# What is a Fault Tree?

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

# WORKBOOK

Presented by

**Dr Chris Jackson**

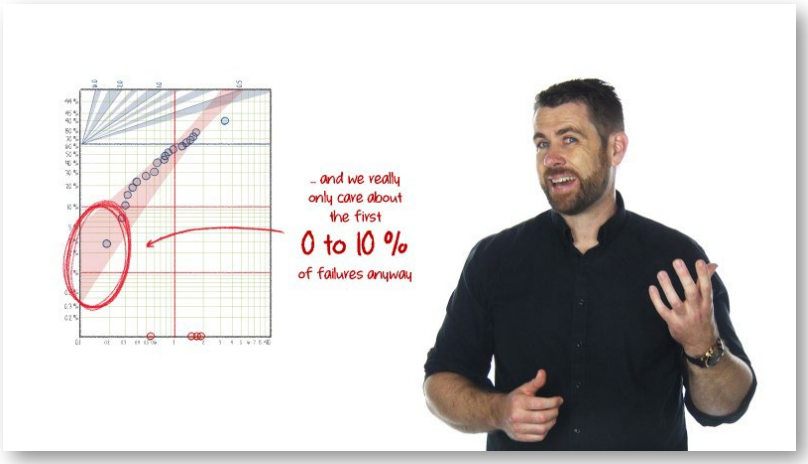
Reliability engineer

acuitas

reliability through insight

# 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 is the cofounder of IS4 online learning, 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 develop 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 us at contact@acuitas.com

## Contents

Who can benefit from using a fault tree? Lots of different people. People who need to solve a specific **reliability engineering** problem. People who want to improve their **reliability engineering** skills and expertise. People creating amazing new products, meaning they need to understand what their customers want. People who need to conduct **risk analysis**.

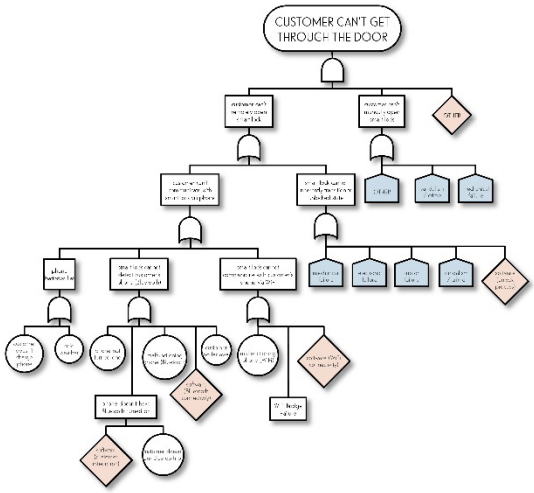
Then there are people who need to work out what the **warranty period** of their amazing new product should be. And people who need to work out why something on their factory floor is now on fire. People who want to know what design features and characteristics customers really want. People who want to improve manufacturing and production performance.

And lots of other type of people. Regardless of who you are, you will soon be able to use **fault trees** to help you make decisions like the ones the people above need to make.

**Fault trees** should only ever be used to help **INFORM DECISIONs**. If not … there is no point in conducting **Fault Tree Analysis (FTA)**. And before you conduct **FTA**, you need to *really* know what **decision** you are trying to make.

## So what does a FAULT TREE look like?

An example **fault tree** is illustrated on the right (it is just a high-level example … don't worry if you can't read the writing!) You can see that it is made up of different shapes and lines, all starting from a single shape at the top which usually represents something 'undesirable' (called the **top event**). So it doesn't much look like a tree.





In practice, **fault trees** look a lot more like the **root system of a tree**. They always emanate downwards from the **top event** which must be related to the **decision** you are trying to inform. So you need to know what options your **decision** has, how we measure the outcomes of the **decision,** what makes an outcome 'good' and so on. The **top event** is the scenario we are interested in studying more … and we will talk about what the **top event** is later.

It is vitally important to really understand your **decision**. If you don't, you will waste a lot of time and effort in developing a **fault tree** that has no impact, wastes everyone's time, and reduces anyone's motivation to use **fault trees** in the future.

To better know what to look for in your **decision**, we need to talk about the three different perspectives or reasons you might want to conduct **FTA**. Most reference sources (textbooks, standards, professors and so on) tend to focus on only one of these perspectives (which is usually their favourite). Not understanding all three perspectives means you lose all manners of amazing flexibility within **FTA** … simply because you don't know what they are capable of.

## Perspective #1 – analyzing system RELIABILITY

**Systems** are usually made up of lots of **components**. Therefore, **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**'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.*

**FTA** as part of **RCA** looks very different to **FTA** as part of **system reliability analysis** (reinforcing how important it is to understand your **decision**). We of course don't just conduct **RCA** to increase our knowledge. We conduct **RCA** to *do something*. **FTA** can help us identify lots of wonderful ideas about how we can prevent that **undesirable 'top' event** from happening in the future (through redesign, preventive maintenance, operational changes, new components and plenty of other **corrective actions**).

## Perspective #3 – robust, customer-centric DESIGN

The techniques employed for **RCA** can also be used to identify the root causes of *potential* failures. This allows us to *prevent* **failures** from ever occurring (instead of waiting for a **failure** to happen before you design it out of your system) This approach is like doing **RCA** in advance and identify the likely scenarios in which **failure** will occur.

If we are smart enough to know how to build a system or process that works, we are smart enough to know how it will likely **fail**. So, we make our **first design a reliable design**. But it isn't just about **failure**.

We can use this pre-emptive approach for any **undesirable event**. So we can apply this not only to the traditional definition of **failure** (based on the physical deterioration of something), but scenarios like not meeting customer expectations with design features and functions. 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** (which we will examine later) 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. This might seem trivial but is one of the most important things you can do to ensure a 'good' system. For example, if you are designing a military vehicle … is it worse if the top speed reduces by 15 km per hour or the payload reduces by 500 kg? Working this out in advance gives much needed design guidance to the young engineers designing the next generation of amazing machines.

And while there are so many useful things **fault trees** can help us out with, it is all for nothing if **FTA** is not done well. **FTA** often involves a group of people. So we will also cover how to conduct **FTA**, facilitate the group, and ensure everyone has a positive experience.

## Focus on the DECISION first and foremost

Are you trying to **measure reliability**, … or perhaps prevent a **failure from happening AGAIN**, … or maybe prevent a **failure from EVER OCCURRING**, … or maybe making sure you **don't fail to meet CUSTOMER EXPECTATIONS**? Make sure you spend enough time on working out the decision you are trying to make before conducting **FTA**.

## using FAULT TREES to model SYSTEM RELIABILTY

**Failure** is a random process. **Reliability metrics** can be parameters or characteristics of this random process.

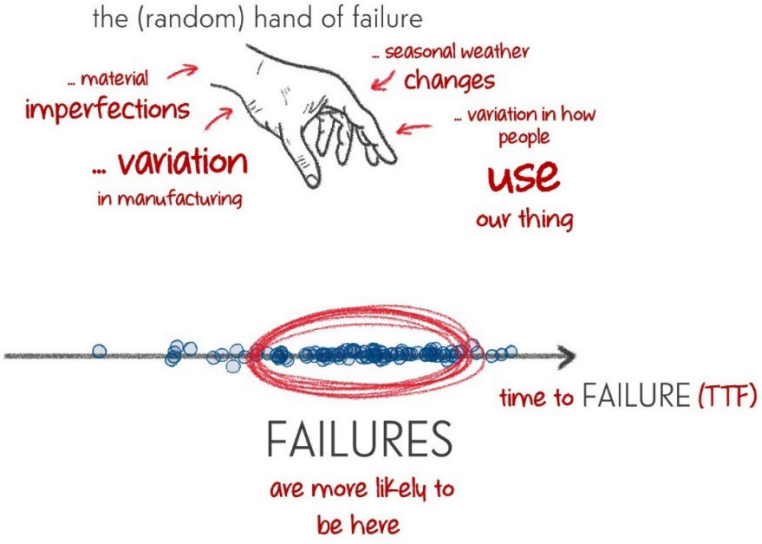
*Metrics are standards for measuring or evaluating something*

**Reliability metrics** can be things like **warranty reliability**, **mission reliability**, **operational availability,** the time taken to repair something during a mission, and any number of other quantities. There are also **metrics** like the number and weight of tools, which aren't random process parameters but are still characteristics of **RAM performance**.

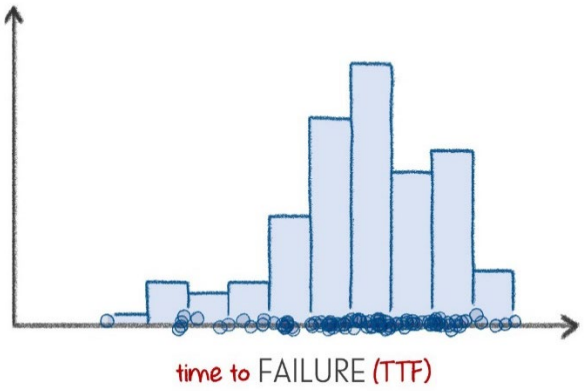
## what is RELIABILITY?

We can't 'schedule' **failure**. If (for example) we have 20 identical engines, they won't fail at the same time.

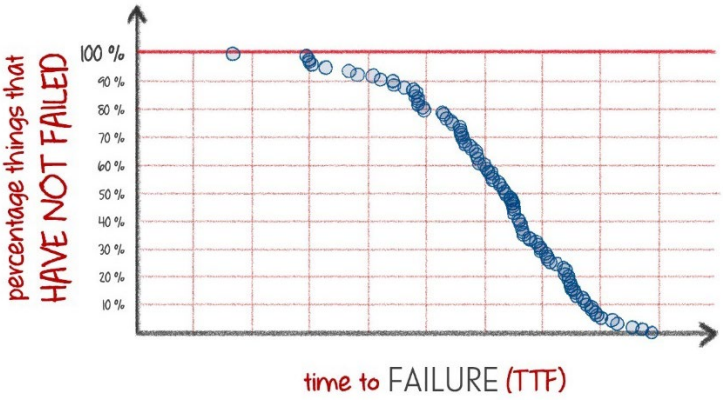
In practice, there are many factors that influence the time it takes for something to **fail**. There are variations in scenario, build state, and lots of other things that might influence how long it takes for our engine will work for. Let's represent all these variable factors with a thing we call '**the (random) hand of failure**.' But just because something is **random**, doesn't mean it isn't predictable.



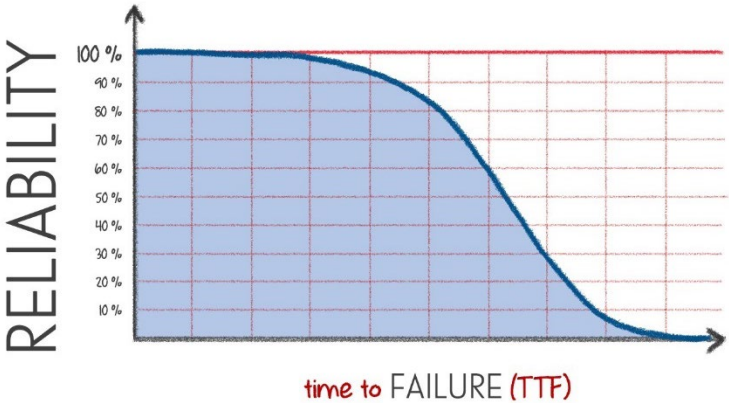
We can visually summarize the nature of this random process by observing **random data** and drawing a **histogram** to represent regions where **failure** is more likely to occur.



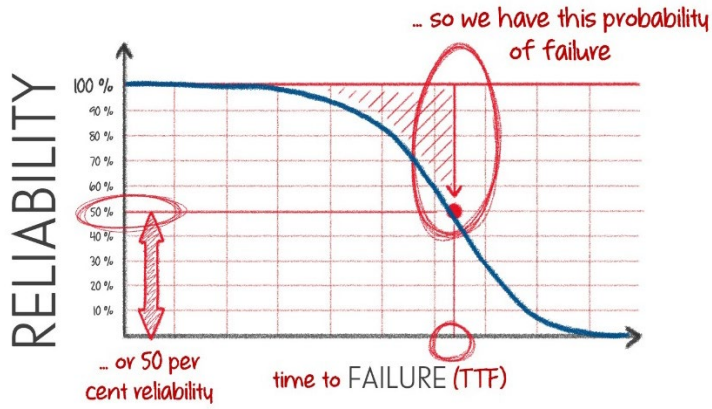

Another way of **visually summarizing** this random process is using the same **random data** to (approximately) plot the percentage of things (engines) that have not failed over time.



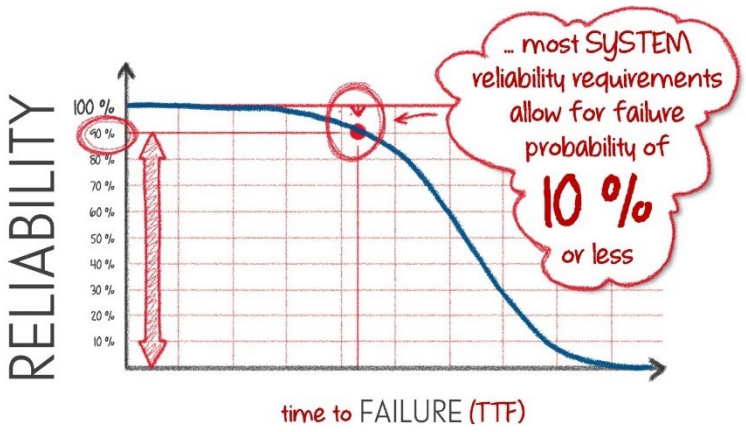

If we were to (hypothetically … but impossibly) observe a HUGE number of engines failing, we could eventually create a perfectly smooth curve that represents the percentage of things that have not failed. This is the **reliability curve**.



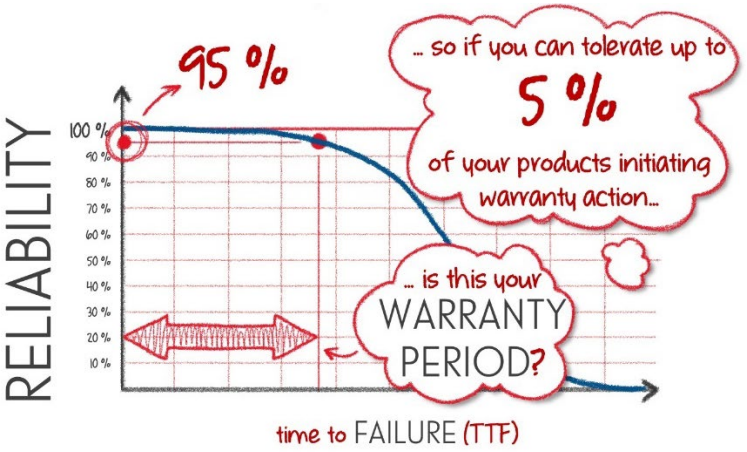
If we know the **reliability curve,** we can determine the **reliability** of a system at any particular point in time or usage.




But ... we really aren't (often) interested in understanding when 50 % of our products, systems, component or items have failed. By then it is too late. Having half of our systems or plants failed or offline usually means we can't achieve the mission or make profit. Instead ...

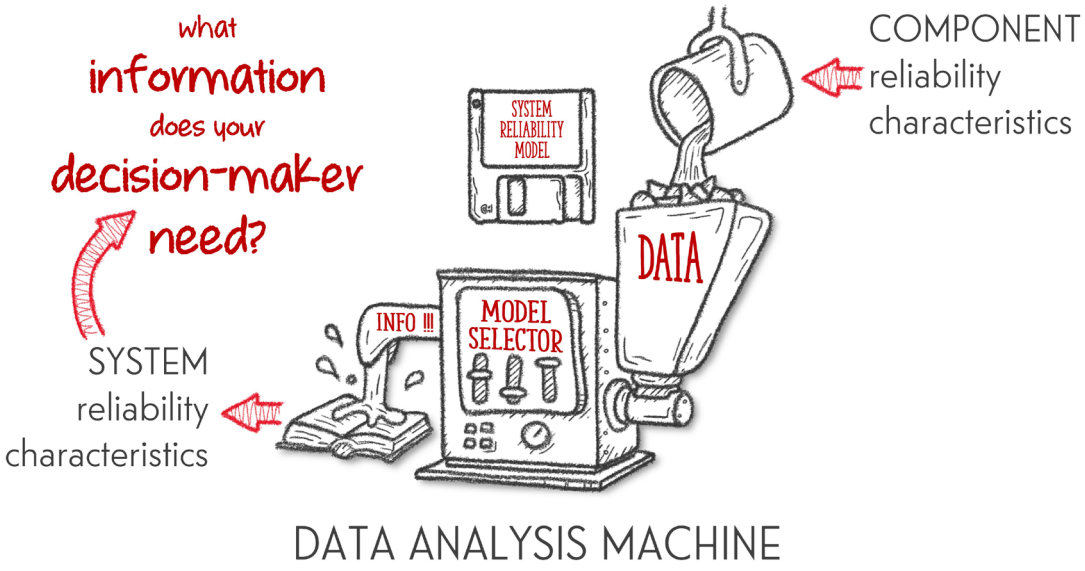



For example, say you were able to work out that for your amazing new product, you can tolerate up to 5 % of your entire sales failing during the **warranty period**. You don't want any more than 5 % to fail otherwise you will start losing too much money through **warranty costs**. You ALSO don't want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



## what does a SYSTEM RELIABILITY MODEL do?

A **system reliability model** allows us to conduct **data analysis** and help us understand **system reliability characteristics** using **component reliability characteristics**. The output of this **data analysis** is information that SHOULD help a decision maker make a decision.

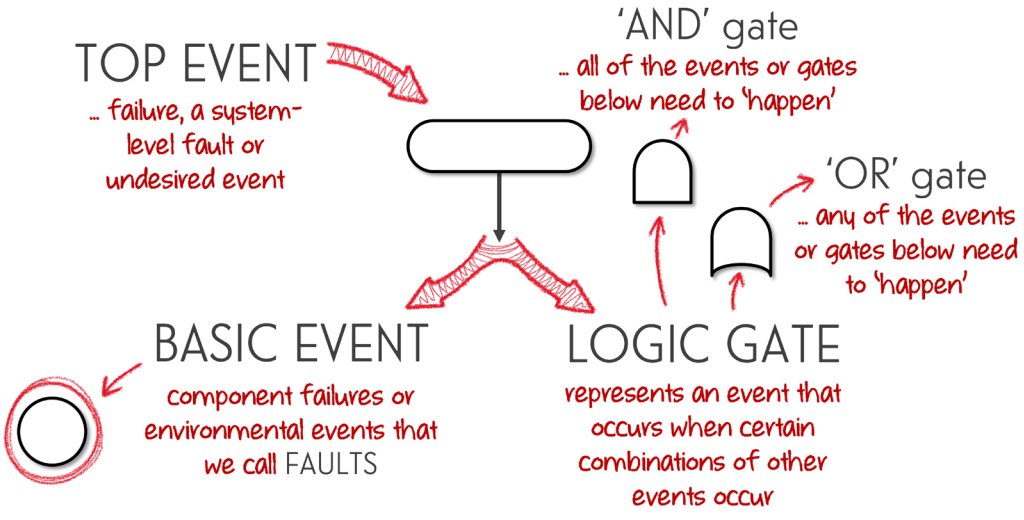



This process can be referred to as **statistical analysis** because all we can ever hope to do is describe some of the characteristics of failure … which is at its heart a random process.

A **fault tree** is one of many ways of representing a **system reliability model**. But to do that, we first need to understand how a **fault tree** works.

## so …. how does a FAULT TREE work?

A **fault tree** is much easier to look at 'one bit at a time.' And to do that, we need to understand the shapes we usually use to make a **fault tree**. The most common shapes are illustrated below …



Most **fault trees** are built by starting with the **top event**. We then start drawing a line down. The line must connect with either a single '**basic event**' or a '**logic gate**.'

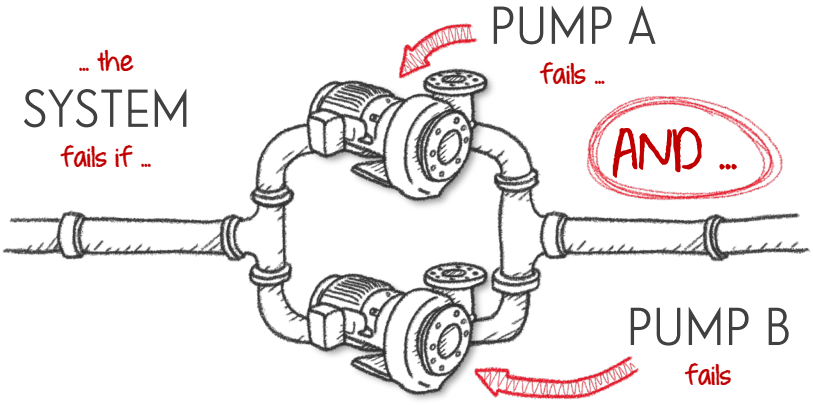
A **basic event** is often referred to as a **fault**, which is where we get the name '**fault tree**.' A **fault** is essentially any event that has the potential to result in **failure**. In **fault trees** used to **model system reliability**, **basic events** are usually **component failures**. In **fault trees** used to conduct **RCA** or assist **robust, customer-centric design**, **basic events** can be design flaws, environmental events, human errors, or any other scenario that might contribute to **failure**. The reasons behind this distinction are discussed later in this course.

We know most systems have more than one component. So **fault trees** usually have more than one **basic event**. Therefore we also need **logic gates**. **Logic gates** represent an event which is the combinations of other events. These combinations can include **basic events** and other **logic gates**.

And '**AND gate**' represents ALL of the events or gates 'beneath' it occurring. The events 'beneath' each gate are called **input events**. An '**OR gate**' represents AT LEAST ONE of these **input events** occurring. The '**AND gate**' and '**OR gate**' are the two most common **logic gates** used in **FTA**. Other **logic gates** used to create **fault trees** will be examined later in this course.

## Let's start with an 'AND gate'

Consider the simple, two pump system illustrated on the right. The system fails if pump A fails **AND** pump B fails. This is what we call a **parallel system** (mainly because this is what it sort of looks like).



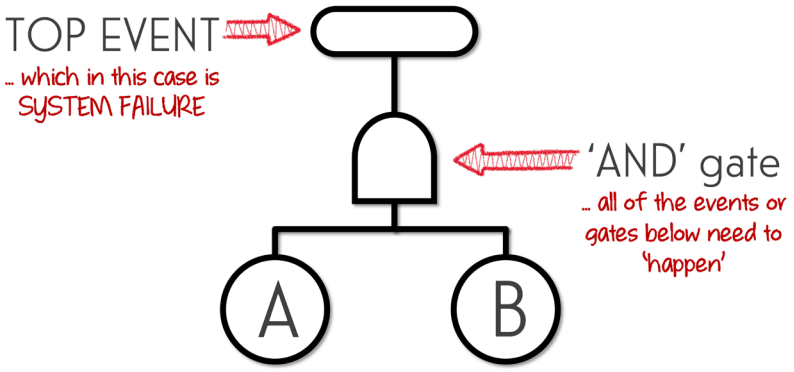

*The most important word for this system is 'AND.'*

To create the **fault tree** for this **two-pump parallel system**, we need to use an '**AND gate**.' The first thing we need to do is define the **basic events**.

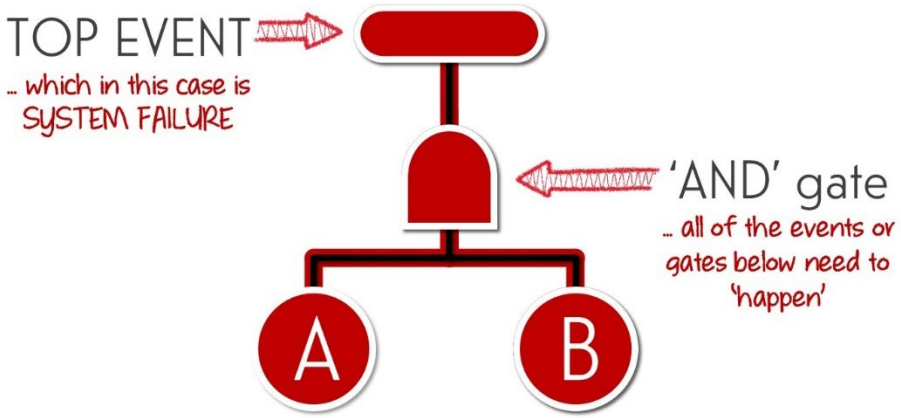
*Basic event 'A' is the event where pump A fails.*

*Basic event 'B' is the event where pump B fails.*

This allows us to use an '**AND gate**' to create the following **fault tree** that models the reliability of our **two-pump parallel system**. The '**AND gate**' has a round top and **smooth bottom**.

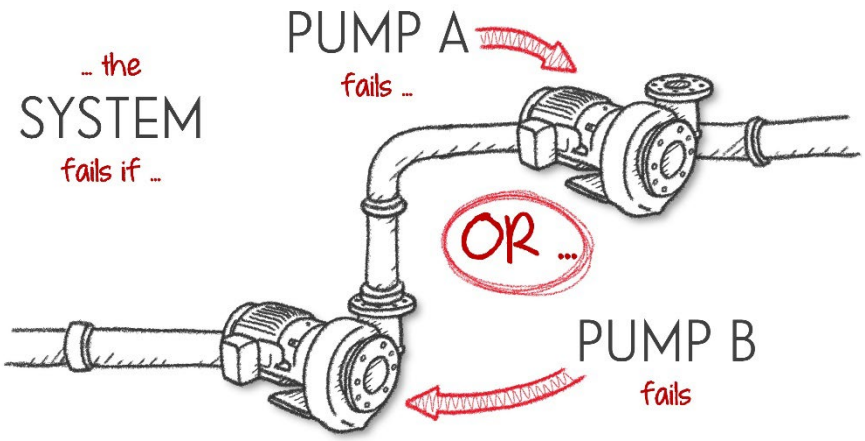



So for the **two component parallel system** that the fault tree above represents, **failure** looks like this (where red represents failure) ...




## 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 what we call a **series system** (mainly because our pumps are arranged one after the other ... in **series**).

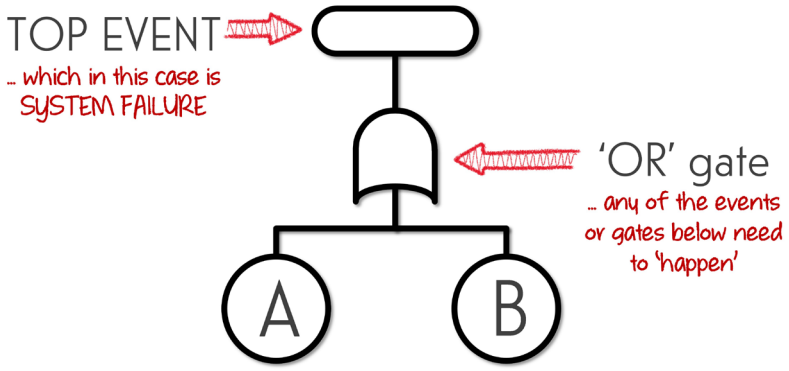
*The most important word for this system is 'OR.'*

Remember what our two **basic events** are?

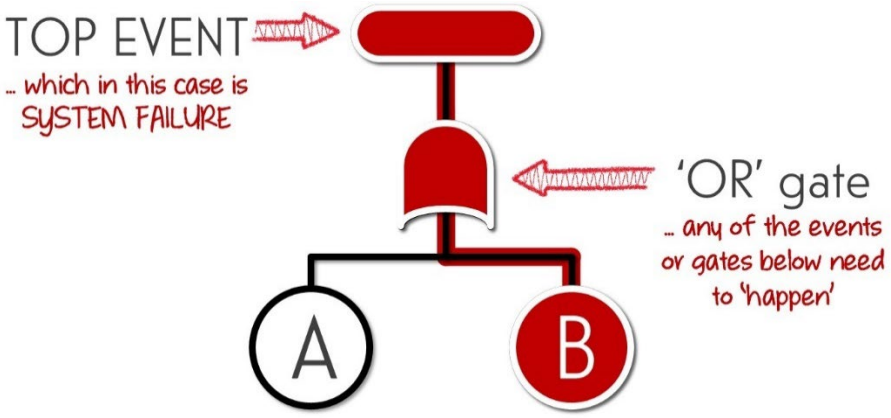
*Basic event 'A' is the event where pump A fails.*

*Basic event 'B' is the event where pump B fails.*

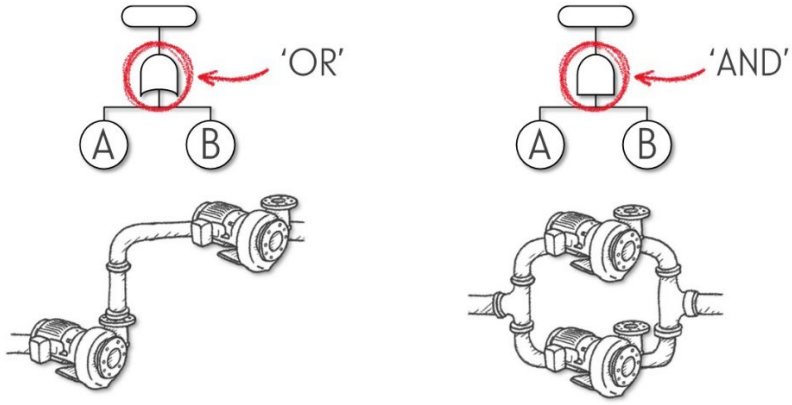
This allows us to use an '**OR gate**' to create the following **fault tree** that models the reliability of our **two-pump series system**. The '**OR gate**' has a round top and **round bottom**.




So for the **two component series system** that the fault tree above represents, one **failure** scenario looks like this (where red represents failure) ...

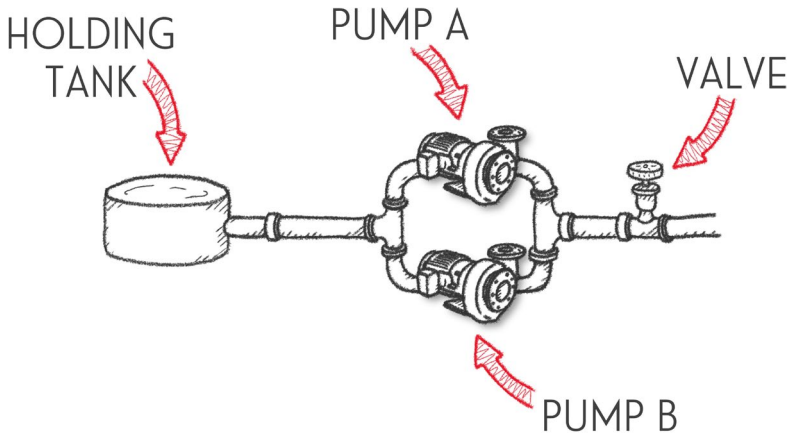


The examples in the previous chapter involved simple, two-component systems that were either arranged in **series** (system fails when one component fails) or **parallel** (system fails when all components fail). The corresponding system reliability fault tree models are based on '**OR gates**' for **series systems** and '**AND gates**' for **parallel systems**.




systems usually have WAY MORE than two components

Systems are usually (much) more complicated than this. Consider our **two-pump parallel system**, which is now a subsystem in this (slightly) more complex system.




So how do we create a **fault tree** that describes this system?

*We start from the top.*

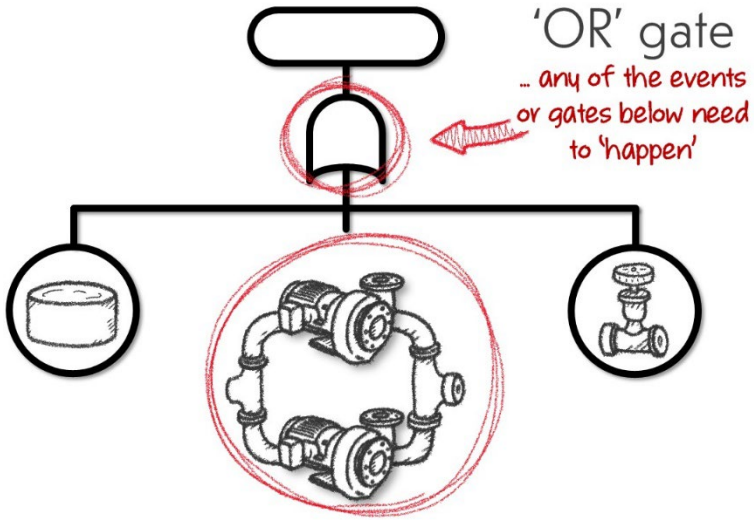
We can see that the holding tank, the **two-pump parallel system**, and the valve are arranged in **series**. This means that for our system to fail, any one of the following events need to occur:

*holding tank fails*

*OR … the two-pump parallel subsystem fails*

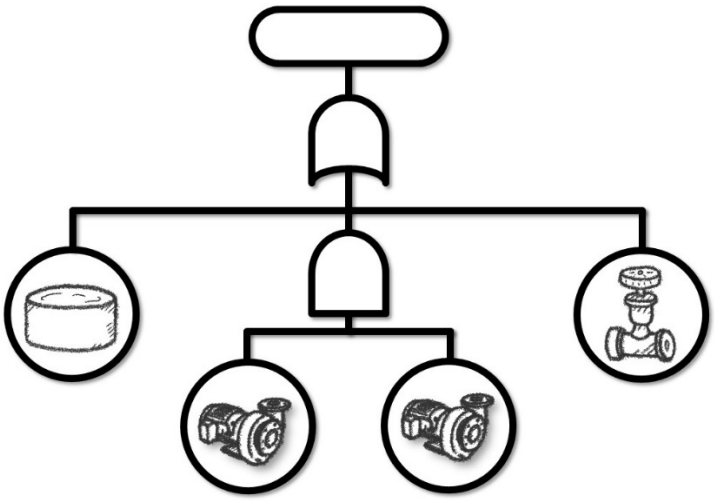
*OR … the valve fails*

This allows us to create the following **fault tree** based on an '**OR gate**.'




We can see that this '**OR gate'** has three **input events**. **Any logic gate can have any number of input events**.

But our **fault tree** clearly isn't finished yet. The next step is to replace the circled **two-pump parallel system** with the '**AND gate**' model we developed in the previous Chapter.



We now want to replace those pictures of our components with letters that can help us better define what each basic event is. So …

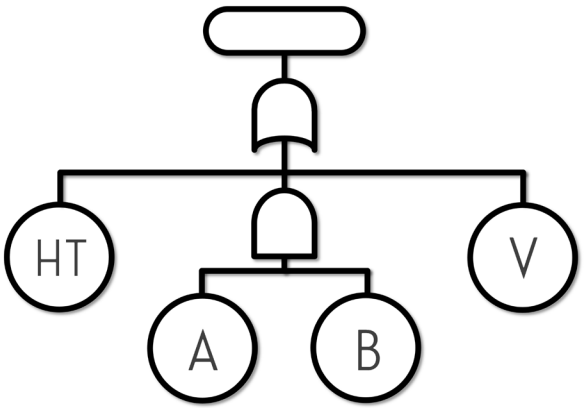
*Basic event 'A' is the event where pump A fails.*

*Basic event 'B' is the event where pump B fails.*

*Basic event 'HT' is the event where the holding tank fails.*

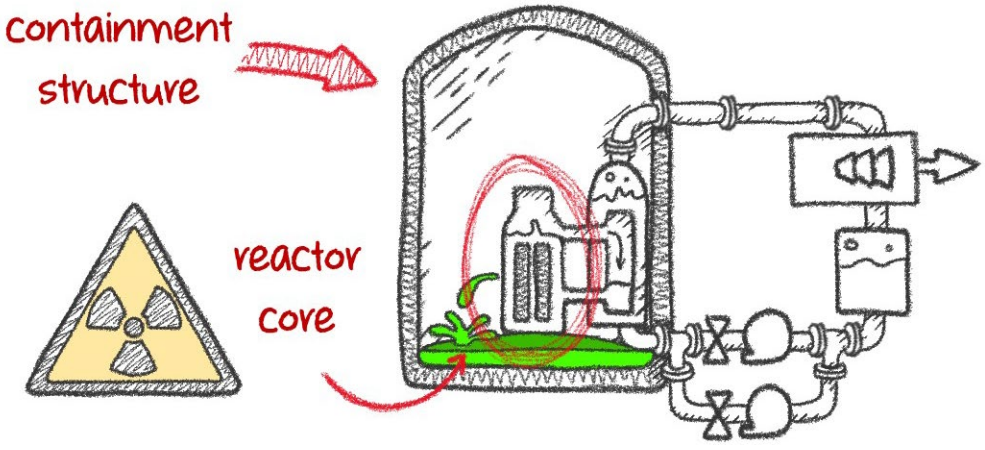
*Basic event 'V' is the event where the valve fails.*

Our final **fault tree** looks like this:




## let's look at a REALLY SIMPLE (still complex) nuclear power plant

One of the more complex systems is a nuclear power plant. In fact, the many hundreds of subsystems that makes up a nuclear power plant are themselves complex systems. A very simplified illustration of a nuclear power plant is shown below.



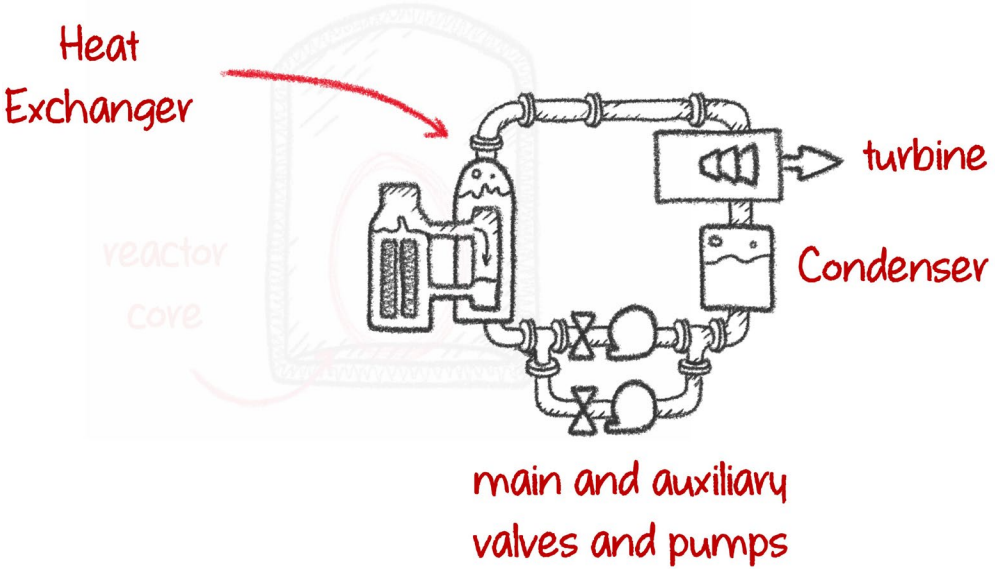

The most 'important' part of a nuclear power plant is the reactor core. This is where the uranium fuel undergoes nuclear fission to generate heat. This heat is imparted into water that circulates the fuel … along with many (very dangerous) radioactive by-products. For this reason, reactor cores are always built within a containment structure that is designed to capture any leaks of this coolant water. This will become the subject of many examples and exercises in this course.

EXERCISE

To generate electricity, the thermal energy that is passed into the coolant water that circulates around the uranium fuel needs to somehow 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.

**BEFORE YOU LOOK AT THE NEXT PAGE ...**

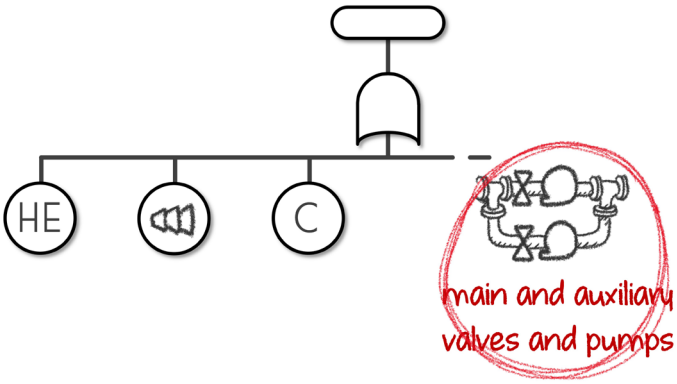
... draw a fault tree of the secondary cooling loop.

**IF YOU FINISH OR GET STUCK**, the following pages will take you through the necessary steps to draw our fault tree.

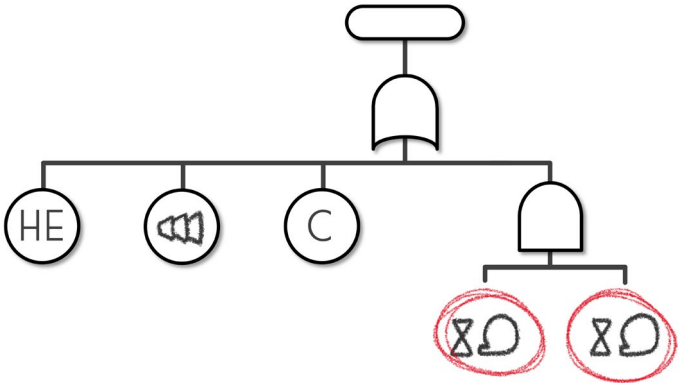
EXERCISE

EXERCISE

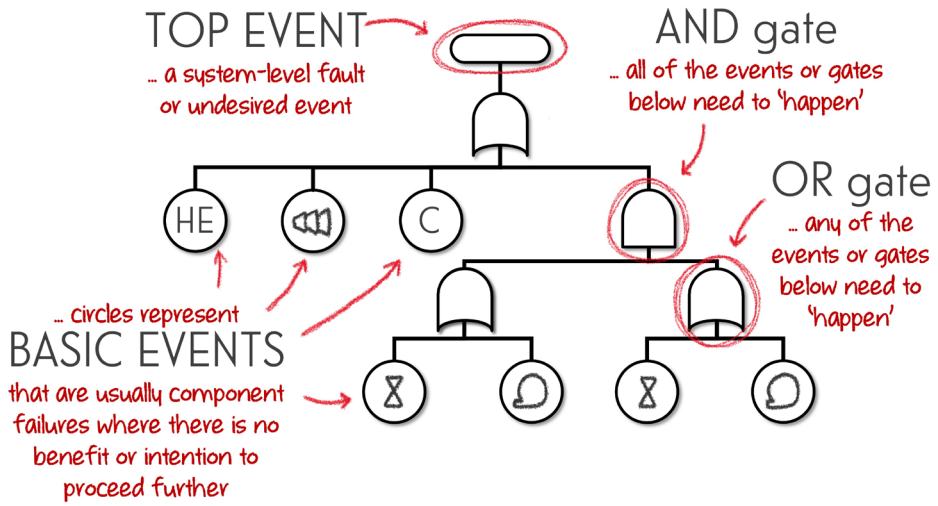
The first thing you should try to do, is arrange the secondary cooling loop into a high-level series system based on an 'OR gate.' For clarity, we are going to use component symbols and not letters as basic event identifiers. This is only to help us remember what each basic event represents IN THIS EXERCISE ... in practice we would use letters or other identifiers.




We now need 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.'

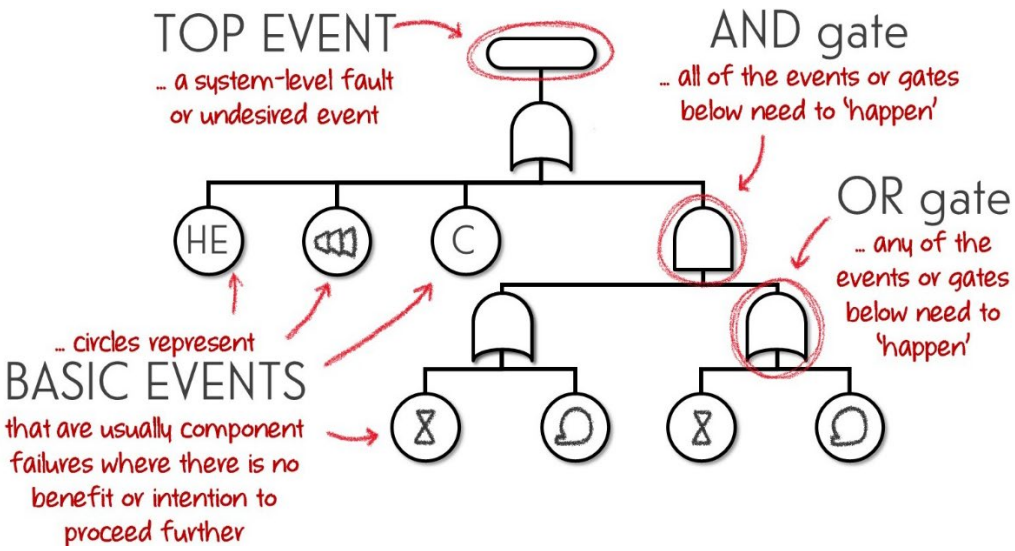



The last step is to look at each valve-pump pair. They are arranged in series. So we replace each valve-pump pair above with a series valve-pump subsystem that is based on an 'OR gate.'

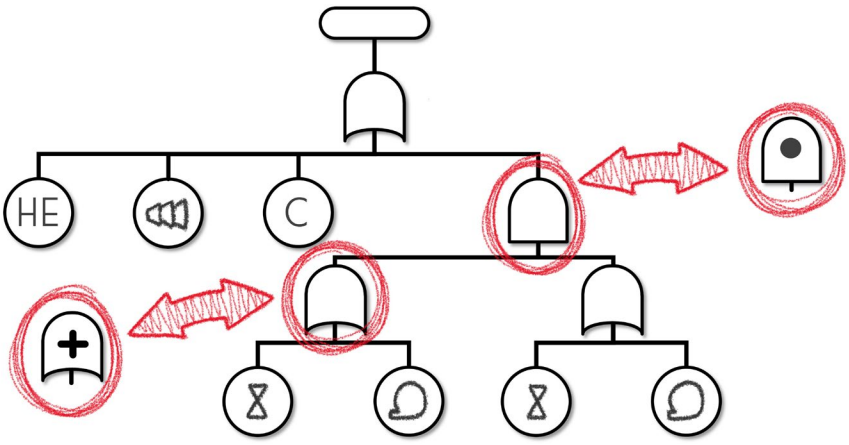


## More fault tree symbols

So far, we have looked at **fault trees** with the following four shapes that represent **events** and **logic gates**.

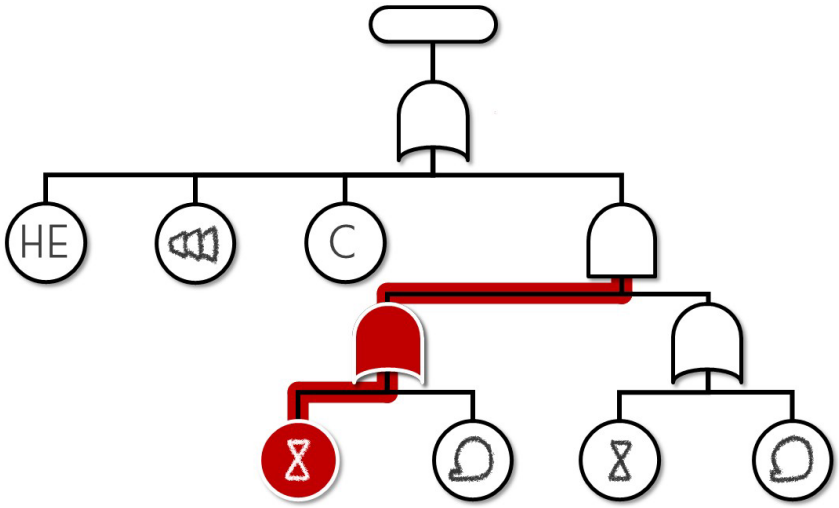



But many **fault trees** have different shapes and features that you need to be aware of. In some **fault trees**, '**AND gates**' contain a dot or circle inside the larger shape. '**OR gates**' sometimes contain a '+' or plus sign. This helps visually differentiate between '**AND**' and '**OR gates**' as some people can't easily differentiate between smooth versus flat shape bottoms.



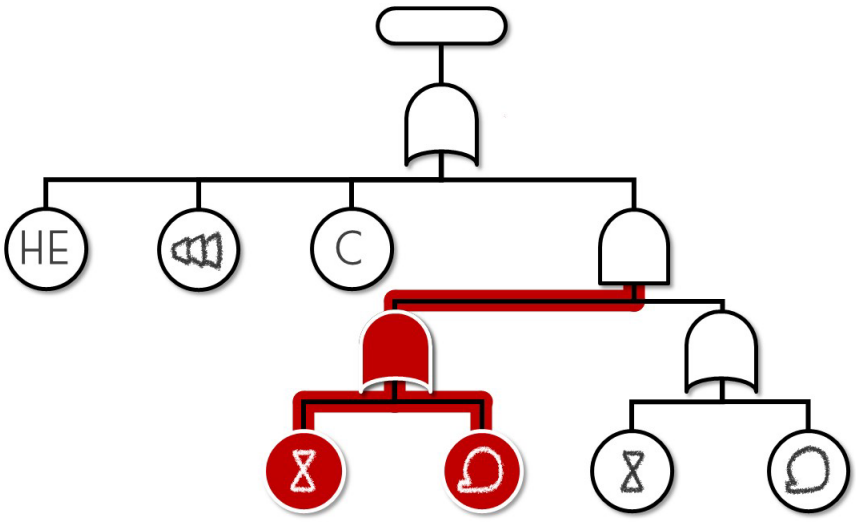
## ... let's look at our COMPLEX system FAULT TREE

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

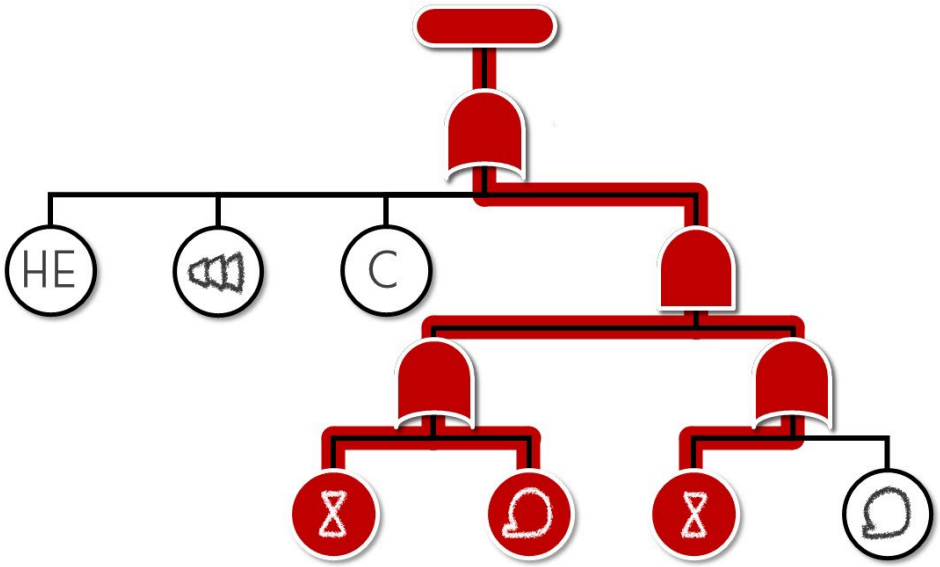



We can see that the system fault passes through the first 'OR gate' because it only requires one (any) input event to occur for an output to occur. But our system fault stops when it gets to the next 'AND gate' because it requires all input events to occur for an output to occur. So when our first auxiliary valve fails, the system still works.

Even if the first pump fails in addition to the first valve, the system will not fail.



But now, if the second auxiliary valve fails (in addition to the first auxiliary valve and the first auxiliary pump), then our secondary cooling loop will. This is because the 'AND gate' that was 'stopping' the system fault from propagating up through the fault tree now has all input events occur. So it now creates an output event that allows the system fault to move up to the next 'OR gate' and pass through it … stopping at the top event.




… but there is a LOT more …

In the next chapter we look at more features of fault trees, as there are many systems that have many more than six components. And there are times when a system includes a subsystem that already has a fault tree completed. So how do we incorporate other fault trees into a 'master fault tree?' This will be covered in the next chapter where we explore even more things we can include into a fault tree to help it describe the way our system fails even better.

# FAULT TREES and ROOT CAUSE ANALYSIS (RCA)

So far we looked at **fault trees** from the perspective of analyzing system reliability. But **fault trees** are also very useful for **Root Cause Analysis (RCA.)** Recall 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 ...

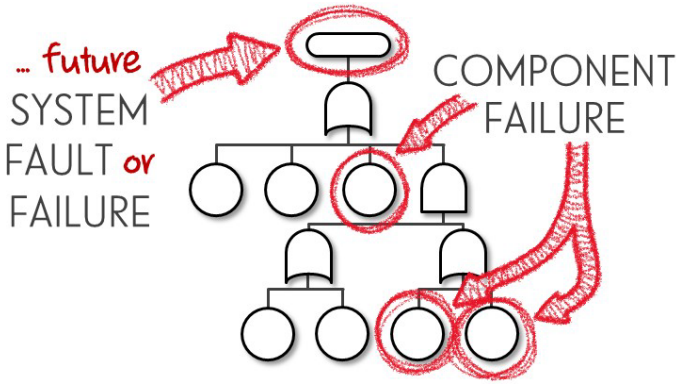
*WHY THINGS FAIL?*

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 ...

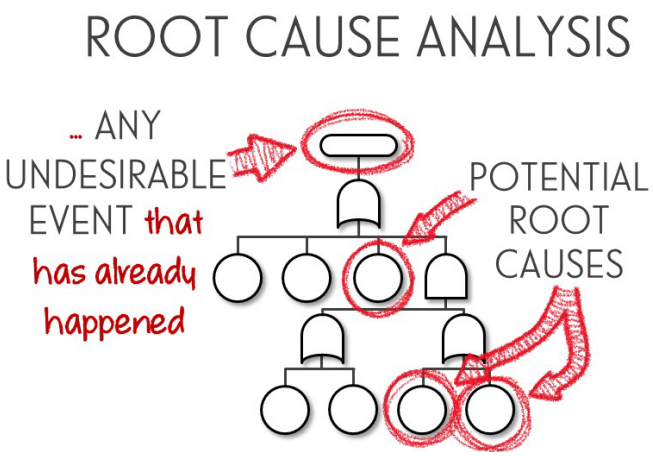
what can we do to STOP *THINGS FAILing?*

## ... what part of FAULT TREE analysis is different for RCA?

**System reliability model fault trees** have the following basic structure where **top events** represent system **faults** or **failures** and **basic events** represent **component failures**.



A **fault tree** constructed as part of **RCA** is very different. The **top event** can be any **undesirable event** (that has usually already happened). The **basic events** are **potential root causes** identified as part of the analysis. We can then investigate the likely **potential root causes** in our **fault tree** further to hopefully identify which **root causes** apply to this **undesirable event**.




This might seem quite similar to a **system reliability fault tree** (after all, **system failure** is an **undesirable event**). But an **undesirable event** can be absolutely anything, like ...

*... company failing to make a profit in the previous quarter ...*

*... an aircraft crash ...*

*... a large number of employees resigning over the previous financial year ...*

And **root causes** are not **component failures** – even if we are conducting **RCA** on a **system failure**.

## ... so what is a ROOT CAUSE?

Many organizations and 'experts' really don't understand what a **root cause** is. Many textbooks and standards say thing 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 lead 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

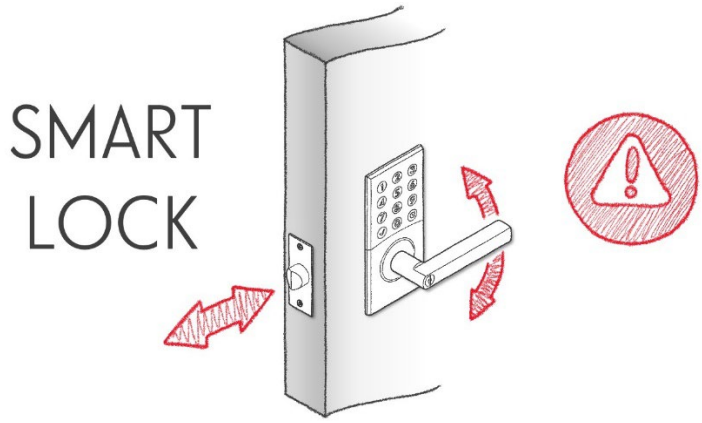
*... a ROOT CAUSE* is the THING we can change to make sure 'that' failure hardly happens again ...

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**.

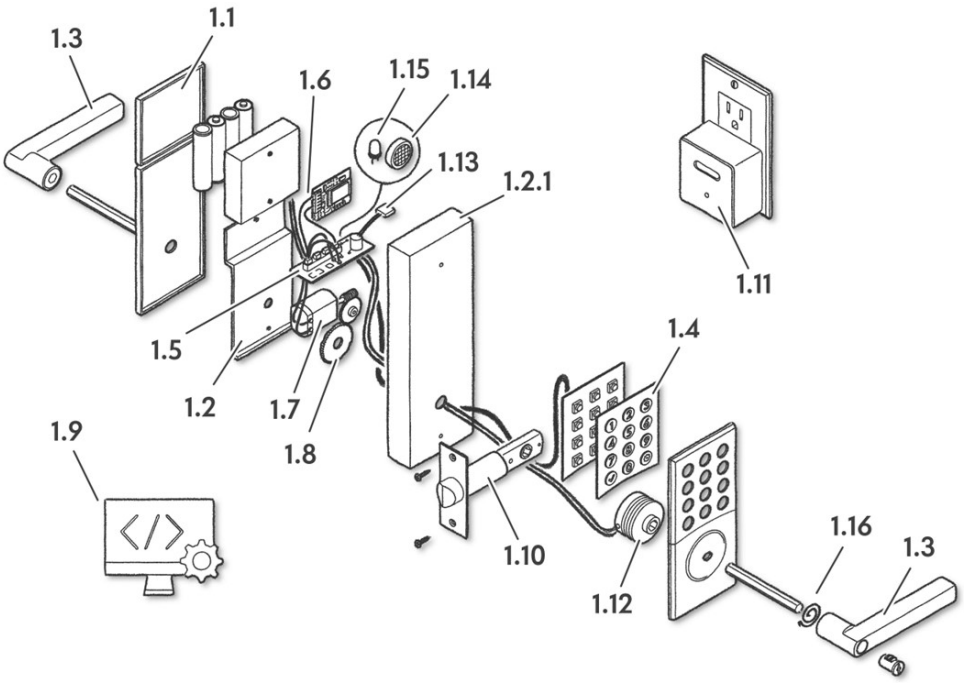
## … let's do some RCA on a smart lock …

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.

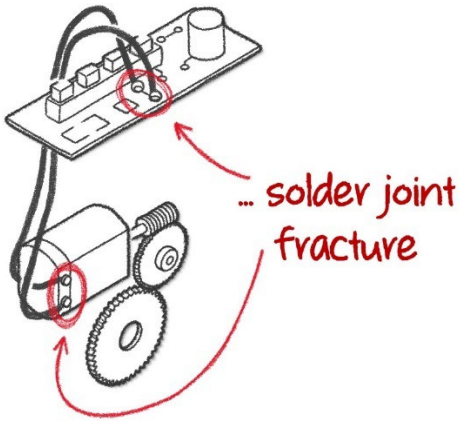
But the production prototype below stopped working during a user demonstration. Now we need to work out why.




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

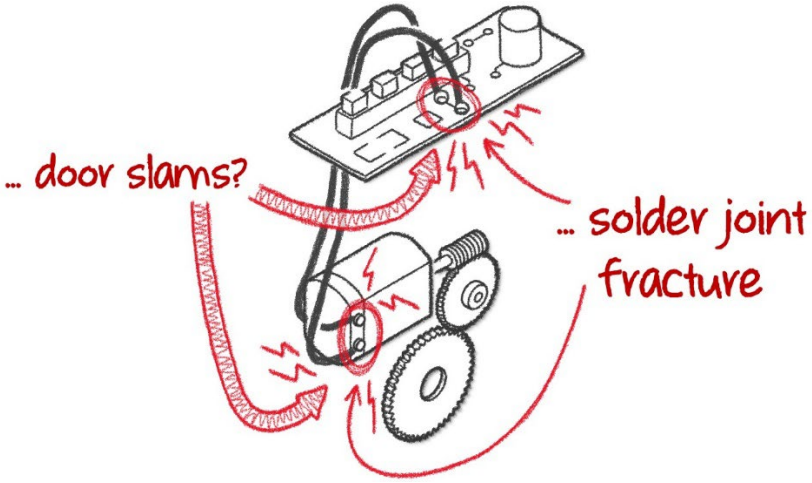



*Is this the ROOT CAUSE of failure?*

If fractured solder was the **root cause** of **failure**, then we should simply be able to correct the design flaw that lead to this failure be 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 lead 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.

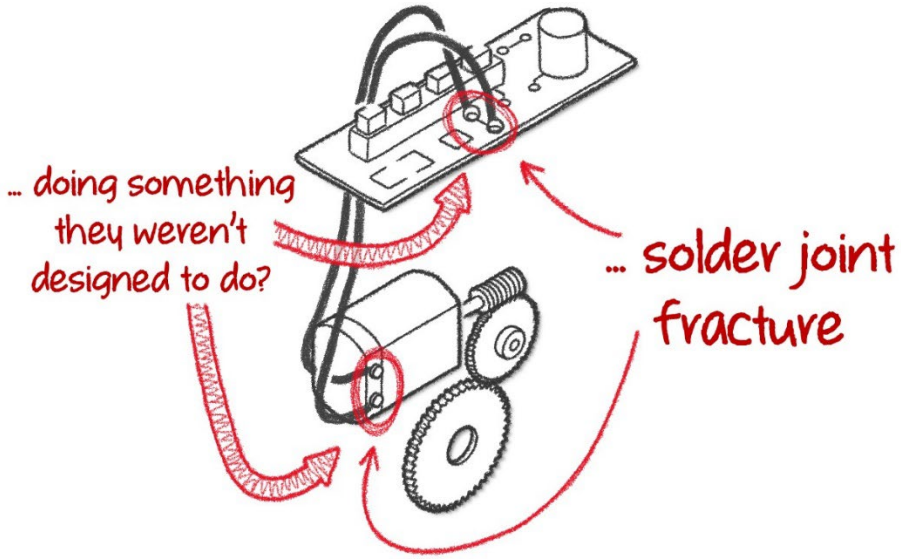



*Are DOOR SLAMS the ROOT CAUSE of failure?*

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 the 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?*

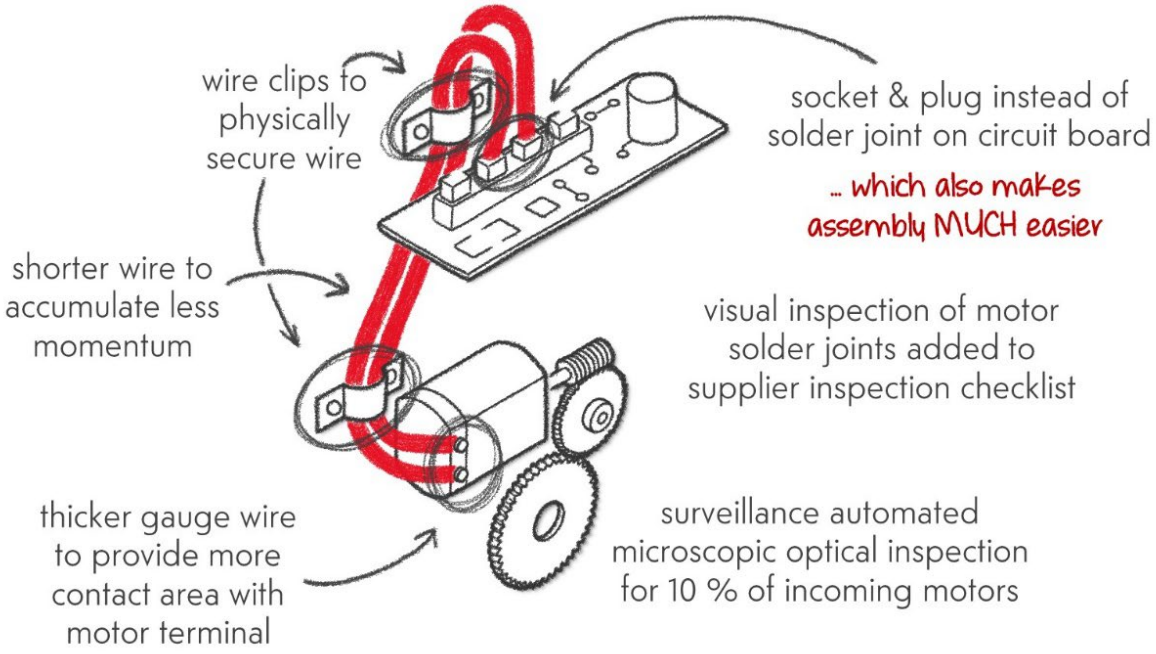
If the answer is yes, then we can perhaps reword the root cause as

*… 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 **CAs**. As long as they start 'talking' about things we are able to address, then each organization will likely come up with fantastic **CAs** that will quickly, efficiently and cheaply eliminate this **root cause** of failure.

## … what do CORRECTIVE ACTIONS (CAs) look like?

While the **root cause** we identified on the previous page appears to be all about design issues, there is no need to limit **CAs** to just design. Six (6) CAs are identified below … two (2) of which deal with manufacture and inspection.




If we identify **CAs** early enough in the production process, then we can eliminate **root causes** for virtually NO COST. The six **CAs** above will involve some basic cost increases (such as purchasing thicker gauge wire), but when compared to the overall cost of the smart lock are relatively insignificant. This is what good **RCA** results in.

*Some of you might be asking … but how do we know we got the RIGHT ROOT CAUSE?*

*… what if the CAs above don't address the ROOT CAUSE behind the failures we observed?*

This (in a way) misses the point. If all the **CAs** above are virtually free to implement (if we come up with them early), then all we need to do is find the list of *likely* **root causes** through. Let's say we come up with ten (10) possible **root causes** for the solder joint failures in our smart lock, and then generate **CAs** for each of them. Implementing all these CAs is still going to be virtually NO COST. We don't want to waste time and money trying to find the RIGHT **root cause** … it is much cheaper to find **CAs** for the LIKELY **root causes** … which we call the VITAL FEW.

It all starts with finding the true **root cause**. Something you can influence. The 'THING' you can change to make sure 'that' **failure** hardly happens again. If it is not something you can influence (i.e. something you have done or about to do) then it is not a root cause.

## … the TREE OF FAILURE

We need to bake **reliability into our first design**. That means we need to STOP and use our team's collective understanding of how our system *can* work to predict how it *might not* work. And when we do, we can come up with lots of really easy and cheap design features and activities to make **reliability** happen.

A big part of **reliability engineering** is STRUCTURED BRAINSTORMING. In fact, some of the most effective **reliability engineering** activities revolve around how we facilitate group discussions that ensure we understand our product, system or service and how it will LIKELY fail.

And to brainstorm, we need a language. So let's start by visualizing **failure** … as an apple.

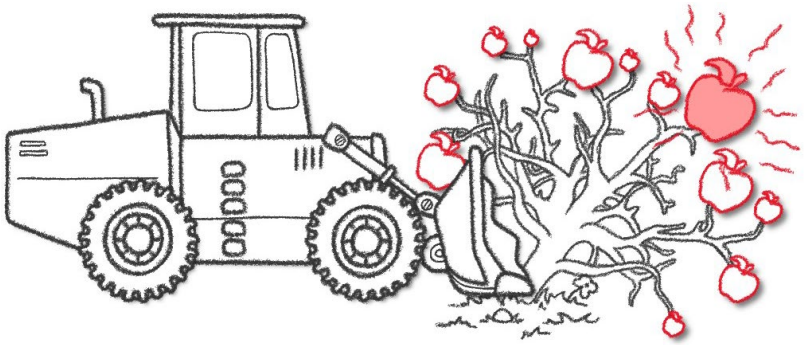
FAILURE event

We know that fruit (like apples) grow on trees. But not all **failures** (or apples) are the same. The same tree can produce fruit of different sizes, shapes and quality. In this case, we represent the **failure process** of a single system with a 'tree of failure' that produces 'big apples' that represent 'critical' or 'severe' **failure**, and 'small apples' that represent 'minor **failure**.'


tree of FAILURE

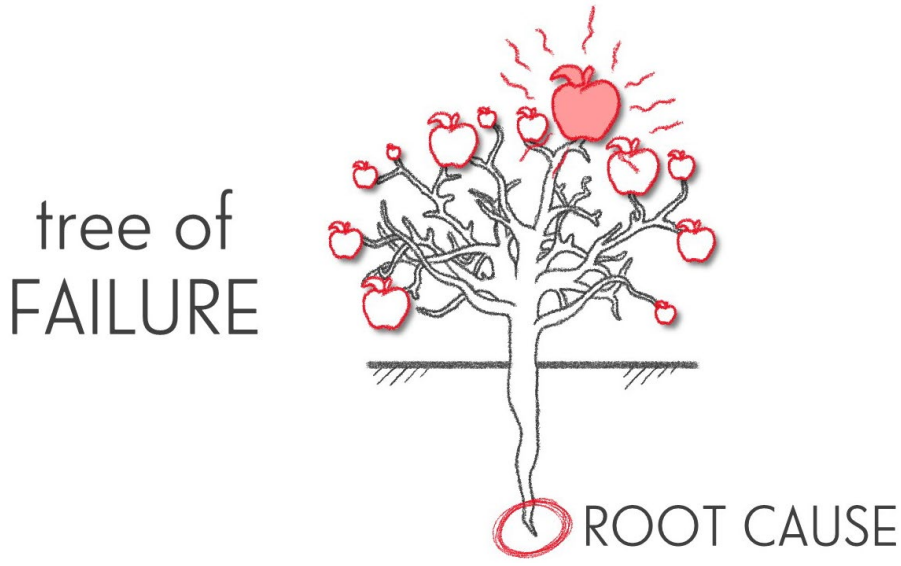
*So how do we deal with these (undesirable) apples?*

One thing we can do is to 'get rid of them' after they appear. We might even think we get rid of the entire tree when we **repair** our system or conduct **Corrective Maintenance (CM)**. Perhaps we come up with a 'workaround' for a particular asset, where we change our processes or procedures to accommodate these **failures**.



But there is a problem with this. **Repairs** and **CM** cost money and always mean our system stops working at inopportune times. So not only do we spend money on **maintenance**, we lose money because our system isn't working when our schedule demands it to be. And our workarounds almost always take additional time and money from our budget.

This is because even if we remove the apples – and the tree that produces these apples – the tree (and apples) will grow back. We might have removed the tree … but we have not removed the **root** (**cause**) of **failure**. The only way to ensure that the apple (**failures**) never come back is to eliminate the tree at the **root** (**cause**).



But there is a problem with the concept '**root cause**.' **Failures** often have more than one **root cause** OR can be best mitigated by addressing another 'contributing' cause that might not be the 'true' **root cause** ... if one exists.

For example, let's say that the 'true' **root cause** of **failure** was a design problem where an electrical engineer forgot to conduct vibration analysis on the circuit board of the smart lock. This means that one capacitor that is located on the corner of the circuit board AND a long way from mounts vibrates excessively when the door is slammed. This means that the capacitor's solder joints are prone to fracture.

But when we first uncover this problem in early prototype testing, there is no real benefit in apportioning 'blame' to the electrical engineer (or even trying to work out if there is blame to apportion). The 'simplest' fix or **CA** might be to not redesign the circuit board, but to redesign the casing to allow the corner of the circuit board to be dampened and secured. This **CA** is now going to be implemented by the design team lead in charge of the casing ... not the electrical engineer in charge of the circuit board. Or perhaps *everyone* is going to implement **CAs** to ensure we never see the capacitor come free again.

*Remember, if a CORRECTIVE ACTION (CA) is implemented early,*
*it is often VIRTUALLY FREE*

You can see this in the six **CAs** on page 28 that were identified for the solder joints on the cable that connects the electric motor which involve designers, engineers, manufacturers and the incoming part test/inspection teams. Who cares whose 'fault' it was in the first place?

So most trees of failure have 'root systems' that look like this ...


*And the more POTENTIAL ROOT CAUSES we find for a particular failure mode, the better the chance we will find a cheaper, easier and faster CORRECTIVE ACTION (CA) to implement!*

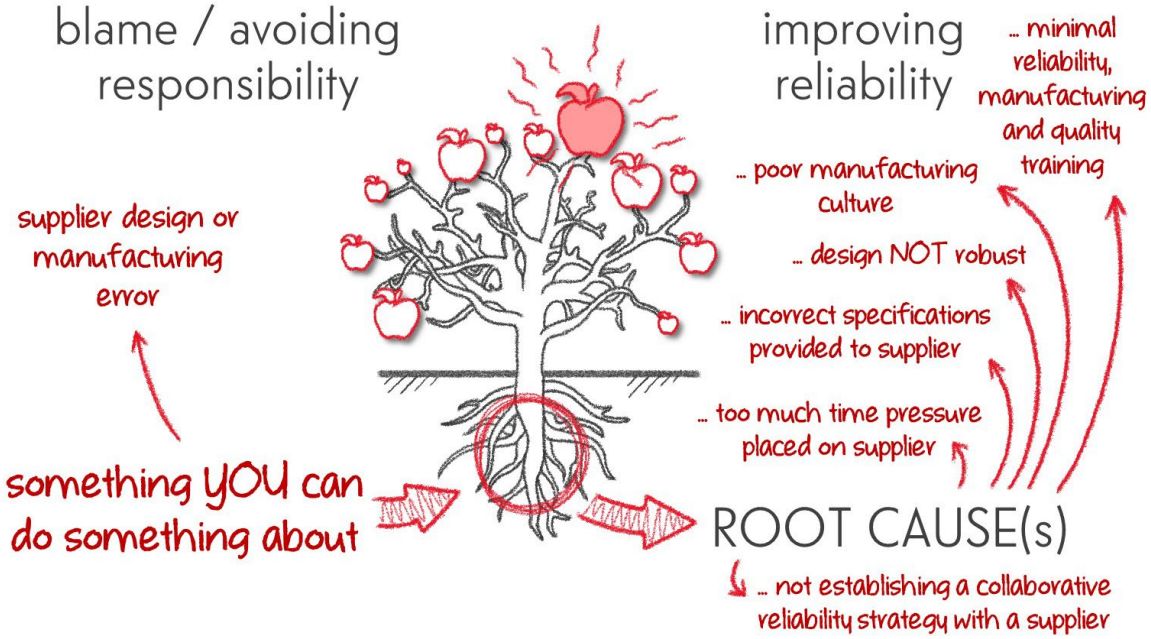
## … ROOT CAUSES and NOT APPORTIONING BLAME

Many **RCAs** devolve into an apportioning of blame exercise. Many organizations (inadvertently) encourage this behaviour where different teams and leaders are personally incentivised for the 'performance' of their team. These organizations very quickly create cultures where every 'issue' needs to be allocated to a particular person or group of people for the purpose of monitoring (so called) performance. It then becomes in the interest of team leaders to argue against 'blame' … which by extension requires them to argue that the issue in question was 'someone else's fault.'

These organizations then ALWAYs evolve a sophisticated blame avoidance system, where suppliers, providers, subcontractors, customers, the government, schools and universities are ultimately allocated the blame for issues. This way all team leaders can argue that they didn't create 'any issues' to protect the perceived performance appraisal (and rewards).

The problem with blaming others (especially third parties) is that we never find true root causes. Third parties in particular are essentially part of the commercial 'environment' organizations operate in. By extension, our ability to do directly change their behaviours is non-existent.

BUT … this doesn't mean we can't influence their behaviour by first looking at 'us.' This diagram below contains a 'true' root cause on the right where we focus on something 'we did wrong.' The **CAs** we come up are things we can do to help influence supplier behaviour. So (perhaps counter-intuitively) if we look at what we can do first, we often have a positive effect on the third parties we so easily blame. As a rule, suppliers and third parties want our business. They are more open to working with you if you move beyond simple apportioning of blame.



Unfortunately, some of the root causes of failures are actually people in YOUR organization. There are three broad types of people that 'get in the way' of good reliability engineering outcomes. So THEY become the root causes. We call these three types of people the **enemies of reliability** … who we will discuss below.

*… the PONDEROUS PROFESSOR*

Analyzing (or admiring) a problem doesn't solve it. Sure … understanding a problem is often a great first step. But you need to take lots of steps thereafter. Never fool yourself into thinking that analysis effort is equivalent to an outcome. **Reliability engineering** means **improving** design, manufacturing, or maintenance to generate value through **improved reliability (and availability and maintainability) performance**. This doesn't involve completing the nearest, most complicated equations - these are only important some of the time, and for some situations.

One of the reasons people perceive ALL reliability activities to include (painfully) complex analysis activities is because of our **ponderous professor**. Good **reliability engineering** involves focusing on the decision you are trying to inform and then working out how to get that information. And if you are focusing on improving reliability and not measuring it … this is often easier than you might think.

*... the INFANT MANAGER*

The infant manager's most famous catchphrase is '**WTF!**' ... which stands for '**give me the WRONG THING FAST!**' He or she is focused on false metrics, meeting the first milestone as quickly as possible, cutting costs from the start and other things that provide the illusion of blindingly fast progress. But this illusion never lasts, because he or she pressures the team into not considering all those really simple and inexpensive design features that prevent problems.

*... the PROCESS ZEALOT*

Many organizations fail to make reliability happen because they keep everything complicated. They don't focus on their organization's *real* strategic goals, which means they don't create a simple strategy that allows everyone to understand their role. This is where our third and final enemy of **reliability engineering** can get in the way – the **process zealot**.

**Good reliability engineering is engaging and rewarding**. It is all about selecting the VITAL FEW (effective) reliability tools to drive high quality and reliability based on the circumstances of the program being supported, based on reliability plans (that work) that identify and deploy the right tools for the program.

# ROOT CAUSEs and CORRECTIVE ACTIONs (CAs)

To address a root cause, we need to find a **Corrective Action (CA)** or fix, which is ...

> *... an action that involves a change to a system design, operation, maintenance procedures, or the manufacturing process of the item for the purpose of improving its reliability.*

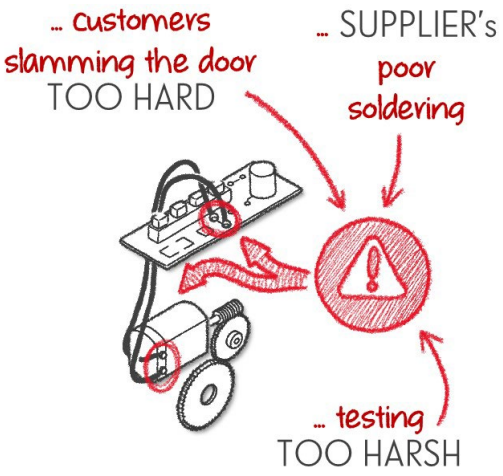
This is the thing that you are going to do differently in the future to ensure that the **failure** we are examining hardly occurs again.

Finding the **CA** is the most important part of **RCA**.

> *Simply finding the root cause and nothing else is at best ADMIRING THE PROBLEM.*

What we want to do is find the VITAL FEW **CAs** that have the biggest (likely) impact on future **failures**. There will always be the VITAL FEW and the TRIVIAL THOUSANDs. So we should always be striving to find the couple of weak points that drive system failure characteristics.

**CAs** are based on **root causes** which are (by definition) things we can control. The following are NOT **root causes** behind the smart lock prototype **failures**.




If we were to improperly arrive at the (clearly not) **root causes** above, we would never be able to find **CAs** that have any meaningful impact. We (by definition) can go no further than apportioning blame. And if everyone else gets the blame, then nothing ever gets done to address it. Supplier's (perceived) poor soldering or customers slamming the doors will inevitably happen again. So our smart lock will **fail** again. Meaning we pay warranty costs, lose customers and ultimately go bankrupt.

What we want is …


And the earlier we come up with **CAs**, the cheaper they become. If we find them early enough, they become virtually free. The next chapter will involve us practicing our **RCA** by examining the smart lock further in a worked example.

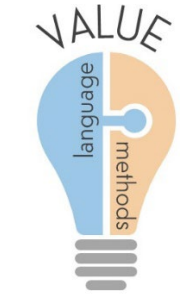
## … the RELIABILITY MINDSET

This book started with using **fault trees** to help us analyze **system reliability** by creating a **system reliability model**. This **model** allows us to convert **component reliability characteristics** into **system reliability characteristics** that help decision-making that include (but aren't limited to) determining warranty periods or optimizing **Preventive Maintenance (PM)** or servicing intervals. This is the '**first perspective of FTA**' that was discussed in Chapter 1.

The previous two chapters we looked at how **fault trees** can be used to help **RCA**, where the focus is on IMPROVING **reliability** more than MEASURING it. This comes about from finding a wide range of potential **CAs** that help us quickly, efficiently and cheaply address the VITAL FEW weak points of our product, system or service. This is '**second perspective of FTA**' that was discussed in Chapter 1.

We are now going to start looking at how we use **fault trees** to create **robust, customer-centric designs**. This is the '**third perspective of FTA**' that we discussed in Chapter 1. We talked about how we might want to use **FMEAs** to identify likely failure modes BEFORE we start designing so that the **CAs** we come up with are embedded in the first design. We talked about how we might want to use testing like **Highly Accelerated Life Testing (HALT)** that deliberately pushes early prototypes and parts of our system beyond their limits so we can identify their weakest points.

But this only happens if we are motivated culturally to find failures as early as possible in the production, design and manufacturing process. This is where the concept of the '**reliability mindset'** becomes important.




THE
RELIABILITY
MINDSET

The **reliability mindset** starts with all of us having a common LANGUAGE of **reliability** and **quality** that allows us to talk to each other without any ambiguity and with full understanding of what others are trying to say. The reliability mindset then involves the METHODS that include the VITAL FEW tools that we and our organization have judiciously identified as being the right tools. This allows us to make better decisions without feeling overwhelmed to maximize value.

**Fault trees** are one of many tools that **FMEAs** can use to identify issues that affect the ability of your product to meet customer expectation.

EXERCISE

**Let's go back to the smart lock and really focus on one of the most basic customer expectations.**

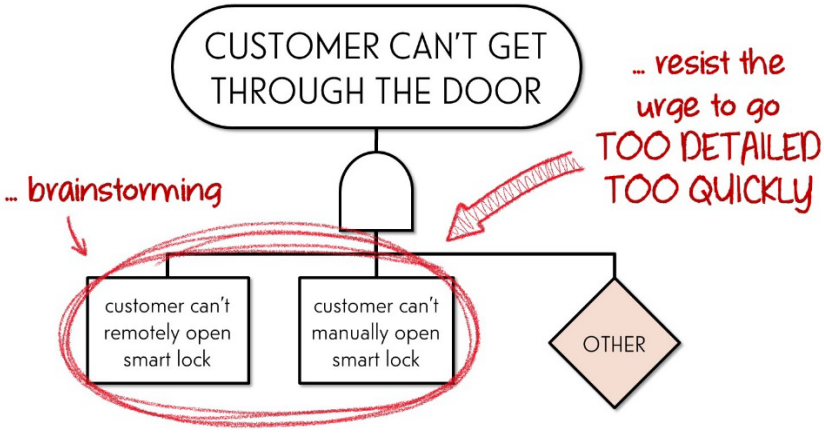

**This might seem quite obvious. But if we really think about getting the customer through the door, it allows us to think of scenarios where (for example) the customer doesn't do things 'right.' How does the customer get through the door if the smart lock's batteries are flat? It's not our fault the customer didn't replace them. So why should we care?**

**The world around us is random and 'imperfect.' We want our smart lock to appeal to all customers. And not the very few 'perfect' customers. So let's use fault trees to work out how we can make our smart lock more appealing to more customers.**

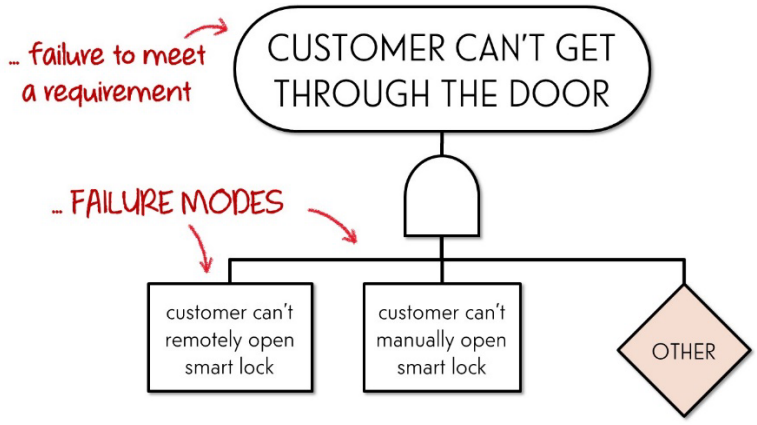
EXERCISE

We start with defining our top event.


We then go through our first round of brainstorming. In this case, we are going to use an 'AND gate' because our customer has several ways of getting through the door with our locked smart lock on it – and each of these ways needs to fail for her to not be able to make this happen.



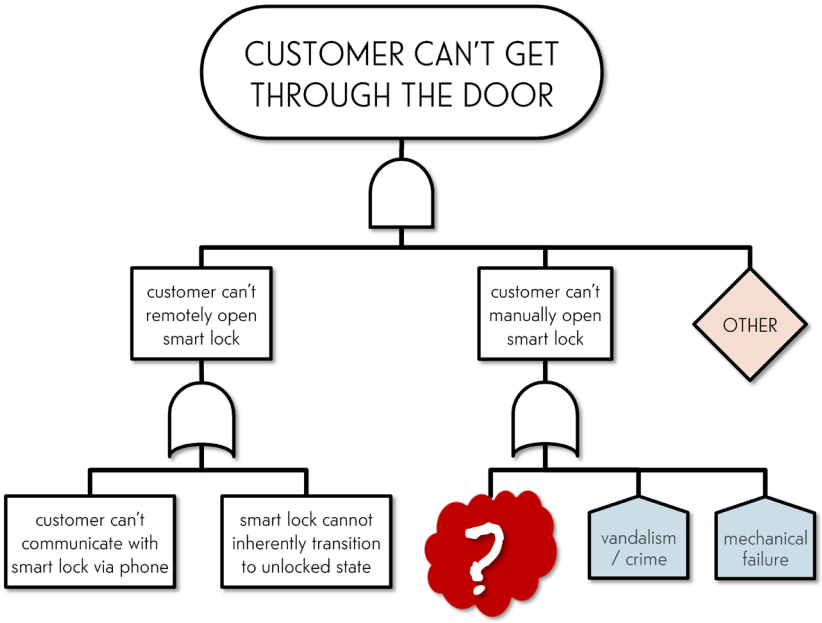

Another way of looking at this (particularly if you are doing this as part of a FMEA) is to consider the top event a 'failure to meet a requirement' and all the next level reasons for this requirement not being met becoming 'failure modes.' Failure modes are the manner in which something fails.




This can also become a really useful way of helping brainstorm specific elements of the failure causal chain. Remember, we don't want to get too detailed too quickly. If we do get too detailed, we exclude lots of other potential failure causal chains that might help us uncover all sorts of other really dangerous potential root causes that we would want to remove from the design of our system as early as possible.

EXERCISE

**Let's brainstorm one more level.**

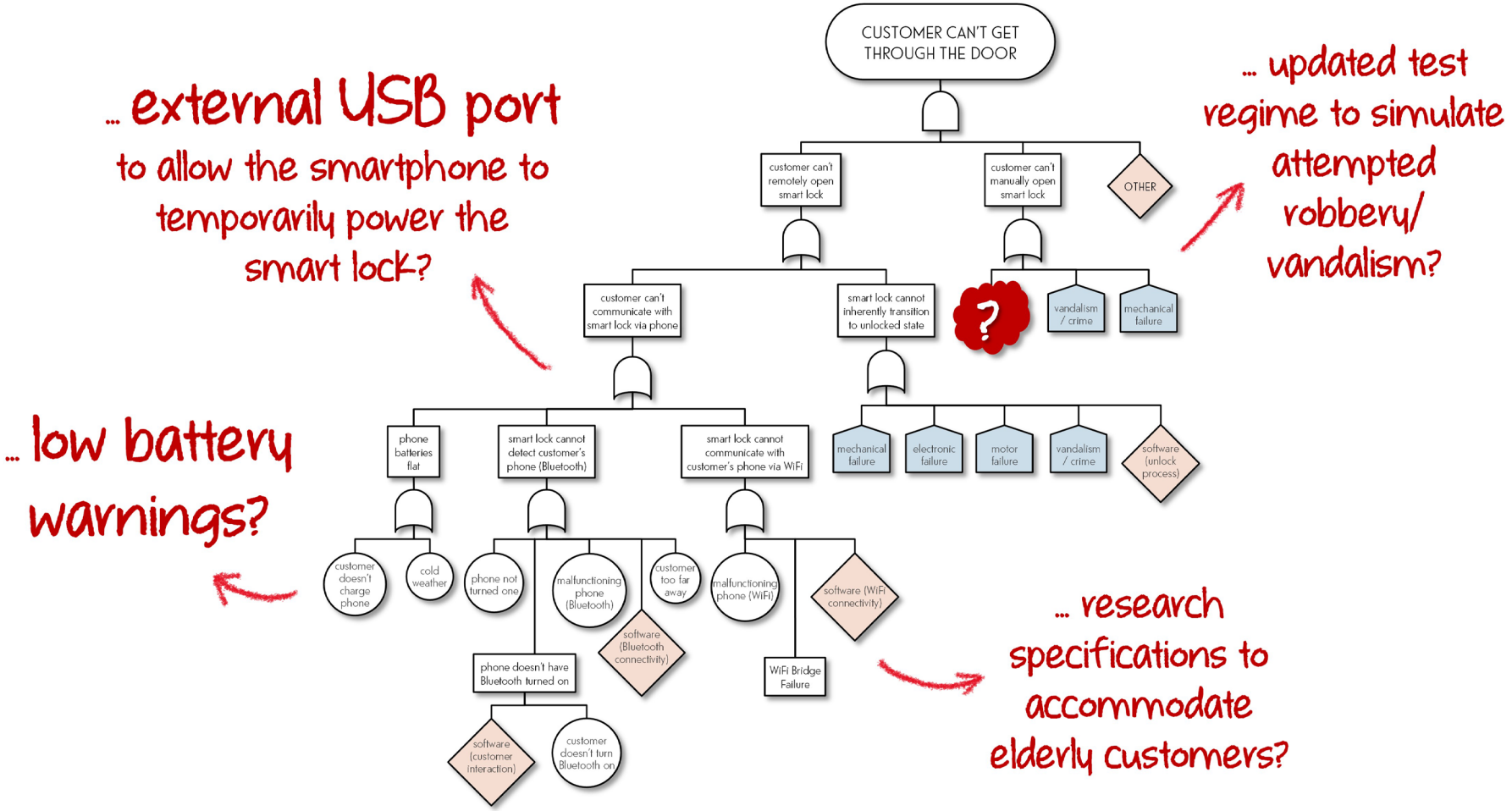



**What else could you put where the question mark is?**

**We can then keep exploring each input event until we reach basic events that represent the root causes of us not meeting our customer's expectations.**

**The fault tree on the following page shows how this fault tree could look once it has been fully analyzed. And by doing this, we have come up with some potentially useful design characteristics that can separate our smart lock from our competitors.**

# EXERCISE


CUSTOMER CAN'T GET THROUGH THE DOOR
customer can't remotely open smart lock
customer can't manually open smart lock
OTHER
customer can't communicate with smart lock via phone
smart lock cannot inherently transition to unlocked state
?
vandalism / crime
mechanical failure
phone batteries flat
smart lock cannot detect customer's phone (Bluetooth)
smart lock cannot communicate with customer's phone via WiFi
mechanical failure
electronic failure
motor failure
vandalism / crime
software (unlock process)
customer doesn't charge phone
cold weather
phone not turned one
malfunctioning phone (Bluetooth)
customer too far away
malfunctioning phone (WiFi)
software (WiFi connectivity)
software (Bluetooth connectivity)
phone doesn't have Bluetooth turned on
WiFi Bridge Failure
software (customer interaction)
customer doesn't turn Bluetooth on
... external USB port to allow the smartphone to temporarily power the smart lock?
... low battery warnings?
... updated test regime to simulate attempted robbery/ vandalism?
... research specifications to accommodate elderly customers?

**Fault trees** can be incredibly powerful design tools and help us …