





Safety Requirements

ISSC21 Ottawa



Dr Mike Ainsworth

Praxis Critical Systems Limited



Objectives

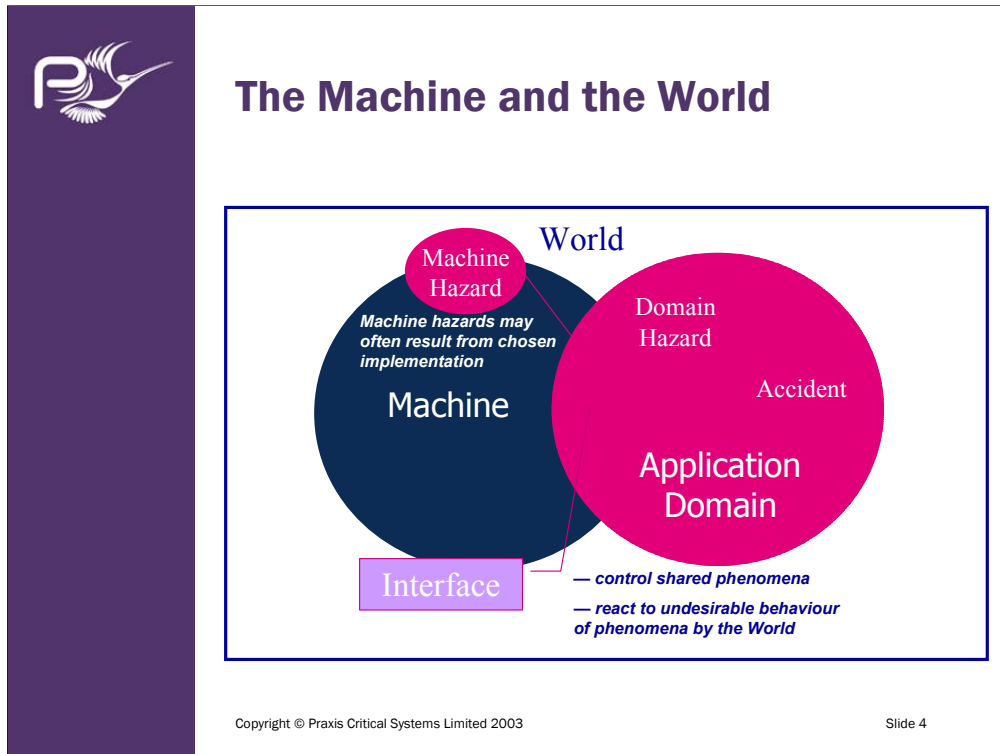
- Why do we write safety requirements?
- How do we make sure they're correct?
- How do we make sure they're complete?
- How do we decide if it's safe enough?



Requirements, Specification and Design

Copyright © Praxis Critical Systems Limited 2003

Slide 3



Accidents only occur in the World

A Hazard is a condition of the World from which accidents can occur

We define three specific classes of hazard:

- World Hazards that are outside the scope of the Application

- Machine hazards occur at the interface

- Domain hazards are those that occur in the Application Domain but are not directly related to the Machine

Accidents we need to consider can be caused by Machine hazards or Domain hazards or both

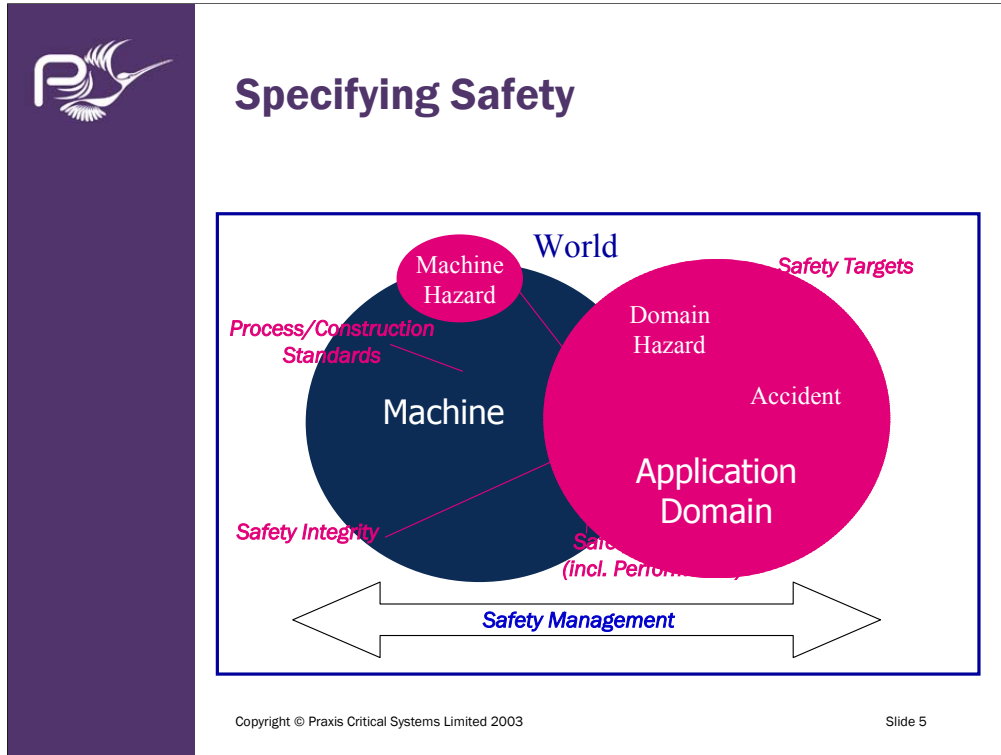
Accidents outside the Application Domain are outside scope

Machine hazards arise

- Due to failure to control shared phenomena correctly

- Failure to react to undesirable behaviour or Domain hazards

Choosing a Nuclear Power Station as a power supply can change the scope of the Application Domain.




Safety targets are about controlling accidents and domain hazards – they are external to our machine.

Safety functions and safety integrity define what the machine has to do to control risk, and how dependable that functionality has to be.

Process/construction standards apply to the design and construction of the machine – they are one way to achieve integrity, but not the only way.

Safety management standards underpin the whole exercise, e.g. requirements for competence of staff.



Requirements, Specifications and Designs

- *Requirements* (\mathcal{R}) are a collection of statements about things in the world (W) that we want the machine to make true
- A *Specification* (\mathcal{S}) describes the machine's external behaviour.
 - it includes only shared phenomena in the interface (I)
 - it defines only things the machine itself can control
- A *Design* is a statement about the machine itself. It describes things in M .

Copyright © Praxis Critical Systems Limited 2003 Slide 6

We use sans-serif letters like W, M and I to refer to *phenomena* : things and events in the world, in the machine and in their interface.

We use script letters like \mathcal{R} and \mathcal{S} to refer to *descriptions*: statements about phenomena.

Requirements are statements about things in the real world that we want to bring about. They need not be *directly* achievable by the machine. In some organisations they may be called Mission Needs or User Needs.

Specifications define what direct effects the machine will have on the world. In some organisations these are called “System Requirements Specifications” or, where the machine is built in software, “Software Requirements Specifications”.

There are different terminologies in use, but these are used consistently throughout the course.

The following table shows approximate equivalences between the terms used in REVEAL and those used in other organisations.


REVEAL...	Sometimes called...	Or...
Requirement	User Requirement	Need
Specification	System Requirement	Requirement



Exercise: The REVEAL Game

Copyright © Praxis Critical Systems Limited 2003

Slide 7



What is and What Should Be

- We can make two kinds of statement about phenomena in **W**:
 - *Facts*
 - *Indicative* statements of domain knowledge **D** describe properties of the world which are (or are assumed to be) true
 - *Wishes*
 - *Optative* statements describe the properties we *want* to be true - our requirements **R**

Copyright © Praxis Critical Systems Limited 2003 Slide 8

If the machine can not achieve the requirements directly, then how can they be achieved?

Almost always, we rely on properties of the real world to connect the actions of the machine (like switching on a red light) to some desired result (like stopping traffic). These properties are called Domain Knowledge, **D**. (Remember that we sometimes call the part of the world we are interested in the *application domain*.)

We **MUST** state, as part of the requirements definition, what these assumed domain properties are.

Our specification is correct if and only if we can show that given the assumed domain properties and the specified behaviour of the machine, then we can deduce the requirements themselves.

We call this reasoning a *satisfaction argument*. The logical deduction

$$D, S \Rightarrow R$$

formalises this idea. It means that the combination of domain knowledge **D** and specification **S** is enough to satisfy the requirements **R**.



Satisfaction Argument

- A machine satisfies the requirements if and only if

$$D, S \Rightarrow R$$



Where do we go wrong?

- Many system failures are not failures to understand \mathcal{R} ; they are *mistakes in \mathcal{D}*
 - A NYC subway train crashed into the rear end of another train on 5th June 1995. The motorman ran through a red light. The safety system did apply the emergency brakes. However the ...signal spacing was set in 1918, when trains were shorter, lighter and slower, and the emergency brake system could not stop the train in time.

Copyright © Praxis Critical Systems Limited 2003

Slide 10

Another example, quoted by Michael Jackson, is a real air crash. Here is a grossly simplified account:

The requirement was that if the aircraft was moving on the ground then the pilot should be able to apply reverse thrust

The specification of the aircraft control systems was that if the wheels were rotating then the pilot should be able to apply reverse thrust

This depended on a domain assumption that if the aircraft was moving on the ground then the wheels would be rotating with weight on them. Unfortunately this was false! The plane landed on a wet runway and some of the wheels did not have weight on or were not turning. So the pilot couldn't apply reverse thrust, so the plane ran off the runway.

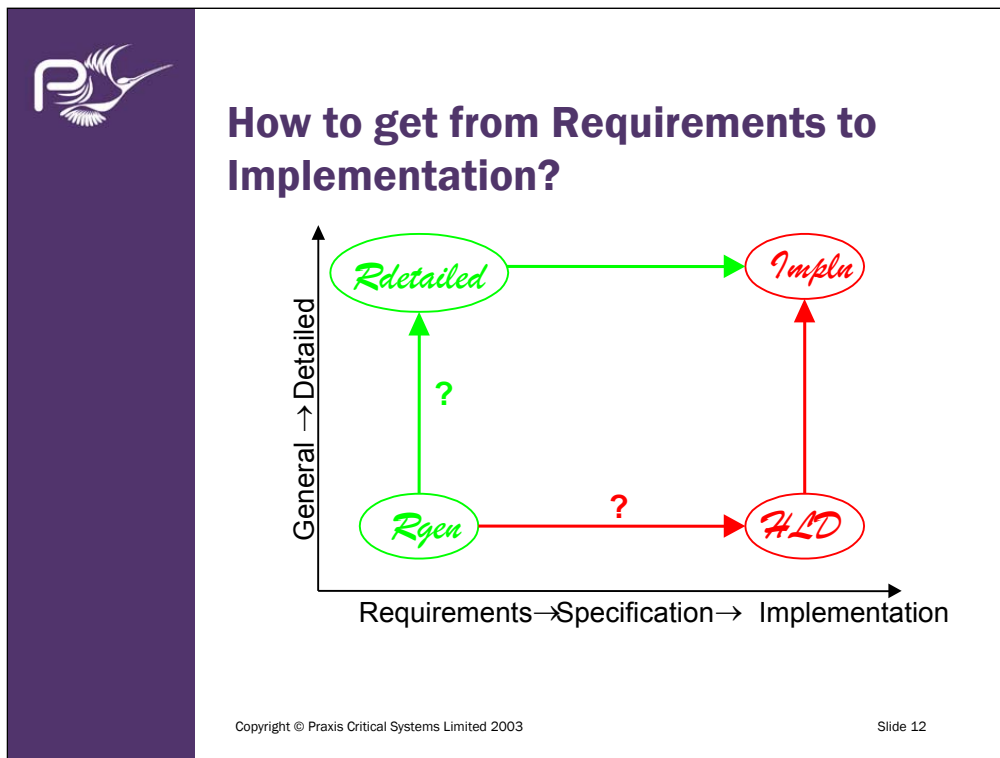
Although we have not done a systematic survey, it is easy to find examples of system failures arising from incorrect domain assumptions. These examples underline the importance of making such assumptions explicit. Writing them down does not necessarily mean that mistakes will be discovered, but not writing them down more or less guarantees that they won't.



From Requirements to Implementation

Copyright © Praxis Critical Systems Limited 2003

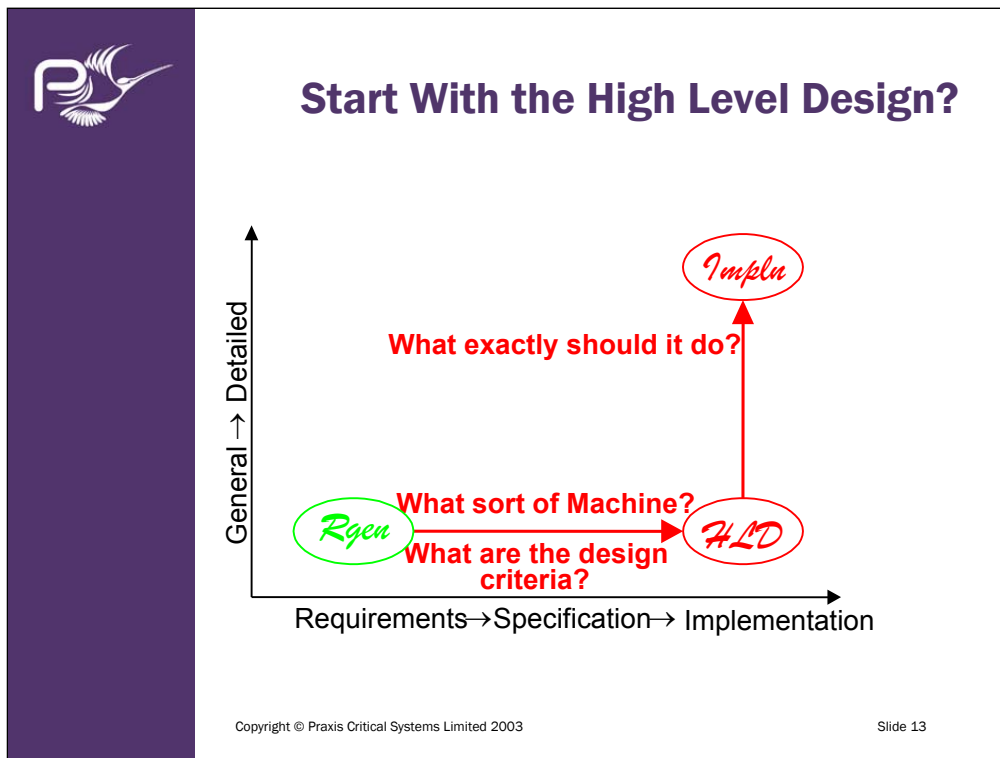
Slide 11



In the bottom left is the initial statement of overall requirements (*Rgen*), such as “improve air traffic flow over oceanic airspace”. This very general statement is sometimes called “Mission Needs”

To get from here to a machine implementation we have to travel along two dimensions: to increase the amount of detail, and to move from the world to the machine. Notice that these are *different* dimensions: we can and should be precise about what a user wants without saying how it is to be achieved.

Two routes are shown on this diagram, but neither of them is realistic.

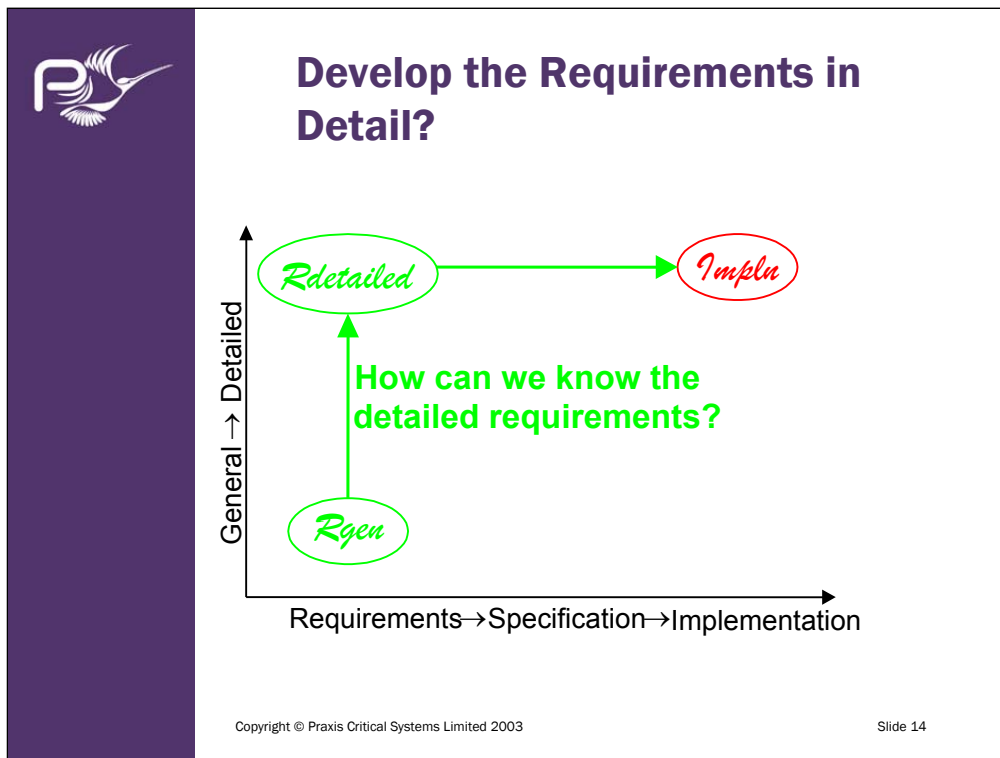


The bottom route is the one all too commonly taken.

It goes straight from the general statement of requirements to a high level design of the machine (*HLD*), and from there to the full implementation.

The problem is that the general statement of requirements gives little clue about how to generate a high level design. Indeed, it may give no hint at all as to what sort of machine might be used to meet the requirements: should it be a new ATC system, or new legislation which encourages airlines to fill their planes more efficiently? All the design decisions in going from *Rgen* to *HLD* are therefore suspect.

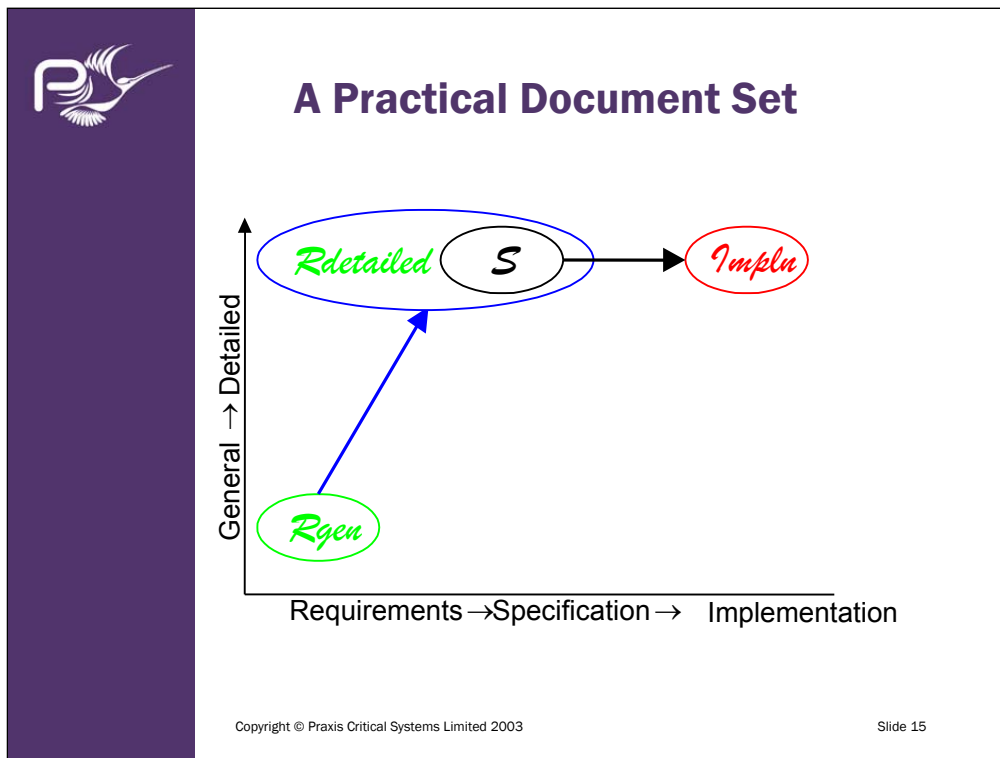
A high level design describes the structure of the machine but does not say what it will do. The detailed design cannot be derived without further understanding of the requirements - but by now the project is in the hands of the developers, not the customer.



The top route might seem better: it develops a detailed statement of requirements before doing any design.

However, this is also problematic, because to add detail to the requirements we need to have some idea about what sort of machine we are talking about - is it an ATC system at all? Maybe we would be better off making aircraft hold more passengers?

Until we have some idea what sort of machine we are trying to build, we can't say what its detailed requirements are. Only when we have decided that we are going to build an ATC system can we start talking about requirements for aircraft separation, controller communications and so on.

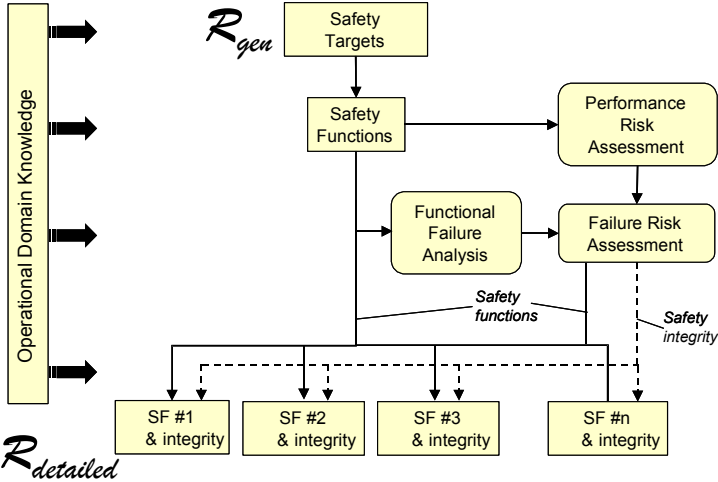


Requirements and specifications have to be developed hand in hand. The most practical route, therefore, is the one shown on this diagram. Starting from the general requirements such as “improve traffic flows”, we refine these into more specific requirements such as “reduce separations”, at the same time specifying what the ATC system needs to do to reduce separations. We thus arrive at a collection of detailed requirements which include a system specification. That system specification can be given to a supplier to design and produce the machine itself.

Typically the specification here would be based on a functional model – a set of functions (e.g. conflict detection) which are concrete enough that detailed requirements can be set, but without getting into implementation details (i.e. we don’t say whether the conflict detection is performed by a human or a machine).




Setting Safety Requirements





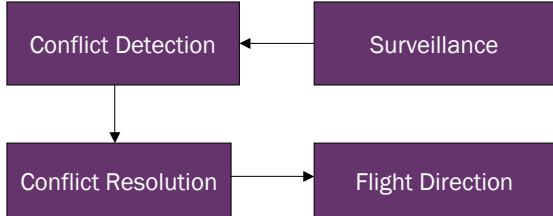
Example: ATC Safety Requirements

- R_{gen} (Safety Target)
- The likelihood of an aircraft accident caused by ATC shall be less than 1.55×10^{-8} per flight hour.



Example: ATC Safety Requirements

- *Specification*



- Conflict detection shall provide alerts when aircraft are within 10, 7, 5 and 3 NM of each other.

Copyright © Praxis Critical Systems Limited 2003 Slide 18

One way of defining the detailed requirements is to use a functional model of the system. The requirements are defined around a simple block diagram of the system, which defines the abstract functions that need to be provided.

This has the advantage of being low-level enough that it is meaningful to users (so they can discuss the requirements in their language), but is not tied to a specific implementation (we haven't said whether conflict detection is performed by a human or a computer). This ensures that although the implementation may change, the requirements should stay the same.



Functional Safety Requirements

Copyright © Praxis Critical Systems Limited 2003

Slide 19



Functional Safety Requirements

- Identify the functions required.
- Next consider how well those functions have to perform



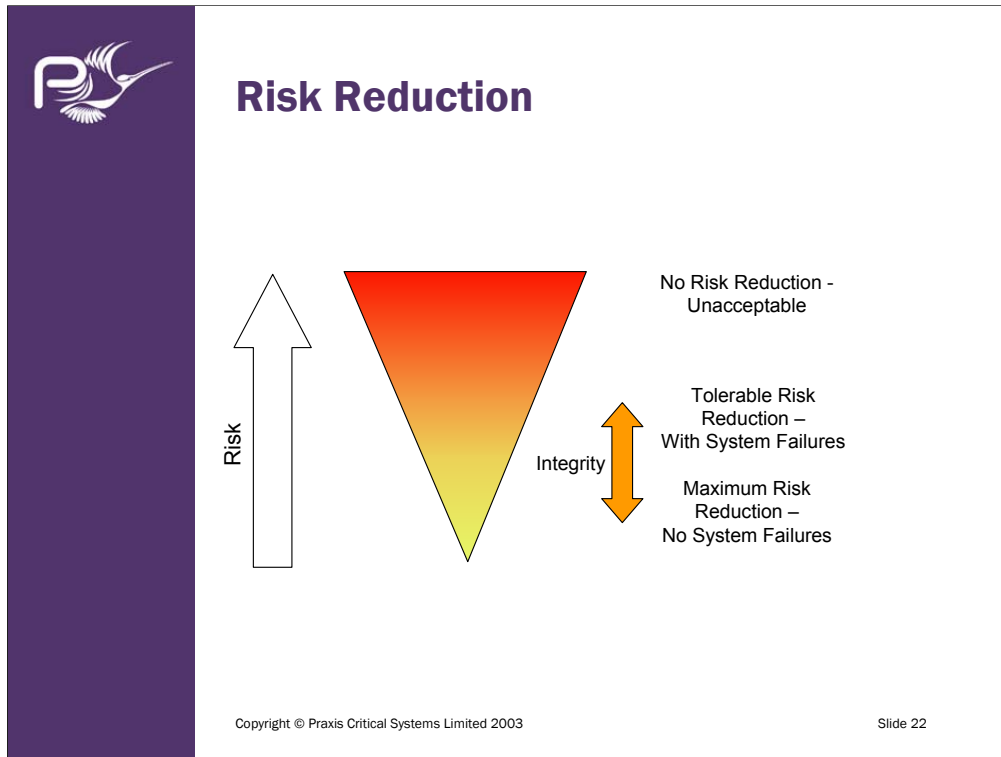
Safety Functions

- Functionality
 - Analysis of correct operation
 - Defines what each safety function must achieve
 - Function
 - Timing
 - Accuracy
 - Capacity etc.
- Integrity
 - Analysis of functional failure
 - Defines how dependable the function must be

Copyright © Praxis Critical Systems Limited 2003

Slide 21

Note that to some extent there is a trade off between functionality and integrity. If the range of acceptable behaviour is broad, then meeting the integrity requirements tends to be easier.



Without the system in place, the risk would be unacceptable.

When the system is performing perfectly (with no system failures), then it delivers its maximum risk reduction. This has to be tolerable (otherwise we need to rethink the entire project).

When we include the risks from failure then the risk must *still* be tolerable. The gap between the two (how well we can tolerate failure) defines the integrity required.

Defining the tolerable limit is difficult, but typically would either be by reference to a regulator or standard, by comparison with known death rates for the given industry, or by using past statistics (we want our accident rate to be no worse than it is at present).



Performance Risk Assessment

- Assessment of the level of performance required to ensure safety in normal operation (without failure).
- Risk target has to be more onerous than the tolerable limit
 - System must remain safe in the presence of predictable failure
- Dependent on extensive domain knowledge.



Performance Risk Assessment

- For minor changes
 - Demonstrate that existing standards are tolerably safe (when failure is excluded)
 - Set performance requirements to match or exceed existing standards

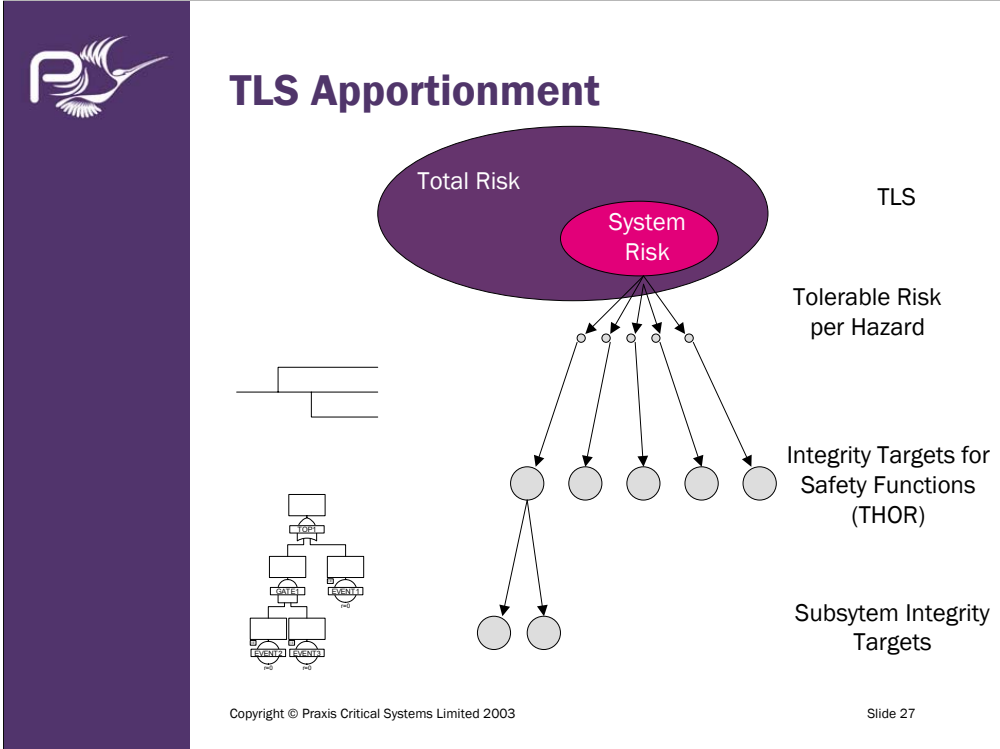


Failure Risk Assessment

- Identify hazardous failures
- Evaluate consequences and mitigations
- Each mitigation must be recorded, either as a safety function or as domain knowledge.

Example: Event Tree

Loss of flight direction to a single aircraft	Aircraft currently separated	Conflict detection allows other aircraft to be kept clear	Establish other communications or use a standard aircraft	Risk of aircraft collision once separation lost	Consequence	Frequency
w=1.000	Q=1.000e-3	Q=1.000e-2	Q=1.000e-1	Q=1.000e-3		1.000
	Success	Success	Success	Null	Successful ATC service provided	8.901e-1
	Success	Success	Failure	Null	Traffic separated, situation will be resolved by aircrew procedures	9.890e-2
Failure	Success	Failure	Null	Success	Near miss	9.980e-3
	Failure	Null	Null	Failure	Mid-air collision	9.990e-6
	Failure	Null	Null	Success	Near miss	9.990e-4
	Failure	Null	Null	Failure	Mid-air collision	1.000e-6



This is explained in more detail on the following slides.



Example: TLS Apportionment


EUROCONTROL ATM Safety
Target = 1.55×10^{-8} per flight
hour

Identify hazards from our system
and its interfaces (including
aircraft)

Check scope – some hazards are
out of scope and do not contribute
to TLS (e.g. deliberate pilot actions)

Check scope – some hazards are
caused by the aircraft but must
be mitigated by ATC (e.g. pilot
reaction to a TCAS alert)

$$TLS_{ATC} = TLS_{Total} - Risk_{Aircraft}$$



Example: TLS Apportionment

Divide the ATC risk by the number of contributing hazards – this gives the tolerable risk per hazard.

$$P(\text{Accident} \mid \text{Hazard})$$

For each hazard, calculate the likelihood of the hazard leading to an accident – use generic assumptions for ‘nuisance’ failures, e.g. 1 in a million chance of an accident.

Also need to calculate the severity of the accidents that arise.

Tolerable Hazard Occurrence Rate

$$\text{THOR} = \text{MaxRiskPerHazard} \times P(\text{Accident} \mid \text{Hazard}) \times \text{Severity}$$

Copyright © Praxis Critical Systems Limited 2003 Slide 29

Dividing the risk by the number of hazards assumes that all hazards are defined at a similar level. It is a reasonable way to develop initial requirements, but may need to be revised later in the lifecycle. If a subsystem cannot meet its original integrity target then risk will have to be re-apportioned (increase integrity elsewhere to keep the overall risk within budget).

We also need to be careful to only include the hazards that can contribute to this particular risk target (if separate targets have been defined). For example, if we have a risk target for mid-air collisions, then we would not include ground surveillance systems in this part of the calculation – they would be covered by a separate risk target for ground movement collisions.

In ATC all accidents are regarded as catastrophic, so severity would not be included in the calculations. For other systems (e.g. railways where there is a difference between high-speed and low-speed collisions) the severity would need to assess the number of equivalent fatalities for each accident (e.g. 10 major injuries = 1 fatality, 10 minor injuries = 1 major injury).

For nuisance failures or failures with extensive external mitigation, this method can lead to very large THORs (e.g. a tolerable limit of several hundred failures a day!) – in such cases, the tolerable limit has to be defined pragmatically by considering the impact on user confidence rather than safety. Even if a system failure is demonstrably safe, a high failure rate would lead to a loss of confidence in the other (more critical) functions of the system.



Correctness and Completeness

Copyright © Praxis Critical Systems Limited 2003

Slide 30



Correctness and Completeness

- Satisfaction Arguments
- The Accident Tree



Satisfaction Arguments

R_{detailed}

- Provide 5NM separation between aircraft

Specification

- Radar System
- Azimuth accuracy of 2 degrees
- Range accuracy 3% of distance

Copyright © Praxis Critical Systems Limited 2003

Slide 32

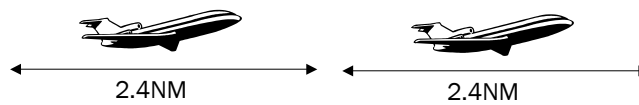
5NM (nautical miles not nanometres!) is the standard horizontal separation for aircraft.

The radar system accuracy specification is the UK minimum for an air-traffic control radar. In reality, most ATC radars are considerably better than this.



Satisfaction Arguments

- **Domain Knowledge:** Typically apply 5NM separation up to 70NM away from radar source.
- Azimuth error at 70NM is
$$\begin{aligned} & \text{Azimuth Error}/360 * 2\pi * \text{radius} \\ & = 2/360 * 2 \pi * 70 \\ & = 2.4 \text{ NM} \end{aligned}$$



At worst case error, if aircraft positions indicate 5NM separation the true positions cannot overlap.

Copyright © Praxis Critical Systems Limited 2003

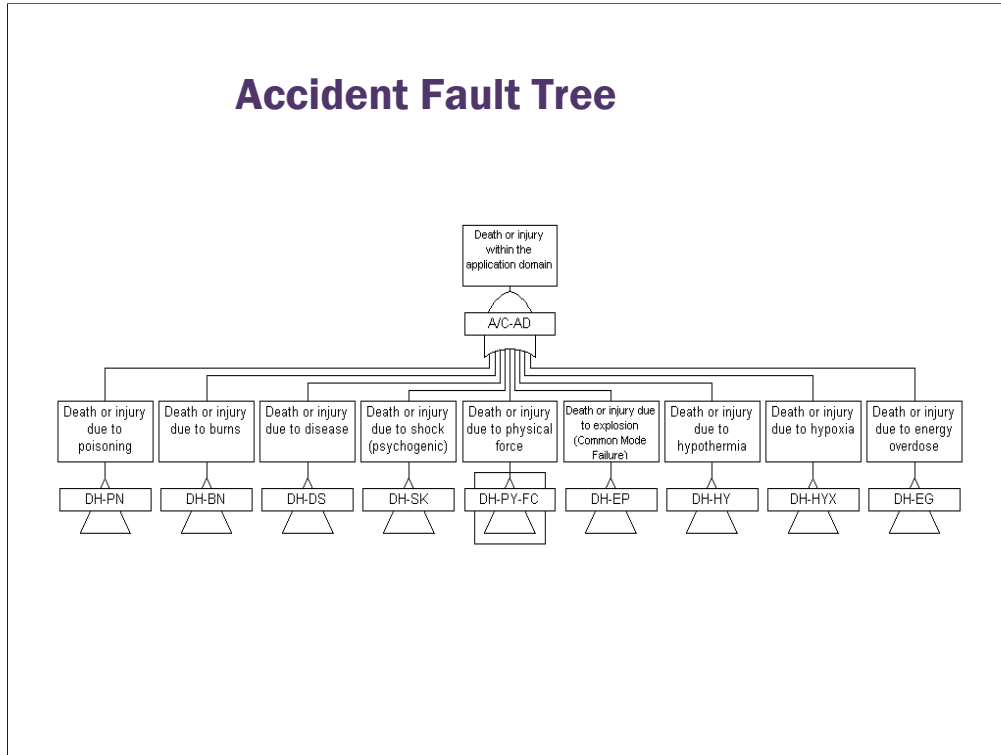
Slide 33

This is a very simplified example. The real satisfaction argument for this would be a lot more complicated and involves detailed assessment of the error distribution of the radar system.



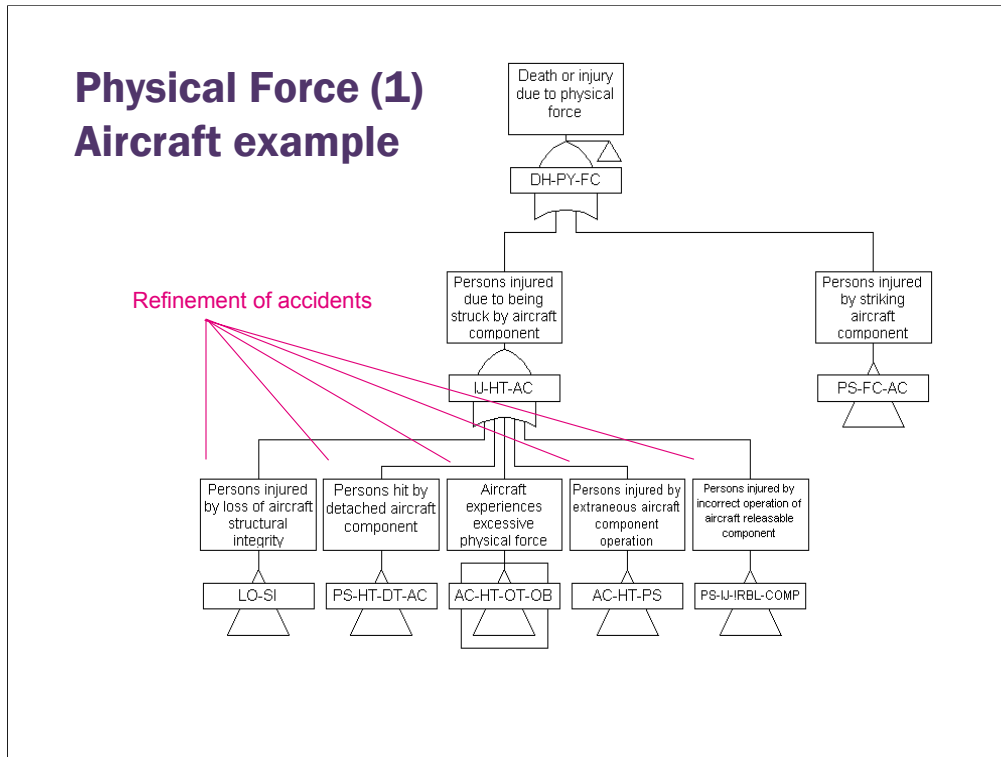
Satisfaction Arguments

- Trying to write satisfaction arguments often identifies:
 - Missing specifications
 - Missing domain knowledge
 - Specifications with no connection to the requirements
 - Infeasible or contradictory requirements



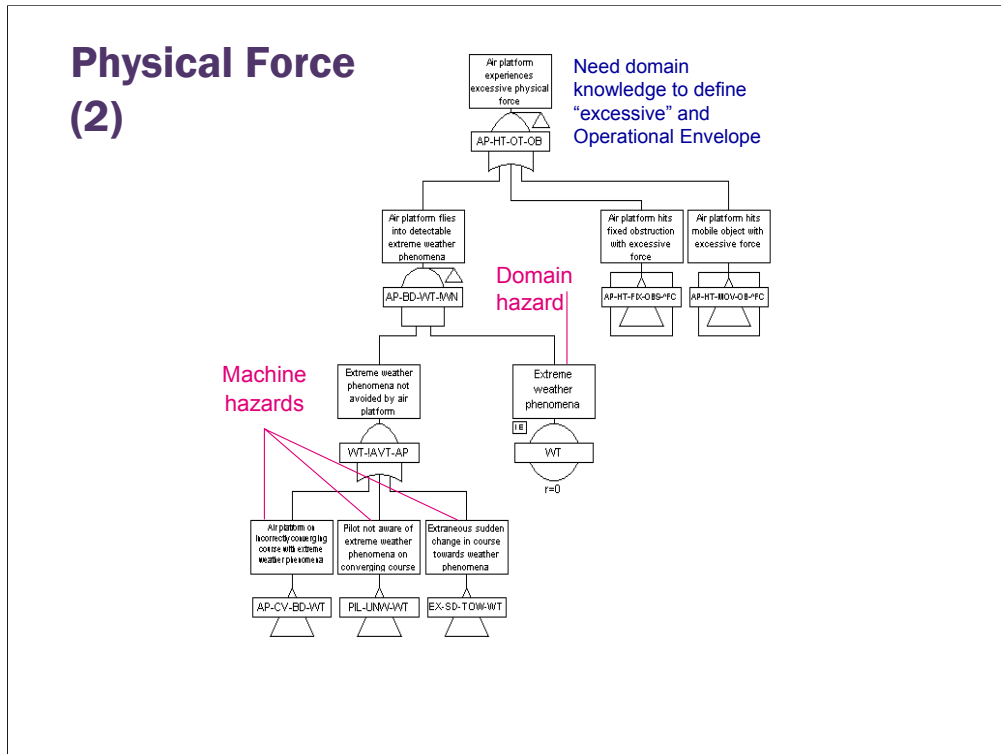
Begin with identification of high level accidents which may be relevant to almost any Application Domain, e.g. primary causes of death.

The physical force aspect of the tree can then be decomposed further as an example.



This is not a complete tree, but a slice to indicate the approach.

5 lower level gates represent refinement of the primary accidents towards identification of machine and domain hazards



Domain knowledge provides a definition of what constitutes excessive physical force in relation to the aircrafts operational envelope and its limits given that an aircraft cannot possibly withstand all weather phenomena

As defined previously, the Shared phenomena is the extreme weather detection mechanisms e.g. ATCO, weather radar

Machine needs to enter the weather phenomena for the accident to occur

Domain Hazard – state of the real world from which accidents can occur hence domain hazard is “Extreme Weather Phenomena”

Note that two of the Machine Hazards can lead to accident scenarios independently of this domain hazard



Accident Fault Trees

- Diverse check that all hazards and sources of risk have been identified.
- Can identify missing hazards and safety functions



Conclusions

Copyright © Praxis Critical Systems Limited 2003

Slide 39



Safety and System Lifecycle

- Understand the scope of the system – what risks are being mitigated?
- Integrity and performance requirements often drive system architecture – these need to be defined early on.
- Safety analysis is iterative – initial risk allocations may need to be revised as system design progresses.
- Hazards can arise from the design as well as from the functionality – some safety requirements may not be known until the design is complete.



Safety Requirements

- The safety requirements will be derived from a number of sources
 - the risk reduction process – e.g.risk targets and independence
 - hazard mitigation - functional and physical properties
 - constraints on design - standards, legislation, regulation, etc
 - constraints on development or management process
 - the resolution of any requirements or specification conflicts

Copyright © Praxis Critical Systems Limited 2003

Slide 41

The risk reduction process and hazard mitigation have been the focus of this tutorial, but there are other types which will need to be handled on any real project.

Constraints include design “must haves” e.g. Aircraft require TCAS in Controlled Civil Airspace, or a requirement to interface with existing equipment.

Constraints on development include process standards such as DO178B, DEF-STAN00-55.



What Kind of Statement Is This?

- All statement types are important
 - Domain knowledge, so you can justify the specification and prevent wrong assumptions about the domain.
 - All requirements even those not directly achieved by the machine, so you can justify the specification and prevent wrong assumptions about what is really needed.
- Each kind of statement plays a different role
 - Label each statement to indicate its role
- Be systematic in classifying statements
 - For example use a checklist

Copyright © Praxis Critical Systems Limited 2003

Slide 42

Here is a checklist for classifying any statement :

Is it a statement of fact which is true **whether or not** you build your machine? *If so, it is domain knowledge \mathcal{D} .*

Is it a statement of fact which is true **however** you build your machine? For example, it may constrain what the machine could possibly achieve.
If so, it is domain knowledge \mathcal{D} .

Is it something that we **want** to come true?
If so, it is a requirement \mathcal{R} . (It may also be a specification.)

Can the machine **by itself** make it true?

If so, is it observable in the world?

If so, it is a specification \mathcal{S} . (It may also be a requirement.)

If not, then it is part of the internal design and should not be in a requirements document.

If it is a requirement but not a specification, then can you find some specification \mathcal{S} and some domain knowledge \mathcal{D} which can make it true?

If so, you can prove $\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$.

If not, then it is an infeasible requirement.

If it is a specification but not a requirement, then is it something needed to satisfy some other requirement?

If so, then it will play a role in proving $\mathcal{D}, \mathcal{S} \Rightarrow \mathcal{R}$.

If not, why is it there?

If it is none of these things, why is it there?



Understand the Connections

- **Cause/Consequence Analysis** provides a way of analysing mitigations
- All mitigations need to be captured, either as **Domain Knowledge** or as **Requirements** and **Specifications**
- **Satisfaction arguments** provide a way of connecting together **Domain Knowledge**, **Specifications** and **Requirements**
- Provide a strong justification that the specification is sufficient to ensure safety

$$D, S \Rightarrow R$$



Functionality and Integrity

- **Safety requirements** have to specify the required **functionality** (including **performance**) as well as **integrity**.
- Functionality in normal operation must be tolerably safe.
- **Integrity** is about how often we can tolerate behaviour which does not meet the specification.



Praxis Critical Systems Limited
20 Manvers Street
Bath BA1 1PX
United Kingdom
Telephone: +44 (0) 1225 466991
Facsimile: +44 (0) 1225 469006
Website: www.praxis-cs.co.uk

Email: michael.ainsworth@praxis-cs.co.uk

Copyright © Praxis Critical Systems Limited 2003 Slide 45

Document Control

Praxis Critical Systems Limited, 20 Manvers Street, Bath BA1 1PX.
Copyright © Praxis Critical Systems Limited 2003. All rights reserved.

Changes history

Issue 1.0 (30th July 2003): Tutorial for ISSC21, Ottawa

Changes forecast

None.