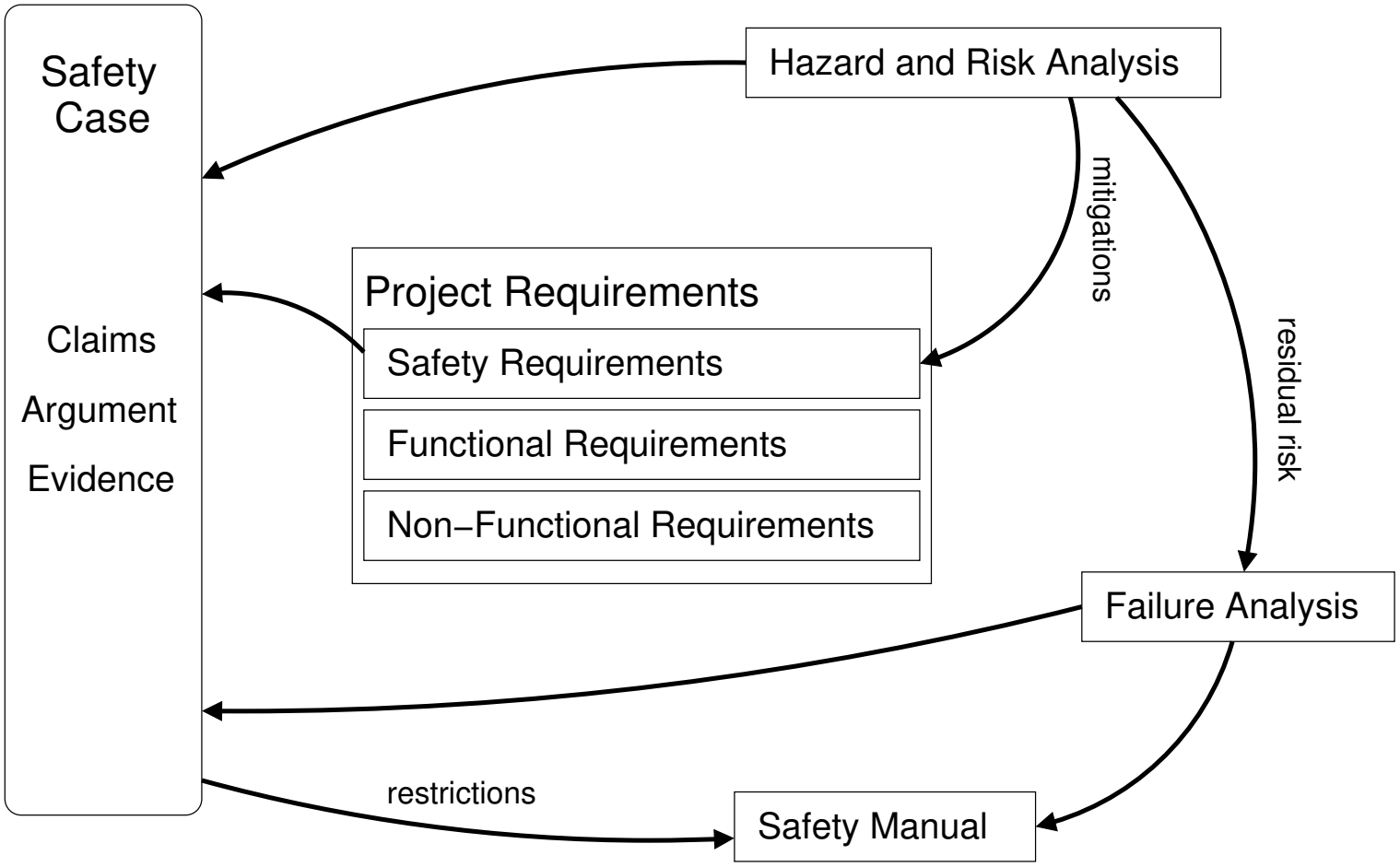
Lecture 5 — Risks, Hazards and Failure

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Overview



(Image reproduced from Hobbs'16)

Functional (or behavioural) Requirements

"define precisely what inputs are expected by the software, what outputs will be generated by the software, and the details of relationships that exist between those inputs and outputs. In short, behavioural requirements describe all aspects of interfaces between the software and its environment (that is, hardware, humans, and other software)."

–Software Requirements, Davis'93

"In Software engineering and systems engineering, a functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behavior, and outputs."

–Wikipedia

- Functional requirements state what a system **should do**
- Example: "**FR1***: user can control trains*."
- Example: "**FR2***: user can instruct system to route train from A to B.*"

Non-Functional Requirements

"Nonfunctional requirements define the overall qualities or attributes to be exhibited by the resulting software"

–Software Requirements, Davis'93

"In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors."

–Wikipedia

- Example areas: *security*, *reliability*, *efficiency*, *usability*, etc.
- Example: "**NFR1***: system should respond in reasonable timeframe*."
- Example: "**NFR2***: system should run on Java 5.0 or later.*"

Safety Requirements

"To achieve an acceptable level of safety for software used in critical applications, software safety engineering must be given primary emphasis early in the requirements definition and system conceptual design process."

–Software System Safety Handbook

Generic SSRs are those design features, design constraints, development processes, "best practices," coding standards and techniques, and other general requirements that are levied on a system containing safety-critical software, regardless of the functionality of the application.

–Software System Safety Handbook

- Example: "**SR1:** *system should employ software interlocking to prevent collisions*"
- Example "**SR2:** *upon detecting a failure, system should lock down network by halting all trains*"

Hazard and Risk Analysis

> *"The idea of this analysis is to* **identify risks** *associated with the component or device, to* **determine mitigations** *to reduce those risks, preferably to zero, and then determine what residual risks are left."*
>
> –Hobbs'16

- How to **identify** hazards?
- How to know have identified **all hazards**?
- How to **estimate** risk?

(Model) Railway Example

- **Hazard:** *Train may derail due to excessive speed.*
- **Severity:** *Catastrophic.*
- **Risk:** *Frequent.*
- **Mitigation:** *Ensure trains travel at safe speeds for track type (e.g. bend versus straight).*

Q) *What are* **residual risks***?*

See *"Analysis of Causes of Major Train Derailment and Their Effect on Accident Rates"*, Liu *et al.*, Journal of the Transportation Research Board, 2012.

(Model) Railway Example (Cont'd)

- **Hazard:** *Two trains may collide.*
- **Severity:** *Catastrophic.*
- **Risk:** *Occasional.*
- **Mitigation:** *Hardware and/or Software Interlock.*

Q) *What are* **residual risks***?*

Q) *What are* **safety requirements***?*

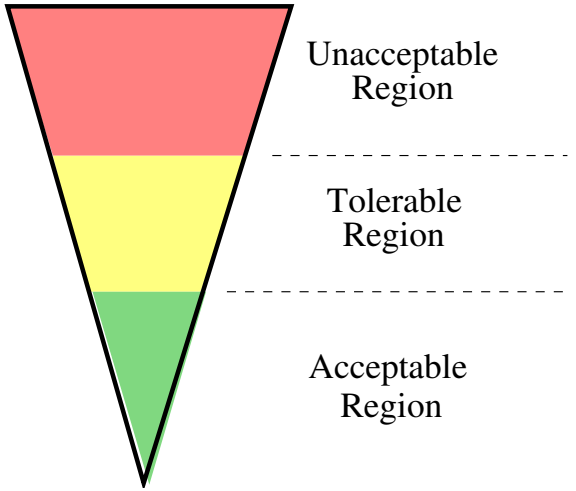
ALARP (outlined in IEC61508)

“ALARP stands for "as low as reasonably practicable", and is a term often used in the regulation and management of safety-critical and safety-involved systems. The ALARP principle is that the residual risk shall be reduced as far as reasonably practicable”

–Wikipedia

- Three regions:

  (Unacceptable) Risks which are too great to be tolerated.
  (Acceptable) Risks which are too small to be of concern.
  (Tolerable) Risks which fall between these extremes.



- Must reduce tolerable risks to lowest practicable level.

Failure Analysis

"All systems fail, and for safety-critical systems it is important to know **how often** *and in* **what manner** *they fail. Essentially this means considering the residual risks identified during the hazard and risk analysis."*

–Hobbs'16

- Examples:
  - *Sensor failure (e.g. intermittent or total)*
  - *Mechanical failure (e.g. with train)*
  - *Electrical power failure*
  - *Software failure in controller (e.g. exception thrown)*
  - *Hardware failure in controller (e.g. memory corruption)*

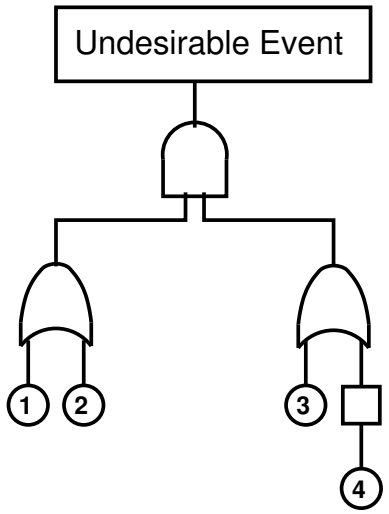
Failure Analysis: *Railway Example*

"The track is divided up into 22 distinction sections. These sections are further divided into detection *and* non-detection *sections with an alternating pattern being used. The detection sections are odd-numbered, whilst the non-detection sections are even numbered. The distinction is that detection sections employ a sensor to determine whether a locomotive is in the given section or not."*

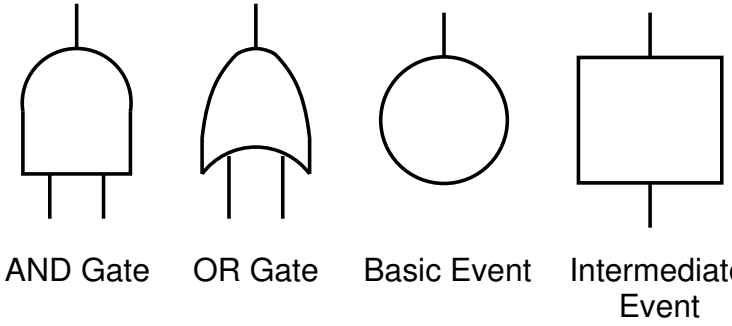
- Consider case for *single track-detection sensor failure*.
- What are possible outcomes?
- What is the risk?
- What are plausible mitigations?

Fault Tree Analysis (FTA)

- **Undesired event** analysed top-down with boolean logic

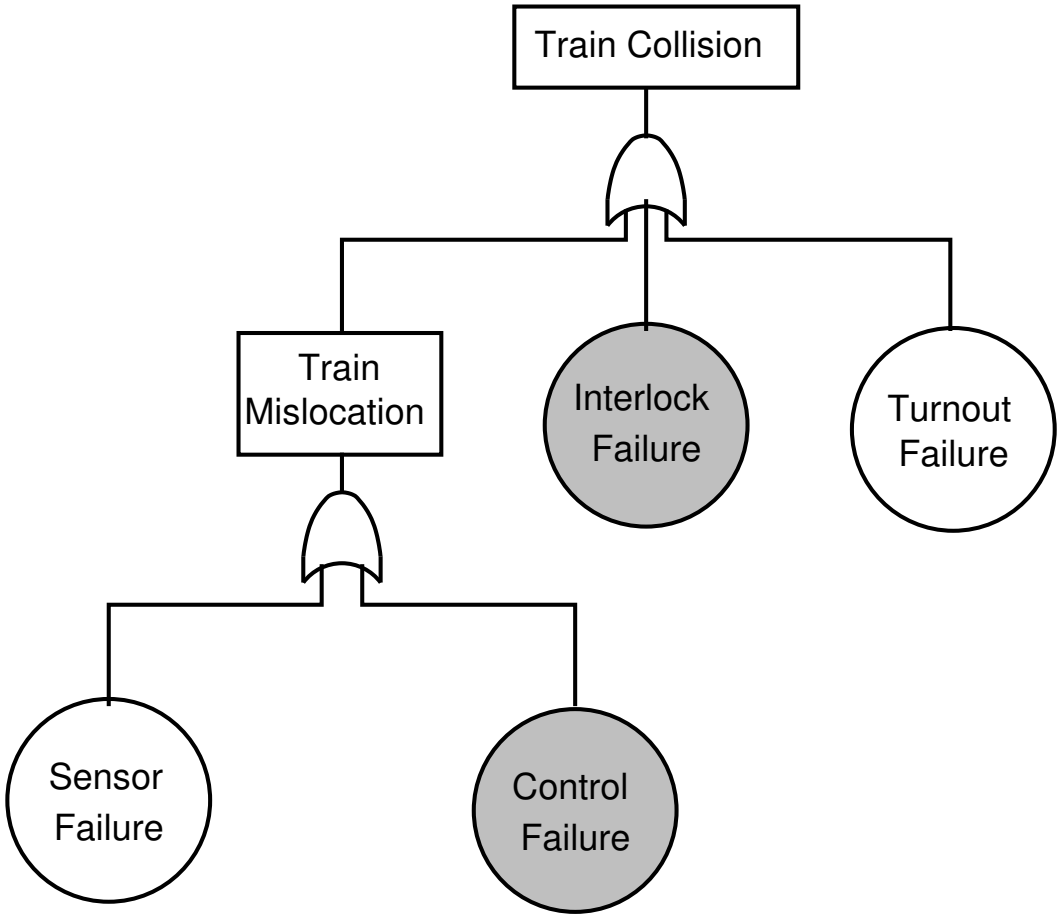


- Tree producing **events** from other events **via gates**
- Example gates:



- Originally developed at Bell Labs in 1962 by Watson

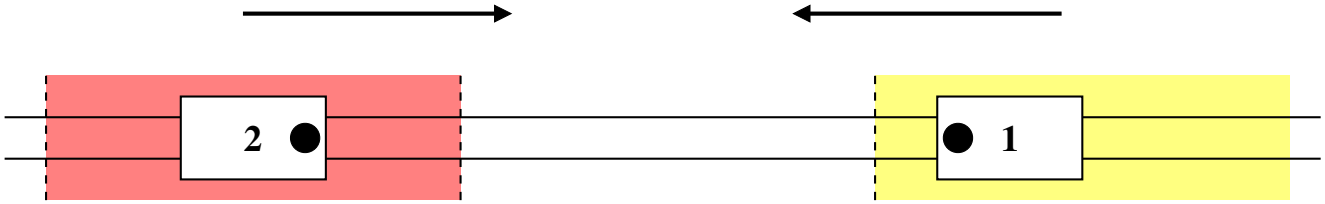
FTA Analysis for Railway Collision



- *What are we missing?*
- *What mitigations suggest themselves here?*

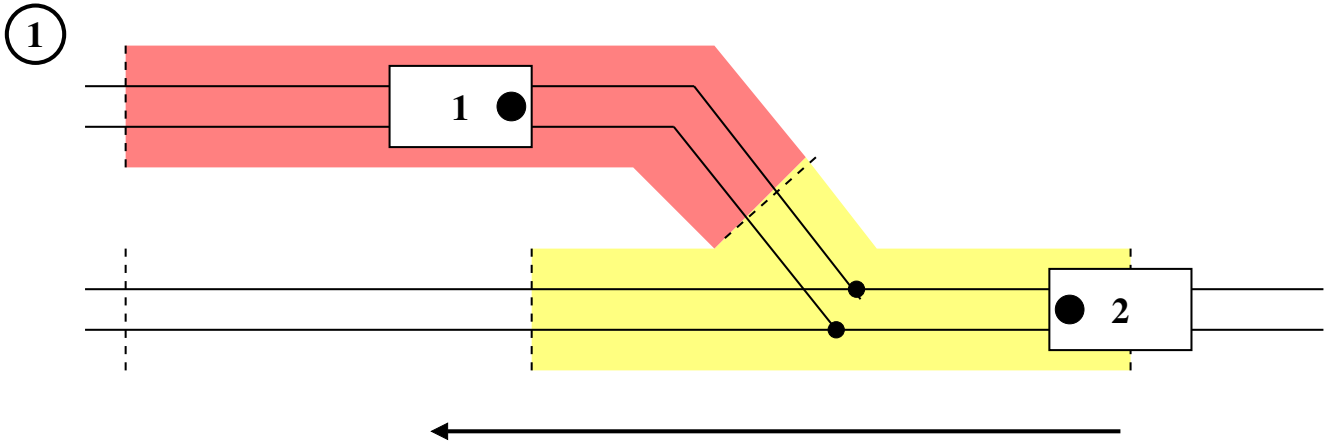
Failure Analysis: *Railway Example Cont'd*

- *Single sensor failure*:



(Trains 1 and 2 are head directly toward each other)

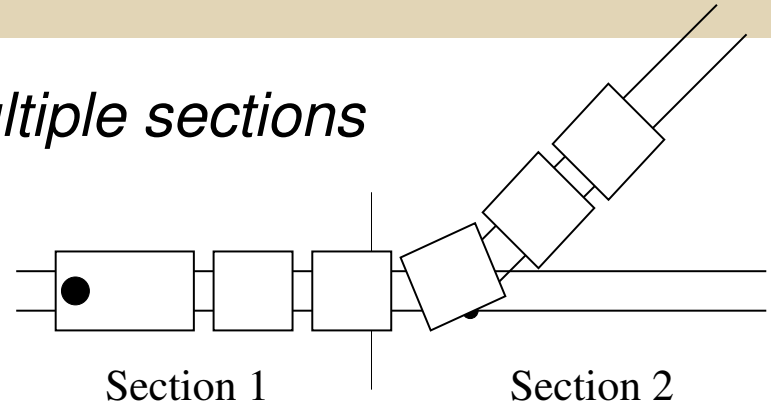
- *Turnout failure*:



(Train 1 waits for Train 2 to pass along mainline)

Failure Analysis: *What's missing?*

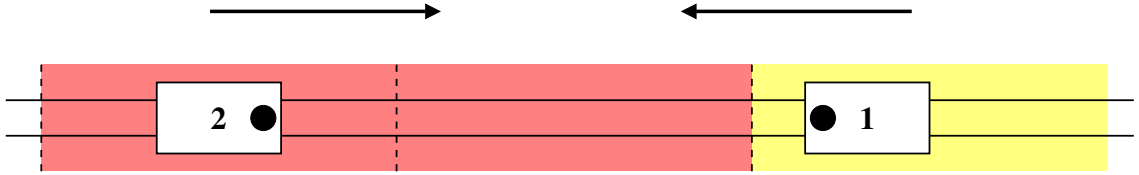
- *Train spanning multiple sections*



- *Train brake failure*

(Train 1 requested to stop, but brakes fail)

- *Train stopping distance*



(Train 1 requested to stop, but insufficient stopping distance)

Software Fails Predictably?

Bohrbug

"A Bohrbug is a nicely defined, solid bug with proprties that don't change when debug code is added to find the bug. Every time a particular line of code is executed with particular values of the input values, the system misbehaves" –Hobbs'16

Heisenbug

"A Heisenbug, in contrast, is a will-of-the-wisp that appears and disappears in a manner that makes it extremely elusive." –Hobbs'16

- What is **probability of failure** in system controller?
- Why **harder to estimate** than for hardware?
- Software does not **wear out** like hardware!

Software vs Hardware

"As a rule of thumb, we suggest that if a device has **few enough internal stored states** *that it is practical to cove them all in testing, it may be better to* **regard it as hardware** *and to show that it meets its safety requirements by analysis of the design and testing of the completed device, including* **exhaustive testing** *of all input and state combinations"*

–Engineering Safety Management, Rail Safety and Standards Board

Interesting Reading

- **What Delays Trains?** (MetLink)

  `https://www.metlink.org.nz/getting-around/what-delays-trains/`

- **Points failure on Wellington-Petone Line.** (Stuff)

  `https://i.stuff.co.nz/dominion-post/news/92535701/points-fault-causes-disruption-to-wellington-train-services`

- **Door "Interlock" on FP/FT Units.** (wikipedia)

  `https://en.m.wikipedia.org/wiki/New_Zealand_FP_class_electric_multiple_unit`