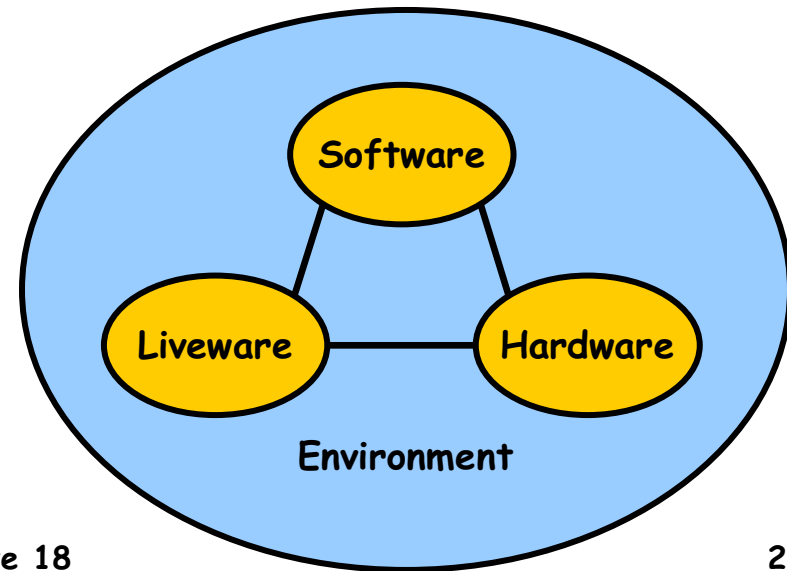


# Engineering High-Dependability Systems

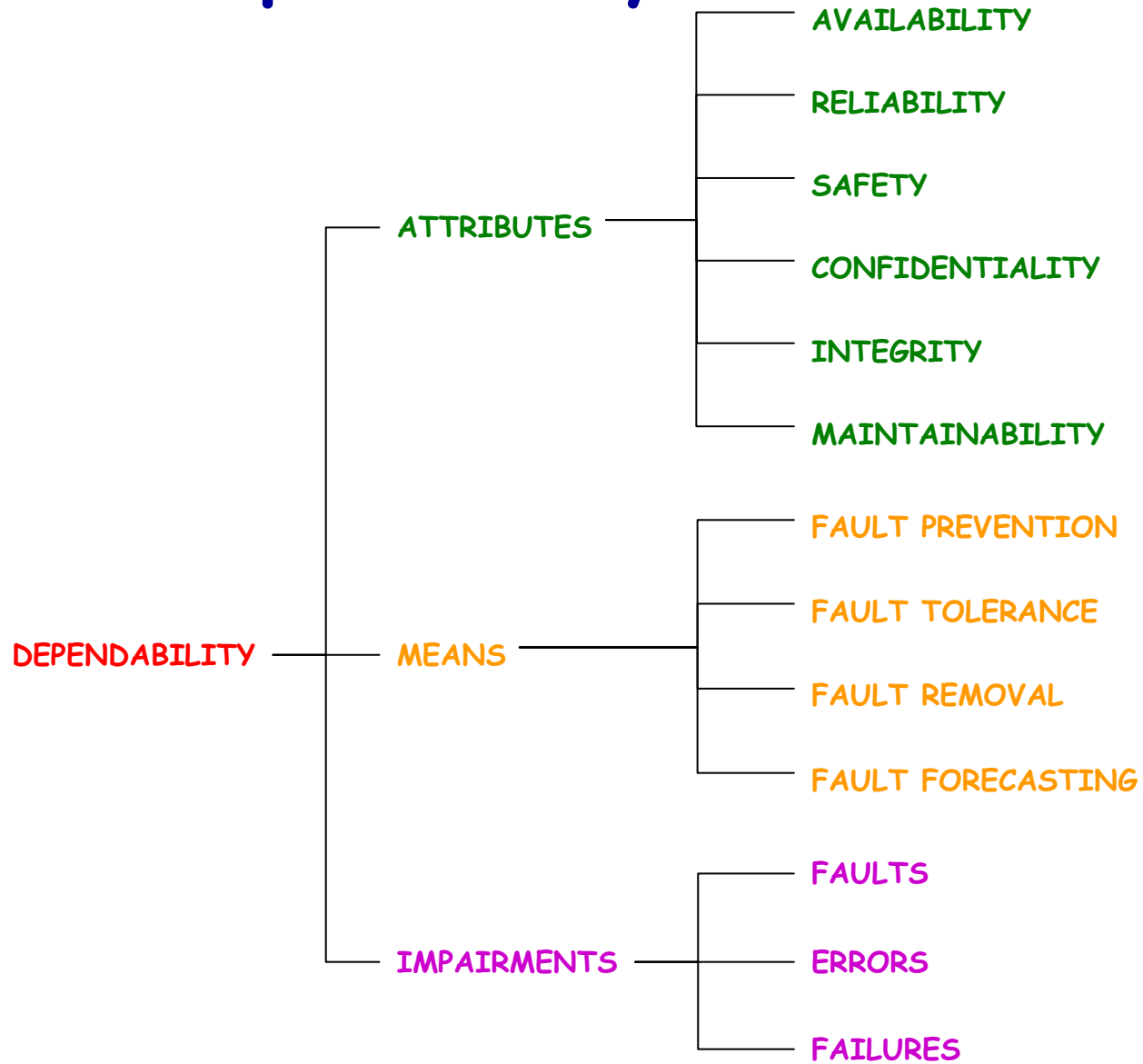
Massimo Felici  
Room 1402, JCMB, KB  
0131 650 5899  
[mfelici@inf.ed.ac.uk](mailto:mfelici@inf.ed.ac.uk)

# Dependability of Socio-technical Systems

- What is Dependability?  
**Dependability** is that property of a computer system such that reliance can justifiably be placed on the service it delivers.
- The **service** delivered by a system is its behavior as perceived by its user(s).
- A **user** is another system (human or physical) interacting with the system considered.
- What are **Socio-technical Systems**? Systems involving humans, organizations, environment,...
- Here is a holistic view of Socio-technical Systems
- Software is a (significant) part of the "whole system"



# What is Dependability?



# (Some) Flavours of Dependable Systems

- **Safety-critical:** failure leads to serious injury, loss of life, or significant environmental damage
- **Security-critical:** access control, permissions and monitoring (potentially in the face of malicious attack) a key issue
- **Fault-tolerant:** system is robust - can withstand errors in, or failures of, parts of the system (e.g., auto-pilots)
- **High-reliability:** likelihood of failure-on-demand exceptional low (e.g., fire-safety shutdown systems)

# What is undependability?

- “Classic” high profile failures:
  - Mars Climate Orbiter
  - Ariane 5
  - Therac 25
  - ...
- What else?
  - **pervasiveness** of computers (e.g., Y2K)
  - Multiple low-criticality failures
  - Dependence of society
  - Service loss: “the system is down”
- Impact on **organizations**
  - NATS, NHS, finance,...

# NASA' Mars Climate Orbiter

- Part of Mars Surveyor programme (1993)
- Developed at cost of \$327.6 million (orbiter and lander)
- Launched December 1998
- Intended to enter Mars orbit September 1999. at 210km altitude
- September 23<sup>rd</sup>, attempted orbit at 57km, burned up in Martian atmosphere
- Investigation - Phase 1 Mishap Investigation report, November 1999
  - Root cause: failure to use metric units in ground software file "Small Forces"
  - Team developing SM\_FORCES used English units of pounds-seconds
  - Team developing navigation software algorithm assumed metric units of Newton-seconds.
  - Project SIS (Software Interface Specification) not followed
- Contribution causes
  - Process did not adequately address transition from development to operations
  - Inadequate communication between teams
  - V&V process did not adequately address ground software

# Therac 25 Radiotherapy Machine

- Therac-25 had two operating modes:
  - Low intensity (electron radiation), wide spread
  - High intensity (X-ray radiation), tight focus
- Software fault in data entry permitted high intensity, wide spread
  - X-ray generated by placing tungsten shield as "filter" for high-intensity electron beam
  - Set-up process takes considerable time
  - Changes during set-up not validated
- 6 known accidents between June 1985 and January 1987, leading to 2 confirmed deaths
- Hardware interlocks in Therac-20 removed (software error present, but caused blown fuse)

# (Some) Other Major Software Failures

- London Ambulance Service
- Taurus Financial System
- CUFS (Cambridge University Financial System)
- Swanwick ATC? Proposed 1988 (for 1996), building commenced 1991, completed 1994, software working "by winter 2002"?



# Safety Critical Systems

- Variety of industrial sectors; both regulated and (relatively) unregulated
  - Regulated
    - Hazardous manufacturing (e.g., chemical, explosives, etc.)
    - Travel and transport (e.g., air, rail, sea)
    - Energy (e.g., nuclear, petrochemical)
  - (less) regulated
    - Automotive (e.g., engine controllers, ABS, etc.)
    - Medical informatics (e.g., radiotherapy, anaesthetics, medical expert systems, etc.)
- Focus on design for assessment
- Motivation/drivers for **safety culture**
- **whole system** issues
- Software not necessarily susceptible to traditional engineering approaches

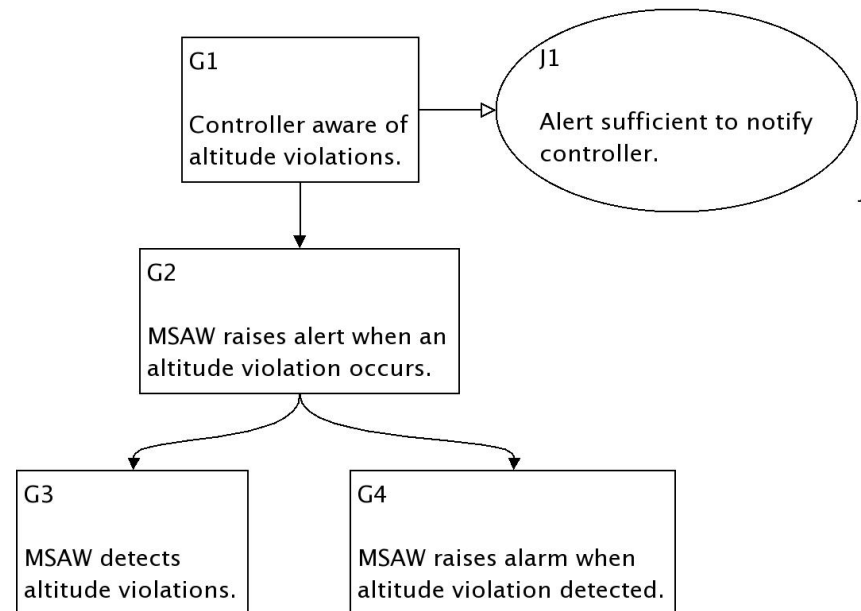
# Regulation and Assessment

- Regulatory standards:
  - National and international
  - Generic and domain-specific
  - Independent assessment and regulatory authorities

- Safety** **cases:**  
**arguments** of acceptable safety of proposed system

- Pre-1990's: largely prescriptive
- 1988: Piper-alpha; Cullen inquiry highly critical of "box ticking"
- Post-Cullen: move to goal-setting standards

## Structured Safety Cases



## Goal Structuring Notation (GSN)

# A Safety Case

- A **safety case** is a document, or collection of documents, that presents the arguments for believing that a proposed potentially dangerous system is **acceptably safe**
- It sets out the **risks** involved with the operation of the process or equipment and the possible consequences of failure, and specifies what will be done (or has been done) to **minimise the probability** and the impact of these **failures**

# Motivation and Drivers for Safety

- Economic - **cost benefit analysis** (one life ~ £1-2 million)
- **Responsibility**
  - Developer versus assessor versus regulator
  - In-house versus 3<sup>rd</sup> party (e.g., COTS/SOUP)
- **Liability**, e.g., British Rail:
  - Pre-privatisation: HSE, rail regulatory authority
  - Post-privatisation: HSE, rail regulatory authority, Railtrack, 3<sup>rd</sup> party maintenance, strategic rail regulator, rail safety assessor,...
- **History**
  - Design for significant accidents
  - Safety culture "disaster-driven"
  - No significant automotive/medical disasters...?

# Software Engineering for Safety Critical Systems

- No “new” software engineering techniques
- Adoption of traditional, physical engineering techniques
  - For design (e.g., triple modular redundancy, fault tolerance, failsafe, error recovery, etc.)
  - For analysis (e.g., hazard analysis, fault tree analysis, failure modes, effects analysis, etc.)
- ...but software unlike physical systems
  - Often **open** and **evolving**
  - High functional **complexity**
  - Common mode **failures**
  - Complex **dependencies**
  - Software **errors** are all latent



# Design for assessment: Arguments of Software Safety

## ■ Process versus product

- Design process used will lead to safe system versus system produced is shown to be safe

## ■ Quality versus quantitative

- Claims of good quality (e.g., unsafe states not possible) versus numerical measures of, for example, probability of error

## ■ Formal verification

- Use of formal methods. However, formal models often constructed but not verified

## ■ Adoption of traditional engineering analyses

- HAZOPS (HAZard and Operability Studies), FMEA (Failure Modes and Effects Analysis), FTA (Fault Tree Analysis)

## ■ Software specified standards

- E.g., MOD 00-55, IEC 61508

## ■ Responsibility

- Open issue - responsibility versus liability



# Safety Standards for Software

- **MOD Def.Stan. 00-55** - Procurement of Safety Critical Software in Defense Equipment
  - Requirements:
    - Safety management
    - Software engineering practice
  - Guidance:
    - Lifecycles
    - Hazard analysis
    - Risk analysis
    - V&V
    - Independent auditing
    - ...



# Safety Standards for Software

- **IEC 61508** - Functional Safety of electrical/electronic/programmable electronic safety related systems
  - Safety at the system level
    - The overall safety lifecycle
    - Management of functional safety
    - Hazard analysis, risk reduction and safety integrity levels
  - Hardware and architectural safety
    - Managing hardware
    - Failure probabilities
    - Integrating hardware into the safety lifecycle
  - Software safety
    - Safety related software
    - Selecting development methods and tools
    - Verification and validation



# Computer-related accidental death

An empirical exploration (D. MacKenzie, 1994)

- **Computer-related** - as before
- **Accidental death** - non-deliberate (i.e., military); empirically "easy" to measure
- **Causes**
  - 4% physical (chiefly electromagnetic interference)
  - 3% software error
    - Therac-25 (2) + Patriot (28)
  - 92% failure in human-computer interaction
  - Estimated number of deaths (until end 1992): 1,100 +/- 1,000

# Fatalities due to Software Error

- 1986, Therac-25
  - Radiotherapy machine
  - Two operating modes:
    - Low intensity, wide spread
    - High intensity, tight focus
  - Software error in data entry permitted high intensity, wide spread
  - Hardware interlock in Therac-20 removed
  - Overdosing leads to 2 confirmed deaths
- 1991, Patriot anti-Scud missile system
  - Internal clock uses tenths of seconds
  - Binary rounding error known
  - Long run times not anticipated
  - 25<sup>th</sup> February: Alpha Battery in uninterrupted operation for over 100 hours; unable to track Scud missiles; 28 US servicemen killed
  - 26<sup>th</sup> February: software fix arrives



## Fatalities due to Human-computer Interaction Failures

- 1992, A320 airbus crashes after over-rapid descent, 87 die. "Glass cockpit" descent display of 3.3 (thousands of feet per minute) wrongly interpreted as angle - Airbus denies responsibility (although interface changed subsequently).
- Similar incidents
  - Habsheim 1988, Bangalore 1990, Moscow 1991, Nagoya 1994, Toulouse 1994, Paris 1994,....
- 1998, USS Vincennes shoots down Iran Air airliner, all 290 on board die: weapon system human interface deemed "not optimal"

# Lufthansa Airbus A320 Crash, Warsaw

- 14 Sept. 1993, flight DLH 2904 from Frankfurt overruns runway on landing at Warsaw Airport, Poland
  - co-pilot and 1 passenger die, 54 people hospitalised
- Autopilot found to have prevented reverse thrust braking for 9 seconds
- Specification of checks to sever (height, wheel-spin, weight on both wheels)
- Dangers of in-flight reverse thrust activation: May 1991, Boeing 767 brought down over Thailand, killing all on board
- Pilot error/specification error? Airbus denies responsibility (although code subsequently changed)

## Fatalities due to Human-computer Interaction Failures

- 1992, London Ambulance Service Computer Aided Despatch Systems
  - primarily management failure, claims of 20-30 deaths
- 1982-1991, North Staffs. Royal Infirmary radiotherapy
  - Double error-correction leads to underdosing of around 1,000 patients, 401 die
  - Clinical verdict is "tens rather than hundreds" due to double error-correction
- 1994, Toulouse: Airbus A330 stalls during testing. Flight level set at 2,000ft instead of 7,000ft
  - 7 dead including Airbus chief test pilot
  - Attitudes to "safety envelope" different from Airbus and Boeing

# Where is the danger?

- Software does not directly cause fatalities
- **Interaction** between software and physical world
- All **software errors latent**; present in specification
  - Requirements missed, incorrect or misunderstood
  - Failure to correctly implement specification rarer, e.g., NASA Galileo/Voyager mission critical software



# How Dangerous is Software?

- Estimated number of deaths (until end 1992): 1,100 +/- 1,000
- Compare with UK road deaths for 1992 alone: 4,274
- Human **perception of risk** and danger complicated
  - Degree of personal control
  - Anticipated benefit
  - Availability of data
- **Hypothesis**: software is safe, because we believe it to be dangerous

## Where does SEOC fit in?

### Software Engineering versus Physical Engineering

- **Art, Craft, Science???**
  - **Science**: strong theoretical foundation, knowledge-based teaching
  - **Craft**: little theory, skill-based apprentices
  - **Art**: highly subjective, innate ability crucial
- Errors are **latent** (often in requirements or specification)
- Physical system convex, e.g., beam is tested for 100kg load and 1000kg load, can assume will carry any load between 100-1000kg
- Software not convex, but structured
  - Can partition system; e.g., independence of failures in interface and operation (c.f., Therac-25)



# SEOC and Dependability Process, Assessment and Components

## ■ Process oriented assessment

- E.g., lifecycle models, project planning and management, structured test plans,...
- Combining qualitative and quantitative modelling and assessment

## ■ Avoiding failure in communication

- During design (c.f., Mars Climate Orbiter)
- During assessment (c.f., Voyager and Galileo; Ariane 5)
- Validation as well as verification (c.f., LASCAD)

## ■ (reuse of) trusted/tested components

- E.g., 3<sup>rd</sup> party developers of high-dependability components

## ■ Principled composition of components

- Theory of composition, e.g., independence of failures in component

