable with making subjective assessments, his analysis will often be too detailed, too late, and too complicated to understand.

Then there is the INFANT MANAGER, with his favourite catchphrase...



WTF stands for 'Wrong Thing Fast.' Tomorrow's problems are never today's priorities as he is always focused on rushing to find the next milestone as quickly as possible. This comes at a cost, as tomorrow's problems will quickly be realized. And when this happens, he is always looking to transfer responsibility and accountability to someone else, so the blame never falls on him.

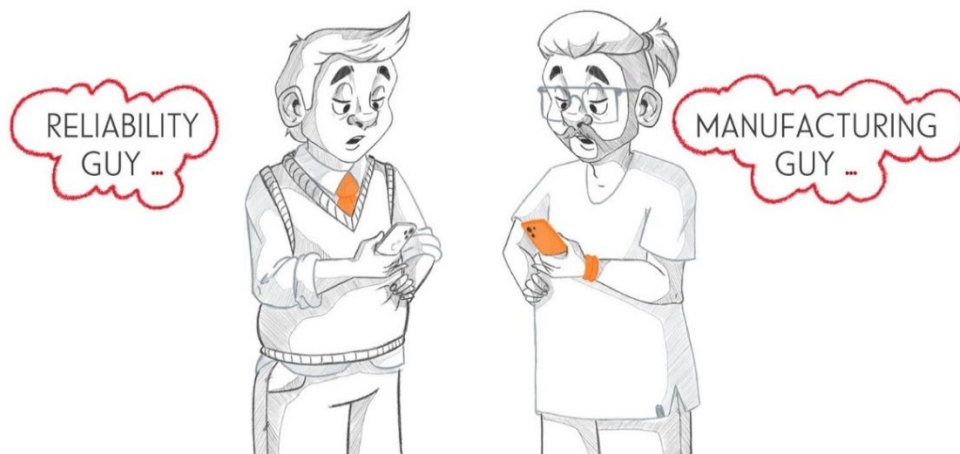


Then there is the PROCESS ZEALOT.



She doesn't care how well something is done as long as it follows a process. This means she is not interested in testing to learn anything... she wants you to test to pass. There is limited flexibility for continual improvement because the process is so central to everything that changing it becomes an overwhelming challenge. So, the process reflects other items or systems, including our own items or systems that we built several years ago.

Then there are the TWO NOT ME ENGINEERS, one is a reliability guy, and the other is the manufacturing or quality guy ...



These gentlemen are always trying to deflect the blame to the other. The reliability 'guy' is always looking to attribute failures to manufacturing defects or quality control. The manufacturing 'guy' is always trying to attribute failures to the design, or the design making it challenging to manufacture.

And the manufacturing guy often has a point:

... around 80 % of 'quality problems' that we tend to see as manufacturing defects can be traced back to DESIGN ISSUES ...

It is true that many designers don't think a lot about appropriate tolerances, or designing things in ways that are easy to manufacture.

And so many **root causes** look like these characters.

The Reliability Mindset

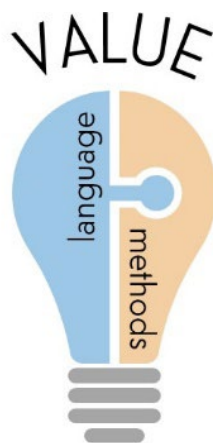
We often don't have the ability to 'see' what goes wrong. For example, we might launch our smart lock, and then start receiving more warranty claims than we thought we should have. We might not get the physical locks back, or even a description of what went wrong.

But that is where structured, well facilitated **RCA** can be incredibly helpful. Many engineers believe that **RCA** is all about finding the 'provable,' single, 'true' **root cause** of **failure**. This doesn't happen very often. Instead, we need to be happy to find a suite of likely potential **root causes**, and then prioritize the **CAs** we implement for each.

The earlier you do this (through things like **FMEAs** and early production **FTAs**), the more flexibility you have to choose **CAs** that are easier and cheaper to implement.

Lots of different approaches and techniques have been developed, along with their acronyms (5Ys, 5WHYs, 5WHYs & 3 tiers, Statistical Process Control or SPC, 8Ds, 4/5/6/8Ms, 5W1H, DMAI, Apollon, Pareto Analysis, Ishikawa Diagrams, and plenty more).

But the most important element of **RCA** is the **quality and reliability mindset** which we will illustrate with the light-bulb icon below.



The '**language**' part of the light bulb refers to the common set of term and concepts we all share, so that when one person is brainstorming as part of **RCA**, they are not confusing or counteracting the thoughts and words of others. The '**methods**' part of the light bulb refers to the structured approach we must use to ensure our brainstorming efforts quickly and efficiently focus on the most likely **root causes** we need to address. And finally, we must always understand the '**value**' of what we are trying to achieve to understand what is worth doing, and what is not. And most importantly, being able to understand how much 'value' our **CAs** will generate.

So How Do We DO THIS RIGHT (...Not WRONG)?

There are some basic things we need to do to get a good, valuable outcome of **RCA**.

1. State the PROBLEM (and the decision you are able to make)

This needs to be set in reality. For example, if **RCA** is left too late... don't do it! This cheapens and diminishes **RCA** in your organization and associates it with being a waste of time. Therefore, not understanding the problem, or decision you are trying to inform, dooms any **RCA** to mediocrity or irrelevance.

2. Work through LAYERS (and not rushing to your FAVOURITE solution)

Most **failures** (or undesirable events) are the result of many possible **chains of events**. These chains can branch (or root) out, and quickly following one **chain of events**, will result in a small number of **potential root causes** that might not actually represent the true reason behind something **failing**. Instead, we need to go down 'one layer at a time' and brainstorm all possible preceding **events**. Each new event we brainstorm then becomes the start of a new **chain of events** we need to pursue. While this sounds like more work, the result is lots more potential **root causes** (and **CAs**) for us to choose from, including many that will be faster, cheaper and more effective.

3. Don't be a LAWYER (who wants to find a SINGLE ROOT CAUSE to allocate BLAME)

So, what if the 'electric motor' is causing **failure**? What if the easiest **fix** involves changing the design of the gearset? This happens a lot, where one component or subsystem isn't behaving the way we like, and the easiest, fastest, cheapest and most efficient **CA** involves modifying the design of another component or subsystem. The same thing applies to suppliers. Yes, it might feel 'just' to apportion blame to a supplier whose component is not behaving. And the money they spend on remedying the issue will eventually be paid for (by you). There will of course be some cases where suppliers need to be accountable for mistakes they might make, but there is a reason why lawyers don't make great designers or manufacturers.

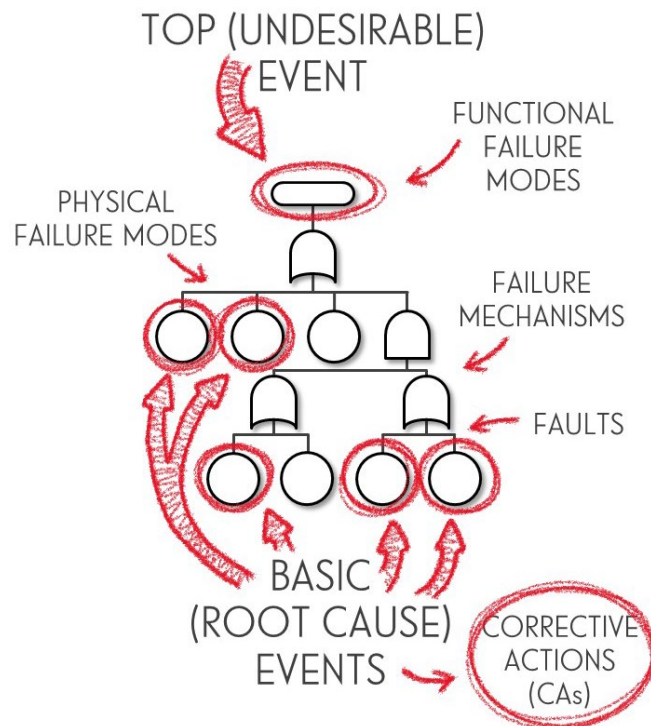
*4. Find LOT's OF PROBABLE ROOT CAUSES
(so we can pick the cheapest and easiest ones to fix).*

5. You aren't done until you FIX the ROOT CAUSES you find and select

RCA is not about admiring problems. It is about finding **CAs** and then managing them until they are validated. In some cases, the probable **root causes** we identify **CAs** for might not end up being the 'true' **root causes**. So, we then need to move onto the next most probable **root causes** and identify **CAs** for them. We keep going until the failure has been addressed and prevented.

Root Cause Analysis of Our Smart Lock

We started to look at fault trees not as ways to represent a system reliability model, but as a tool to help us conduct **Root Cause Analysis (RCA)**. **RCA** can be conducted on any **top (undesirable) event**, but when it comes to reliability it helps if we incorporate some common definitions as we move down our **fault tree**.



The **basic events** at the bottom of our **fault tree** represent a single **root cause**, which is some behaviour we can influence. And from each **root cause**, we can generate a number of **Corrective Actions (CAs)** to ensure that the **undesirable event** (which is often **failure**) hardly happens again.

You often waste your time trying to find the 'TRUE' root cause.

It is often impractical, time consuming, expensive and NOT WORTH trying to find the '**TRUE**' **root cause** of **failure**. This is because there are often multiple contributing factors that led to a **failure** occurring in addition to any '**MISTAKES**' that a designer, manufacturer, engineer or supplier made. And what a '**MISTAKE**' is can be very subjective. The benefit of focusing on all contributing factors when thinking about the **root cause** is that you get a wider range of potential **CAs**. This means you are more likely to find cheaper, faster fixes when you stop trying to apportion blame to the person who made a '**MISTAKE**' and is therefore the elusive '**TRUE**' **root cause**.

So instead of looking for the 'TRUE' root cause so that you find someone to blame, try and find a number of LIKELY root causes that allow lots of different people and teams to come up with CORRECTIVE ACTIONS (CAs)

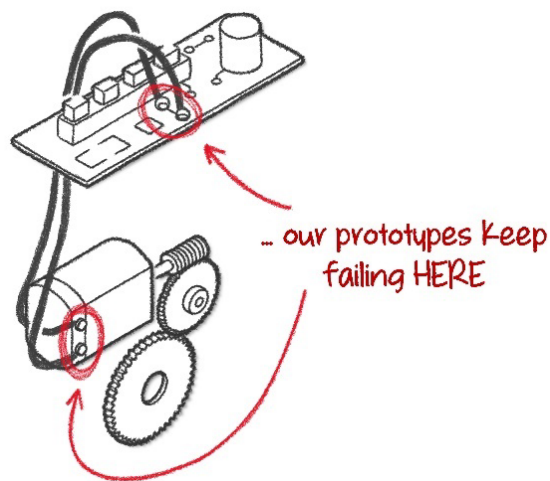
You often waste your time RUSHING to find root causes.

RCA is best completed when you do it ONE STEP AT A TIME

When you work down to **root causes**, you want to identify as many plausible explanations for each step of the causal chain to allow as many branches (or roots) as practicable. This gives a wider and more useful range of **potential root causes** and not just our FAVOURITE **root causes**. If we have a wider range of **potential root causes**, we again have the opportunity to select from a wider range of **CAs**, so we can find the cheapest, quickest, most efficient fixes for our **failure**.

The SMART LOCK

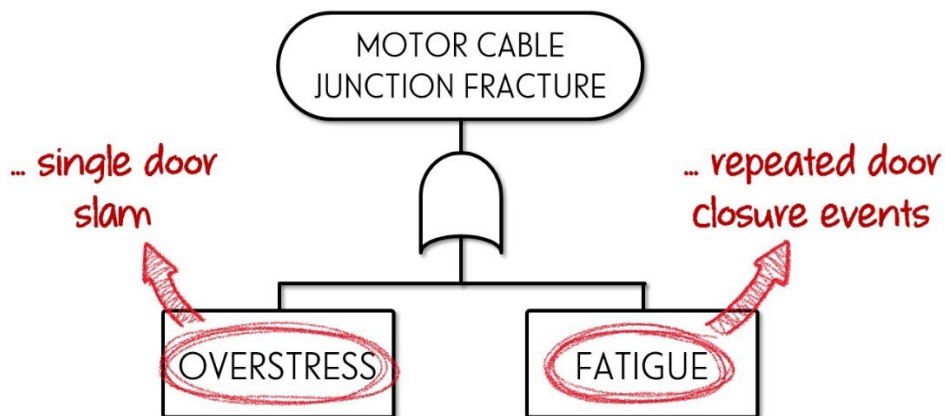
Let's take the **fault tree** techniques we have learned and apply them to the smart lock, starting with the point in time where we learned that the prototypes were **failing** due to solder joint fracture at the cable that connects the electric motor to the circuit board.



If we can identify the **physical failure mode** (solder joint fracture) then we tend to make that the **top event** of our **fault tree**.

MOTOR CABLE
JUNCTION FRACTURE

We now brainstorm the next 'layer' down to find as many immediate, plausible reasons our top event might happen, and then include them in our **fault tree**.

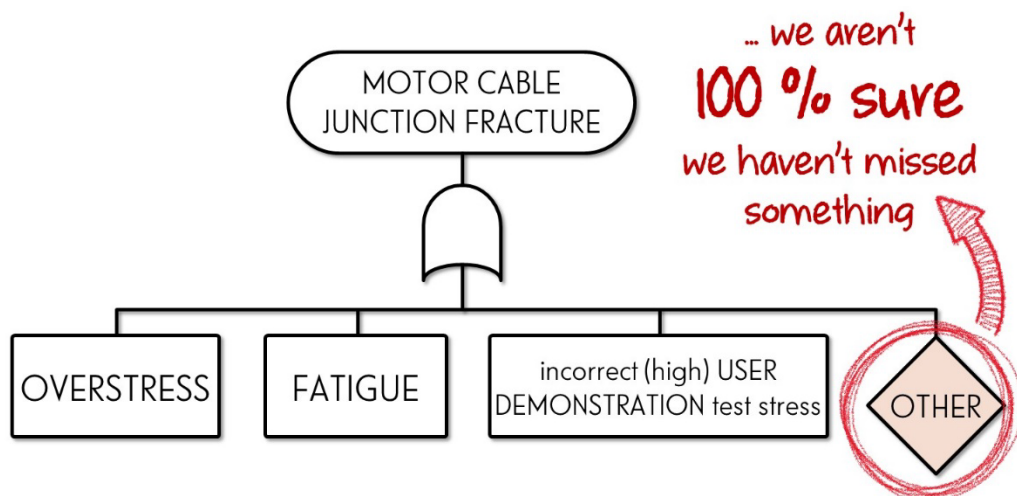


You can see how important it is to include all possible **events** as they each imply different things (single door slams versus repeated door closures). Each will require different **CAs**, so if you miss one **event** you might miss the **CA** you need to resolve the problem.

At this stage, you might be able to send the failed smart lock to a laboratory for more detailed **Failure Analysis (FA)**. This is where technicians and scientists use microscopes, x-rays, ultrasound, etching and anything else they need to determine whether failure was caused by overstress or fatigue. This can be time consuming, expensive, and sometimes non-conclusive. So, you need to make a decision about whether it is worth it.

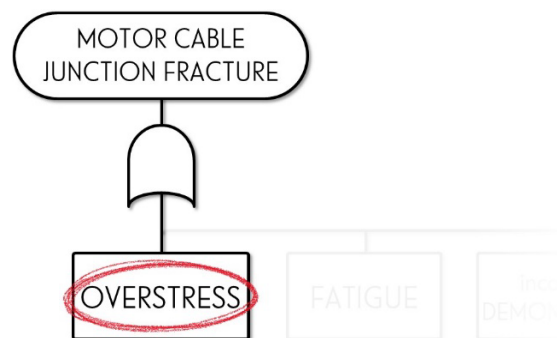
One factor that might influence your decision is ...*who cares?* Let's say that the **failure** was caused by fatigue. If we conduct **RCA** that includes overstress as a potential failure mechanism, our team will come up with wonderful **CAs** that might mitigate that **failure mechanism** quickly and cheaply as well. So don't artificially limit the scope of the **RCA** as you can use it to improve **reliability**.

If you ever feel like things are slowing down, simply add an **undeveloped event** to signify that there still might be some additional potential explanations, but we don't know what they might be. So, the **fault tree** under development now looks like ...

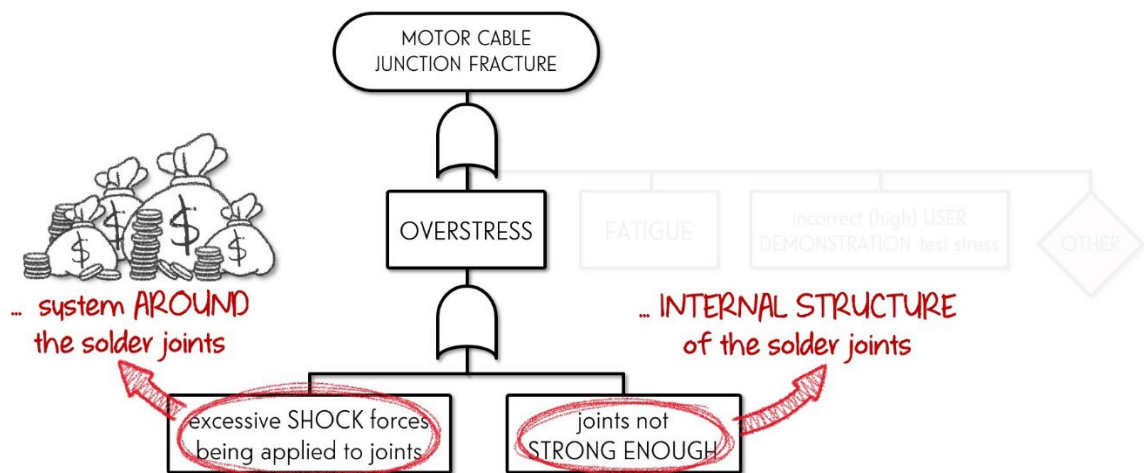


It is important to not discount anything. For example, identifying the input event '*incorrect (high) USER DEMONSTRATION test stress*' means that we might investigate how the test was carried out. If we find out that the test wasn't carried out correctly, we might be able to conclude there was no 'real' **failure** because the smart lock **failed** when exposed to stresses it would never experience in actual use. Finding this out might mean we terminate the **RCA** and maybe not implement any **CAs**. Even if the **CAs** are simple to implement, it is sometimes important to make sure we focus our organization's reliability 'attention span' on the **VITAL FEW** issues.

The next step is to identify one of the input events and analyze it further. For example, let's choose 'OVERSTRESS' and keep asking the set of questions that lead us to this **input event**.



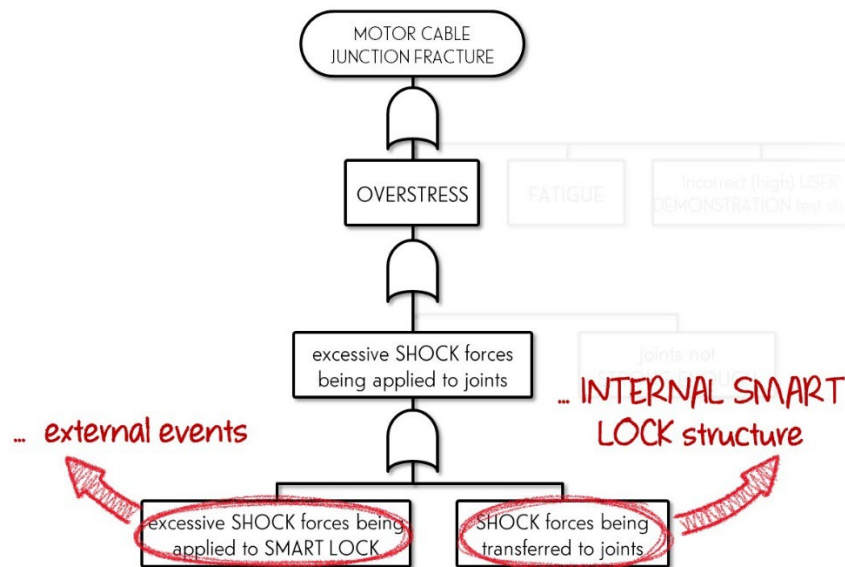
This question might be a little overwhelming. So **SLOW IT DOWN** and keep it simple. The brainstorming session might look at this question in terms of where the answer belongs. And this allows us to get the following **events**.



The two **events** identified above are quite different. The **event** '*excessive SHOCK forces being applied to the joints*' involves the entirely smart lock structure that is not dampening forces or otherwise protecting the cable from the forces the system experiences. You might reasonably expect that this will generate **CAs** that are more expensive. Conversely, the **event** '*joints not STRONG ENOUGH*' focus on the solder joints themselves. This will lead to an entirely different set of **CAs** that could feasibly be much cheaper.

Being methodical is always cheaper. The more potential **CAs** you find, the more choices you have to implement the cheapest, easiest one.

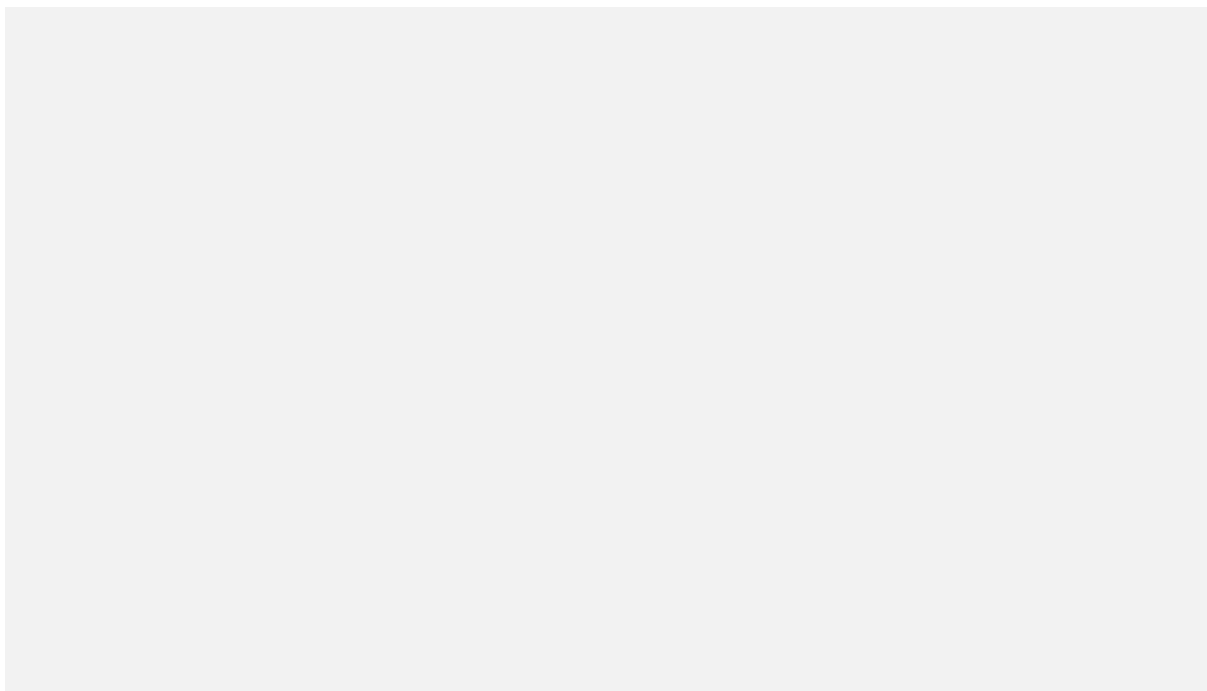
We now select one of the **input events** we came up with and analyze it further. Let's select 'excessive *SHOCK* forces being applied to joints.' And we keep brainstorming, slowly and methodically to get ...



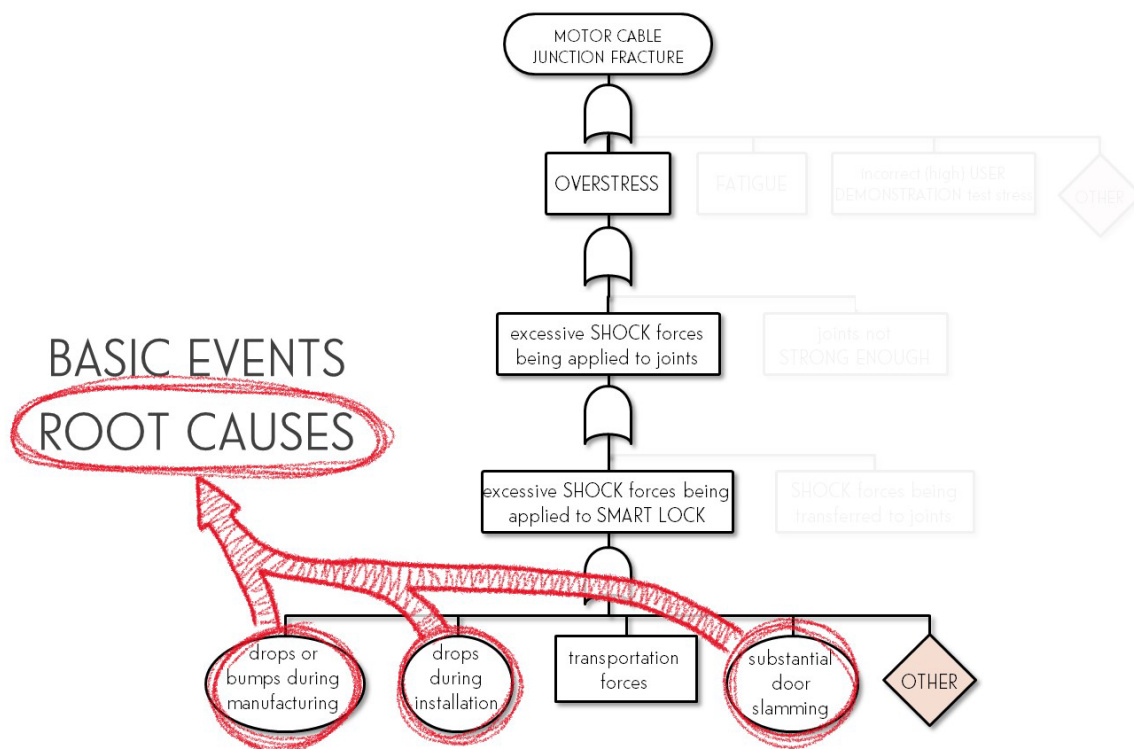
We are now breaking our **event** into more categories. The **event** 'excessive *SHOCK* forces being applied to SMART LOCK' involves something outside the system creating these forces. This includes customers or any other third party doing something to our lock. So, any **CAs** that emanate from this branch will focus on how we protect the smart lock from external forces.

The **event** '*SHOCK* forces being transferred to joints' is all about the internal smart lock structure and the dampening characteristics it has (or does not have). So, any **CAs** that emanate from this branch will focus on internal mechanisms to absorb shock.

We then choose one of these **events** and keep brainstorming.



If we choose 'excessive SHOCK forces being applied to SMART LOCK' for further analysis, we might actually get to events that we deem **root causes**. These are the things or behaviours we can influence AND for which we can identify a clear set of unambiguous **CAs**. If we do get to an **event** we deem a **root cause**, then instead of it being another **input event** (represented with a rectangle), it becomes a **basic event** (represented with a circle, oval or ellipse). You can see below that the brainstorming came up with four possible explanations for 'excessive SHOCK forces being applied to SMART LOCK' with three of them being classified as **root causes**.

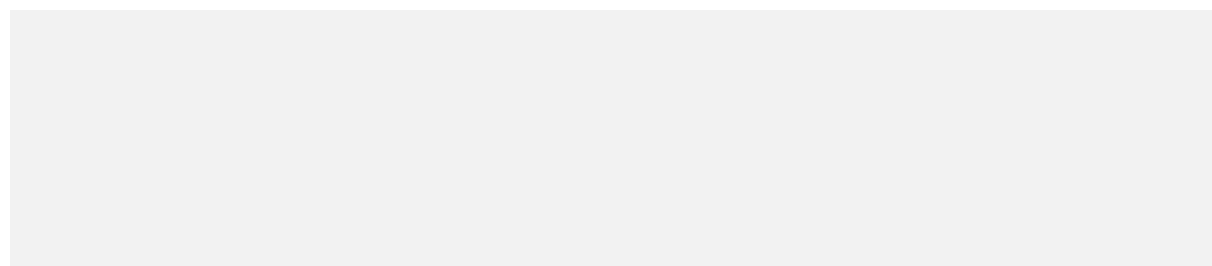


No **root cause** will be investigated further as they now become the basis for **CAs**. But any remaining **input events** like 'transportation forces' will need further analysis until it finds all possible **root causes**.

It is sometimes challenging to work out if you have reached a **root cause**. One useful question to ask in order to help is ...

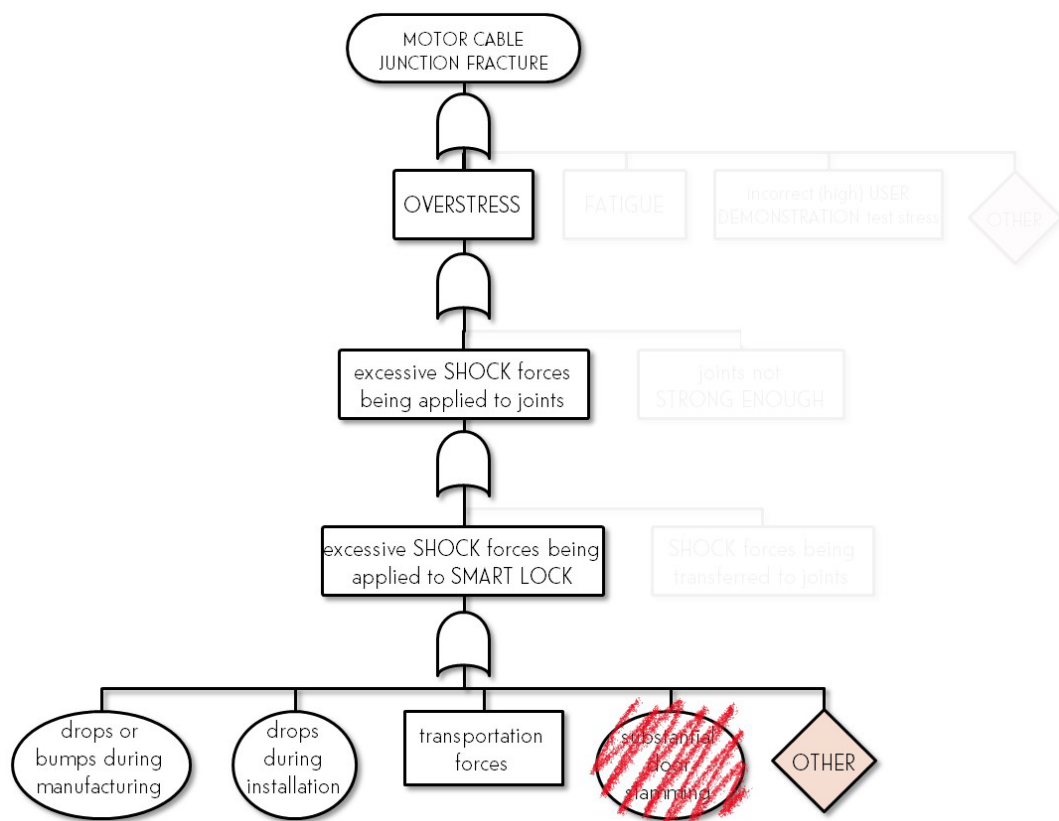
... are there clear **CORRECTIVE ACTIONS** (CAs) for this event that I think is a **ROOT CAUSE**?

If the answer is NO, then you probably haven't got to a **USEFUL root cause**. So, keep investigating and analyzing until you do.



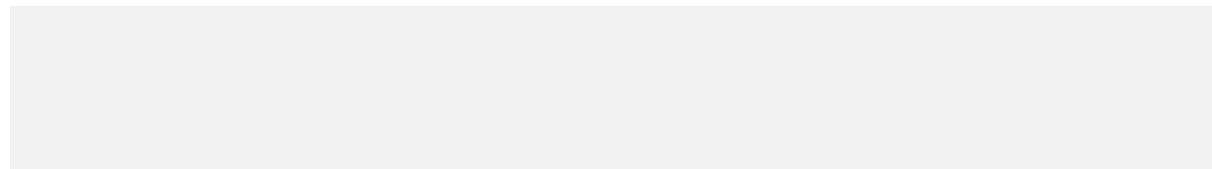
You might have noticed that one of the **root causes** on the previous page was '*substantial door slamming.*' You might also recall that in this and previous chapters, we talked about how we cannot (easily) influence customer or user behaviour. And if we can't influence their behaviour, how can '*substantial door slamming.*' be a **root cause**?

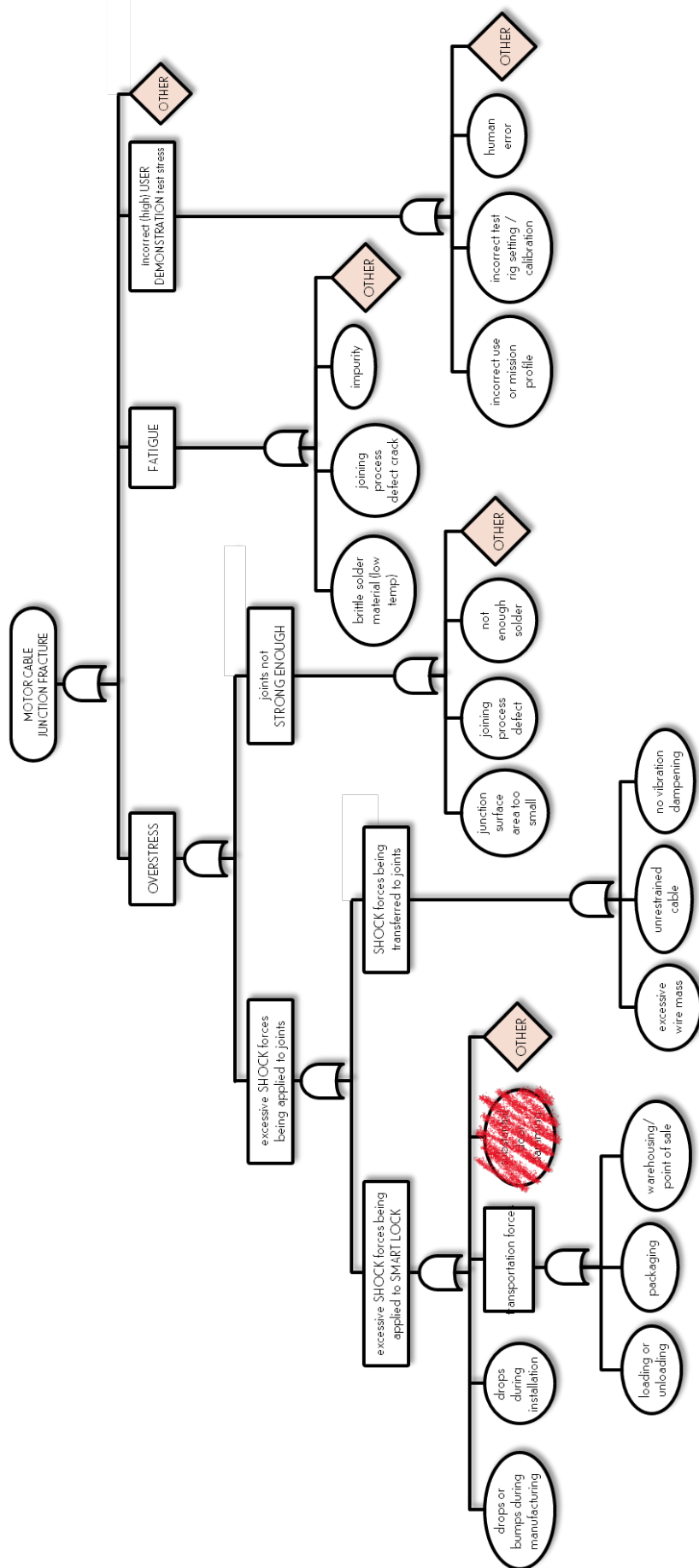
The answer is: it might not be. But it might have been important to include it in the **fault tree** to facilitate the group dynamic and brainstorming session. Generally, we don't want to rule out any suggestion people come up with. However, we might come back to the **root causes** we identify during analysis and see if they are truly something we can influence or not. If they are not, we might exclude them (as per the diagram below).



We might keep the 'crossed out' **basic event** in our fault tree along with an explanation so that anyone who reviews our **fault tree** in the future gets a full understanding of the group's brainstorming process.

We repeat this process of analysis for every input event until we have nothing but **basic events (root causes)** for every branch of your **fault tree**. The completed **fault tree** for the motor cable junction fracture is illustrated on the next page. A different brainstorming team might come up with a different set of potential **root causes**. THIS IS OK! They are all likely to have a positive effect on the failure we are trying to prevent moving forward.





Do we need the 'TRUE' ROOT CAUSE?

Sometimes there is pressure on us to find the 'TRUE' **root cause** and not a set of factors that all contribute to **undesired events**. Often there is no such thing as the 'TRUE' **root cause**. If (for example) an **RCA** found 10 'events' or 'factors' where if any one of them was removed, the **failure** would not have occurred ... which one of these is the 'TRUE' **root cause**?

Sometimes we feel pressured to find the 'TRUE' **root cause** in a very legalistic way ... which essentially means we want to apportion blame. In many cases this is not helpful but can often help us identify if outright negligence was involved. Even the US's National Transportation Safety Board (NTSB) usually identifies several 'causal factors' that contribute to each major aircraft crash they investigate, and these types of **RCAs** are very 'legal' in their framework and findings.

There may be scenarios where we do need to find the 'TRUE' **root cause** for genuine engineering purposes. For example, we might have identified that a **failure** was caused by either corrosion or fatigue. If this is being conducted on a large piece of machinery that is being operated by a utility (electricity generation company), then the cost of implementing any **CA** might be substantial because we are no longer at the early stages of design where any changes are often trivially cheap. So, they might want to know if it was fatigue or corrosion in order to deploy a **CA** that might involve an expensive plant shutdown.

But here, the **fault tree** is still very useful. Conducting **fault tree RCA** with brainstorming that identifies likely **root causes** will help any organization focus in on the most likely **root causes** they should be investigating further. If the smart lock manufacturer (for whatever reason) wanted to at least try and find the 'TRUE' **root cause** of the motor cable junction fracture, they now have a set of **potential root causes** that they can now prioritize for further, more detailed analysis.

But, because the smart lock manufacturer encountered this **failure** at the start of the design process, it is almost certainly better, faster, and cheaper to implement **CAs** that address all the **potential root causes**. And, they might also inadvertently fix other design problems as they do this.

*... it is very important for engineers, designers and manufacturers to not have
LEGALISTIC (BLAME-BASED) mindsets when it comes to RCAs ...*

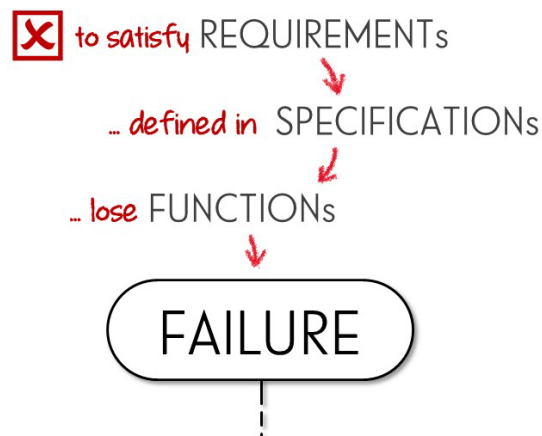
*... usually the cost of finding the 'TRUE' ROOT CAUSE (if it exists) is much
higher than the cost of quickly implementing the CORRECTIVE ACTIONS
(CAs) of the LIKELY ROOT CAUSES ...*

Robust, Customer-Centric Design

*Creating something that DOESN'T FAIL is one thing.
Creating something that doesn't
FAIL TO AMAZE OUR CUSTOMERS and USERS is another.*

The previous chapter covered the **reliability mindset**, where we are motivated to PREVENT **failures** from ever happening because we understand how valuable this is. We know that **failures** are one form of many different types of 'problems' that we will encounter during the production process. The **reliability mindset** is based on a common understanding or language of **reliability engineering** concepts, used to execute the VITAL FEW **reliability engineering** methods that have been identified as part of the natural production flow of the organization.

FTA might be one of these methods. **Fault trees** can help us conduct **RCA** on likely **failure modes** before they have occurred to help us develop **corrective actions** to be embedded into our first design. But this perspective on **failure** can be rather restrictive. **Failures** (in this context) are seen as very 'technically defined' events that are linked to **requirements**.



But what if our **requirements** are not 'right?' Or 'complete?' Or we don't understand what our customers want? When organizations stop rigorously assessing and updating their **requirements** based on their customer and user, they very quickly stop being successful.

BUT... that is outside the scope of this tutorial...

As are **redundant systems** that aren't **parallel**, **Common Cause Failure (CCF)** and **dependence**, links to **Reliability Block Diagrams (RBDs)**, '**k**' out of '**n**' systems with different **components**, **component** and **system Mean Times to Failure (MTTF)** and **Mean Times Between Failure (MTBF)**, **rare event approximation**, how to model **bridging systems**, other unusual **logic gates** that should and should not be used in **Fault Tree Analysis**, and how we facilitate **RCA**. So please keep learning!

... and the final WORDS

Fault Trees are tools. It doesn't matter how good a tool is if it is being used in the wrong way, to solve the wrong problem, or by someone who doesn't know how to use that tool. And **fault trees** are incredibly flexible to help solve lots of different problems.

We first looked at how we can model **system reliability** with a **fault tree**, and then use that **fault tree** to model **system reliability**. Whether it is a **parallel, series, 'k' out of 'n' system** or something more complex, **fault trees** are great. And we can use this analysis to find things like **warranty period, warranty reliability**, the effect of variation in **component times to failure** and so on.

Fault trees are also really great at helping us with **Root Cause Analysis (RCA)**. We use **fault trees** in different ways. We are all about trying to create 'layers of explanation' and so on to work out what we need to do to stop **undesirable events** occurring. The **fault trees** use the same **logic gates** in each application, but they are used very differently.

And then finally, there are **fault trees** that help us with **robust, customer-centric design**. And that involves helping us work out what **features** and **functions** we need to include in our design. Again, **fault trees** are used in different ways.

So when you are faced with a '**reliability problem**,' work out if **fault trees** can help you, and if they can, how they can.