Embedded Systems Engineering

System Reliability - 1

- Critical Systems & Reliability
- Measures of Reliability
- Faults, Failures & Effects
- System Specification and Formal Methods
- Number Representations and their problems

Michael Brockway

Critical Systems & Reliability

- Examples

	Consequences of failure
Military aircraft fly-by-wire	Loss of control within 0.5 s
Aero engine advanced variable control systems	Engine blow-up
Aircraft auto-land system	If < 30 m, catastrophe
Airbag deployment system	Inconvenience – death
Air-sea rescure maritime reconnaissance aircraft – loss of search radar	Cannot complete search mission
Telecom switches	loss of service
On-line transaction processing	loss of money

Critical Systems and Reliability

- Read the following articles on the web -
  - The Challenger accident:

    `http://www.fas.org/spp/51L.html`
  - The Pentium Division Bug:

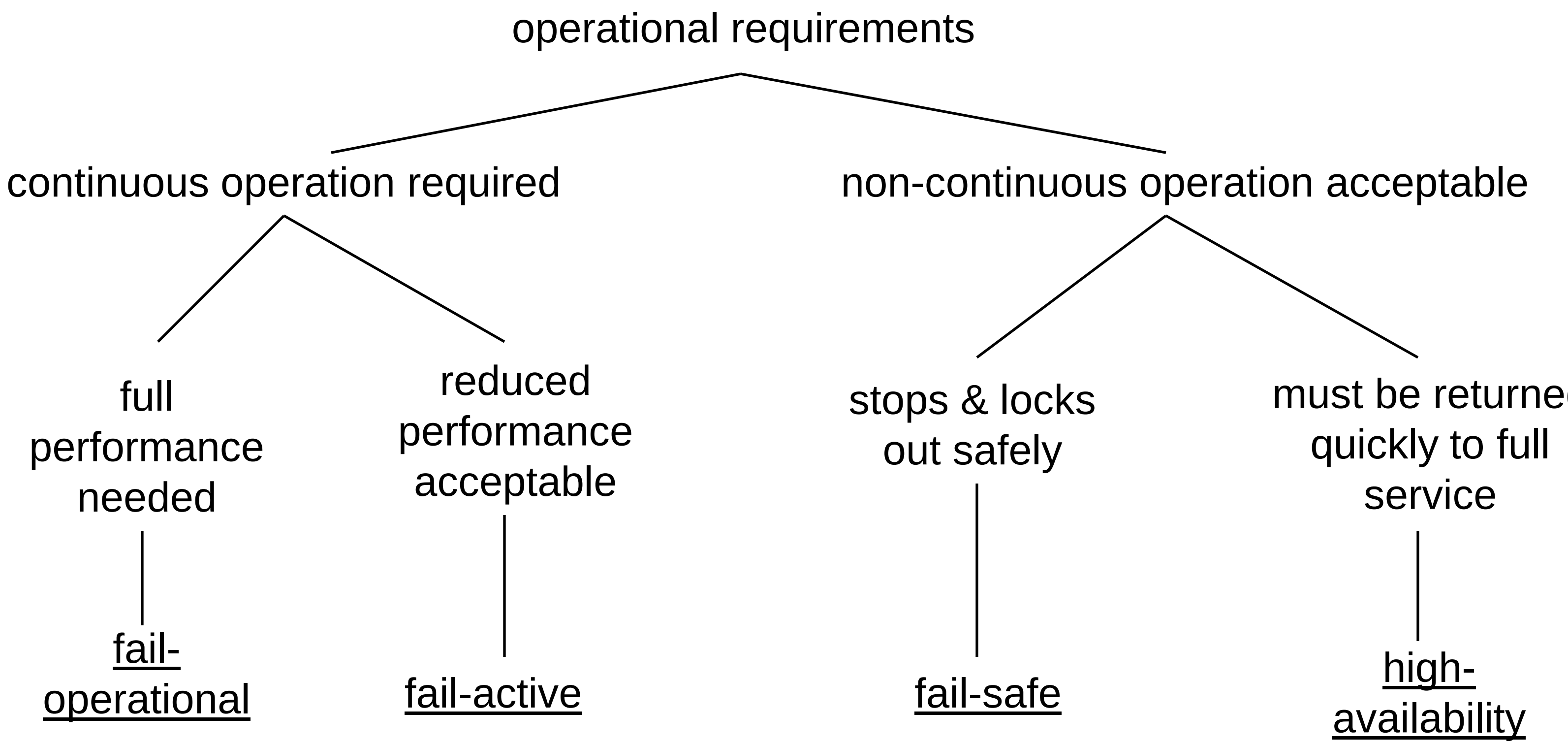
    `http://www.maa.org/mathland/mathland_5_12.html`
  - The Mars Pathfinder:

    `http://catless.ncl.ac.uk/Risks/19.49.html#subj1`
  - The Therac-25 Accidents

    `http://courses.cs.vt.edu/~cs3604/lib/Therac_25/Therac_1.html`

Critical Systems & Reliability

- Safety-critical vs mission-critical
- Four broad groupings of fault-tolerant systems



Critical Systems & Reliability

- FO
  - Full performance in presence of faults – no external visible sign of a fault
- FA
  - Continuous but reduced performance
  - "Graceful degradation"
- FS
  - System ceases to work but goes into a safe mode
- HA
  - System may cease to work but must be returned to normal service very quickly
  - Faulty units may be replaced while system is on-line

Measures of Reliability

- MTBF as a measure of reliability
  - Mean time between failures vs failure rate
  - Eg MTBF = 1000 hours ⇔ failure rate = 0.001 per hour

- For HA systems
  - MTTR = mean time to repair
  - Availability = MTBF / (MTBF + MTTR)
- Some Rules of thumb:

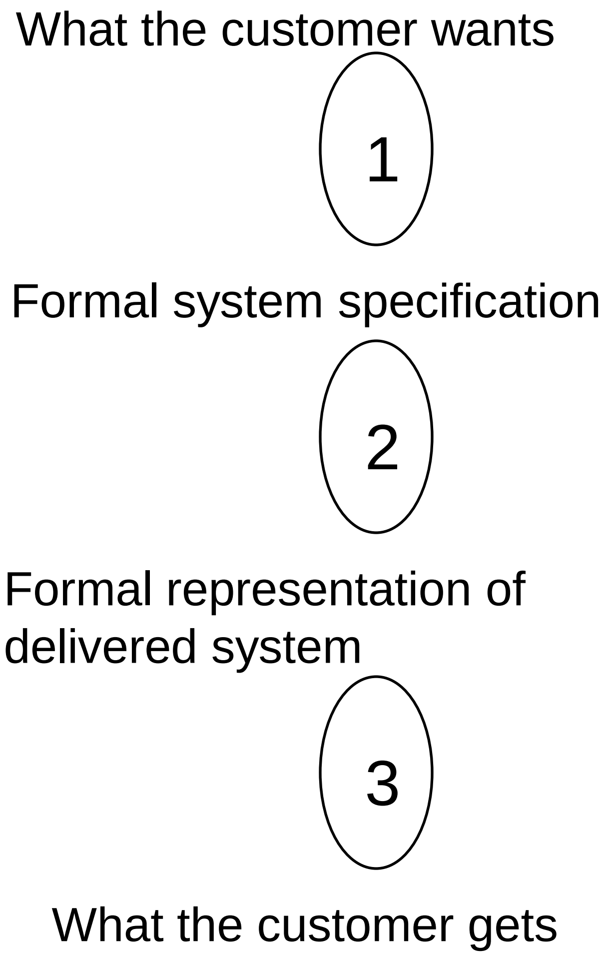
Severity level:	Minor		Significant	Critical	Catastrophic
Probability of failure:	reasonably probable	unlikely	rare	extremely rare	extremely improbable
acceptable failure rates:	..........10^{-3}	..........10^{-5}	...........10^{-7}	.............10^{-9}	....
Application area	mission critical			safety critical	

Faults, Failures and Effects

Fault	Failure	Effect
the basic cause of the problem	why the system failed	what happened in consequence
Software could not handle number range	numeric overflow	loss of control of space vehicle
wrong value input	wrong number computed	space vehicle crashed into planet

- 3 layers of "defence" against faults
  1. *fault prevention* techniques (stop faults arising in first place)
  2. *software fault tolerance* (detect failures and take action to correct)
  3. *hardware fault tolerance*

System Specification - The Need for Formal Methods



- Three “reality gaps”
- Bridging gap 1 is a matter of analysis, requirements engineering: a mixture of informal and formal methods.
- Bridging gap 3 is a matter of testing and review- a mixture of informal and formal methods.
- Formal methods are not a complete solution for these.
- But they are a *requirement* for being able to express a specification in sufficiently precise terms.

The Need for Formal Methods

- Gap 2 can be addressed effectively by *formal methods*
  - In design, development phases, use formal modeling methods
  - We limit the scope of a formal methods -- we apply it only to “critical” components and properties, eg Safety and Liveness properties of concurrent or real-time systems.
- Provided
  - the specification is given in a sufficiently rigorous language &
  - the delivered system is modeled faithfully in the same (or a related) formal language, then
  - gap 2 can be checked *formally*, in principle *rigorously*, and in some cases, *automatically* – The specification is expressed in a formal language and this is input to a (software) tool which checks all logically possible behaviours of the specified system against a list of “undesirable” behaviours.

Formal Methods

- In high-integrity embedded systems we are concerned specifically with things like -
  - Safety issues
    - Nothing bad (interference, deadlock, lost data transactions, ...) will ever happen...
  - Liveness issues
    - Some good will eventually happen -- the process will make progress and eventually deliver required results.
- In real-time systems we are concerned with behaviour (including these issues) within numerically specified time constraints.
- We can use formal modeling/specification languages and tools specifically tailored to these.

Formal Methods -- Languages, Tools

- Equivalence, Entailment
  - We use logic to establish that certain requirements or assertions are *equivalent* or that one *entails* another
  - Logical methods and tools are capable of rigorously, mathematically proving that a formal model M ***satisfies*** a given specification S
    - All propositions (statements, assertions) that are part of S are true in (any possible run or execution of the system modeled by) M

Formal Methods -- Languages, Tools

- Process algebras - eg
  - CSP (see book by Mett, Crowe, Strain-Clark),
  - CCS (Milner),
  - FSP (see technical references 3)
- Logical Methods -
  - Set theory

$x \in A \cap B \leftrightarrow x \in A \; \& \; x \in B$

$A = \{2, 3, 5, 7, 11\}$

  - Z
  - Predicate Calculus,

$\forall x (Px \rightarrow Qx) \; \& \; (\exists x \; Px) \rightarrow (\exists x \; Qx)$

  - Temporal Logics,
  - Automata

Languages, Tools

- For Specification, make assertions, express requirements in some kinds of logic
  - Linear Temporal logic (LTL), CTL, ...
  - Timed LTL

- For System Modeling we can use
  - Labeled Transition systems -
    - Finite State Automata
    - Büchi automata
    - Timed Automata,
  - System specification languages
    - Promela, ...

Why do this?

- Once we have, in formal terms,
  - a system specification
  - a description (model) of the system we have built
- ...we can use *automatic* tools to
  - simulate a run of the system, either
    - at random -- useful in early stage of building to test a "first cut", or
    - a guided simulation, eg Perhaps we have found a bug: in order to investigate it, we would like to reproduce the situation that cause the bad behaviour.
  - generate *Message Sequence Charts*
    - Trace of all messages, interactions between components of concurrent system.
  - generate a profile of execution
    - Trace of what actions occurred, in what order, among several concurrent processes.
  - monitor values of symbols -- track changes in variable values
  - check validity of assertions
  - check logical *state space* of system

Number Representations and their problems

- Making measurements
  - type
    - discrete, or
    - continuous
  - what the result bits represent
    - range
    - resolution (accuracy) versus
    - precision (number of significant figures in display)
      - truncation v rounding

Number Representations and their problems

- Number representation
  - Fixed-point positional systems – written form

    $D_3\ D_2\ D_1\ D_0 . D_{-1}\ D_{-2}$

  - actually means

    $D_3*r^3 + D_2*r^2 + D_1*r^1 + D_0*r^0 + D_{-1}*r^{-1} + D_{-2}*r^{-2}$

    - `r` is the *radix* or *base*: eg 10 (denary or decimal), 16 (hexadecimal), 8 (octal), 2 (binary)
    - Have as many digit terms as you like to the left, to the right
    - Eg (decimal)

$3051.68 = 3*10^3 + 0*10^2 + 5*10^1 + 1*10^0 + 6*10^{-1} + 8*10^{-2}$

    - Eg (binary)

$1011.01 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2}$

Number Representations and their problems

- Number representation
  - Floating-point positional systems – the number is quoted in the form

    `D.DDD * r`$^{\pm EE}$ `= (mantissa) * radix`$^{(exponent)}$

- In real systems, the number of digits allowed for a number is limited
  - waste of computation time, memory producing more precision than the accuracy of the data warrants
  - We may trade precision off against speed
  - In floating-point working, we trade mantissa digits off against exponent digits
    - precision against range

Number Representations and their problems

- Standard Fixed point representations
  - `unsigned char` (in C) is a fixed-point binary number of 8 bits
    - range 0 – 255, resolution 1
  - `char` is a fixed-point binary number of 8 bits with 2's complement
    - range -128 – 127, resolution 1
  - Can achieve different ranges by *scaling* but the resolution is also scaled
  - Similarly
    - `unsigned short` – 16 bits: range 0 – 65535 (= 2^{16} – 1)
    - `short` – 16 bits: range –32768 (= -2^{15}) – 32767 (= 2^{15} – 1)
    - `unsigned long` – 32 bits: range 0 – 4294967295 (= 2^{32} – 1)
    - `long` – 32 bits: range -2147483648 – 2147483647

Number Representations and their problems

- Fixed Point Calculations
  - Addition, subtraction done with hardware adder, 2’s-complementer
  - Multiplication, division may involve bit shifts
- Problems
  - Overflow
    - Can be detected and handled as an exception; but ...
  - Potential loss of information
    - Eg in dividing: bits are shifted to right: shifted bits are lost

Number Representations and their problems

- Standard Floating-point formats
  - An IEEE 754 single-precision number (`float` in C) is a floating-point binary number of 32 bits made up of
    - Mantissa: 1 sign bit (ie 2's complement) + 23 bits
    - Exponent: 8 bits, representing values in range –128 -- +127
    - The maximum positive number that can be represented is

    1.1....1[23 bits after pt] $* \ 2^{127}$ which is slightly less than $2^{128} \approx 3.4 * 10^{38}$.
    - The minimum positive number that can be represented is

    1.0....0[23 0 bits after pt] $* \ 2^{-127} \approx 1.7 * 10^{-38}$.
    - The range of negative numbers is the mirror image of this.
    - The precision with which a value can be recorded is the value of the least significant bit in its mantissa * 2 to the power of its exponent.

Number Representations and their problems

- Standard Floating-point formats
  - An IEEE 754 double-precision number (`double` in C) is a floating-point binary number of 64 bits made up of
    - Mantissa: 1 sign bit (ie 2's complement) + 52 bits
    - Exponent: 11 bits, representing values in range –1024 -- +1023
    - The maximum positive number that can be represented is

    $1.1....1$[52 1-bits after pt] $* \ 2^{1023}$ which is slightly less than 2^{1024} $\approx 1.7 * 10^{308}$.
    - The minimum positive number that can be represented is

    $1.0....0$[52 0-bits after pt] $* \ 2^{-1022} \approx 2.2 * 10^{-308}$.
    - The range of negative numbers is the mirror image of this.
    - The precision with which a value can be recorded is the value of the least significant bit in its mantissa * 2 to the power of its exponent.
      - The maximum error is half this in case of rounding rather than truncating.

Number Representations and their problems

- Floating point calculations
  - Multiplication and division are straightforward to define -
    - $(m * r^{e}) * (m' * r^{e'}) = (m * m') * r^{e+e'}$
    - $(m * r^{e}) / (m' * r^{e'}) = (m / m') * r^{e-e'}$

  - Addition, subtraction are more complicated.
  - For example pretend for the moment our floating point format has just an 11-bit mantissa. To add $1.11010001011 * 2^{12}$ and $1.00011001110 * 2^{10}$ there are three steps:

1. re-align the smaller number so that it has the same exponent as the larger: $1.00011001110 * 2^{10} \rightarrow 0.01000110011 * 2^{12}$
2. Add the mantissae: $1.11010001011 + 0.01000110011 = 10.00010111110$
3. re-normalise if necessary: $10.00010111110 * 2^{12} \rightarrow 1.00001011111 * 2^{13}$

Number Representations and their problems

- Floating point calculations
  - Subtraction is similar but the second mantissa is 2's-complemented first.
- Problems with Floating point representations
  - Loss of information
    - Did you notice that in re-aligning one of the numbers above,

[`1.00011001110 * 2`^{10} `-> 0.01000110011 * 2`^{12}] we lost a 1-bit? Bit-shifts tend to lose data.

    - This can also happen in the multiplication or division of mantissae when we multiply/divide two floating-point numbers.
  - Overflow
    - The result can be bigger than the largest representable positive (or smaller than the smallest negative) value

Number Representations and their problems

- Problems (ctd)
  - Underflow
    - The result can be smaller than the smallest representable positive (or bigger than the largest negative) value.
  - The order of operations can matter
    - ... violating the commutative laws [x + y = y + x etc]or associative laws [x+(y+z) = (x+y)+z]
    - eg `1.00000000001 * 2`^{10} `– 1.00000000000 * 2`^{10} + `1.10000000000 * 2`$^{-4}$
    - This initial subtraction yield `1.00000000000 * 2`$^{-1}$ on re-normalisation; then adding `1.10000000000 * 2`$^{-4}$ gives `1.00011000000 * 2`$^{-1}$.
    - But adding `1.00000000001 * 2`^{10} and `1.10000000000 * 2`$^{-4}$ first yields `1.00000000001 * 2`^{10} again: re-aligning `1.10000000000 * 2`$^{-4}$ loses ALL its bits. Then subtracting `1.00000000000 * 2`^{10} yields `1.00000000000 * 2`$^{-1}$ – the wrong answer.