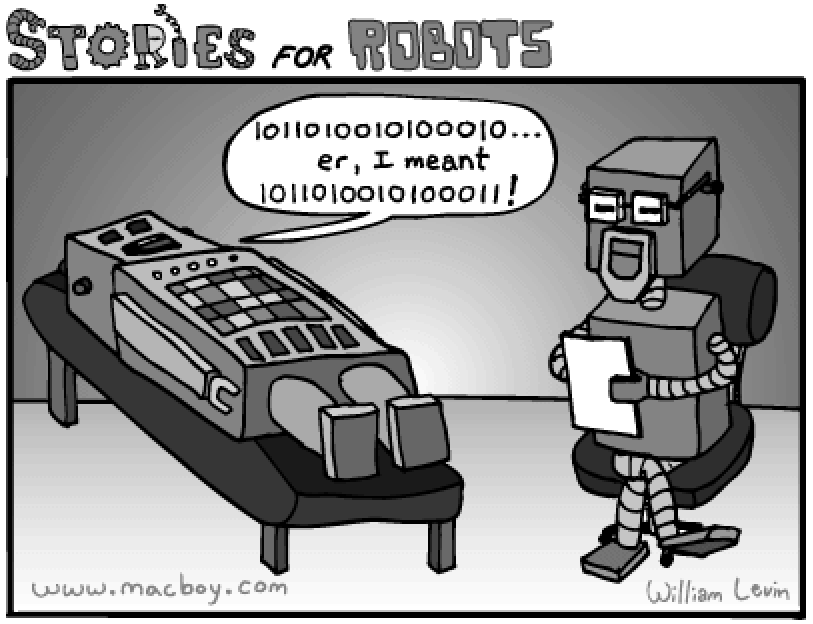
Fault-Tolerant Computing

Motivation, Background, and Tools

About This Presentation

Edition	Released	Revised	Revised
First	Sep. 2006		

Design, Implementation, Operation, and Human Mishaps



Droidian Slip

CONGRATULATIONS! YOU HAVE JUST DISCOVERED A COMPLETELY NEW ERROR!
GOFF

The Curse of Complexity

Computer engineering is the art and science of translating user requirements we do not fully understand; into hardware and software we cannot precisely analyze; to operate in environments we cannot accurately predict; all in such a way that the society at large is given no reason to suspect the extent of our ignorance.[1]

Microsoft Windows NT (1992): ≈4M lines of code
Microsoft Windows XP (2002): ≈40M lines of code

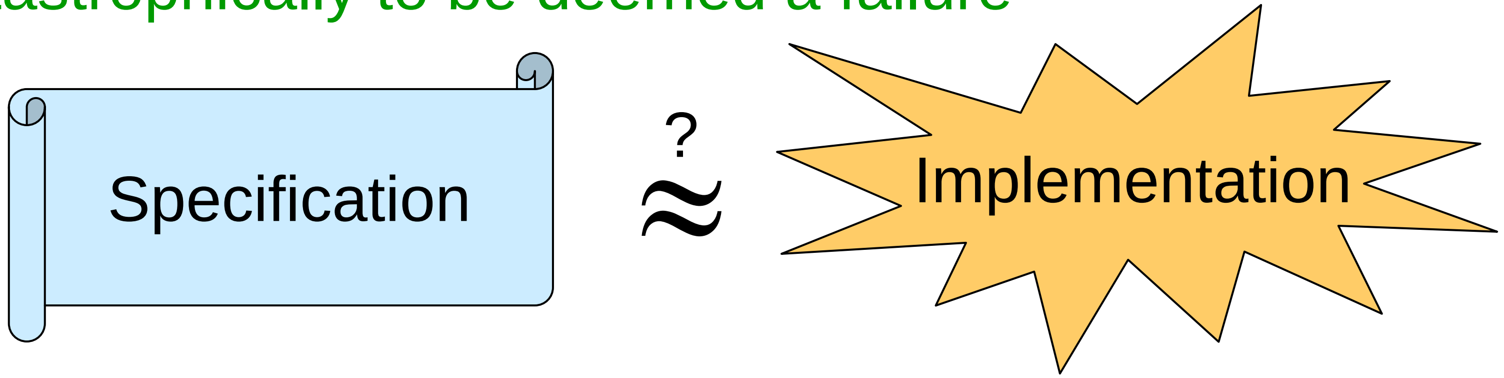
Intel Pentium processor (1993): ≈4M transistors
Intel Pentium 4 processor (2001): ≈40M transistors
Intel Itanium 2 processor (2002): ≈500M transistors

[1]Adapted from definition of structural engineering: Ralph Kaplan, *By Design: Why There Are No Locks on the Bathroom Doors in the Hotel Louis XIV and Other Object Lessons*, Fairchild Books, 2004, p. 229

Defining Failure

Failure is an unacceptable difference between expected and observed performance.[1]

A structure (building or bridge) need not collapse catastrophically to be deemed a failure



Reasons of typical Web site failures

Hardware problems:	15%
Software problems:	34%
Operator error:	51%

[1] Definition used by the Tech. Council on Forensic Engineering of the Amer. Society of Civil Engineers

Design Flaws: "To Engineer is Human"[1]

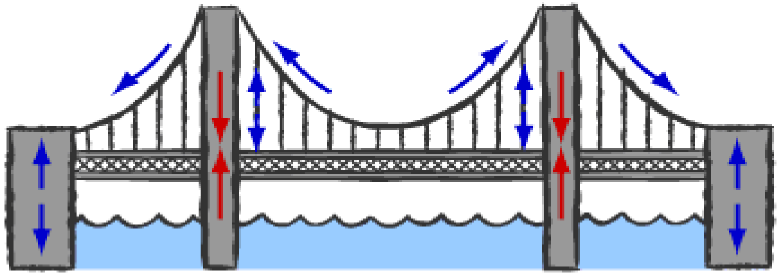
Complex systems almost certainly contain multiple design flaws

Redundancy in the form of safety factor is routinely used in buildings and bridges

One catastrophic bridge collapse every 30 years or so

See the following amazing video clip (Tacoma Narrows Bridge): http://www.enm.bris.ac.uk/research/nonlinear/tacoma/tacnarr.mpg

Example of a more subtle flaw:
Disney Concert Hall in Los Angeles reflected light into nearby building, causing discomfort for tenants due to blinding light and high temperature



[1] Title of book by Henry Petroski

Design Flaws in Computer Systems

Hardware example: Intel Pentium processor, 1994

For certain operands, the FDIV instruction yielded a wrong quotient
Amply documented and reasons well-known (overzealous optimization)

Software example: Patriot missile guidance, 1991

Missed intercepting a scud missile in 1st Gulf War, causing 28 deaths
Clock reading multiplied by 24-bit representation of 1/10 s (unit of time) caused an error of about 0.0001%; normally, this would cancel out in relative time calculations, but owing to ad hoc updates to some (not all) calls to a routine, calculated time was off by 0.34 s (over ≈100 hours), during which time a scud missile travels more than ½ km

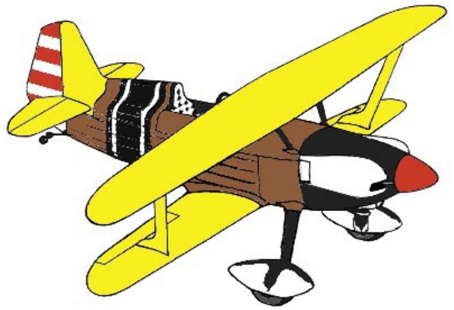
User interface example: Therac 25 machine, mid 1980s[1]

Serious burns and some deaths due to overdose in radiation therapy
Operator entered “x” (for x-ray), realized error, corrected by entering “e” (for low-power electron beam) before activating the machine; activation was so quick that software had not yet processed the override

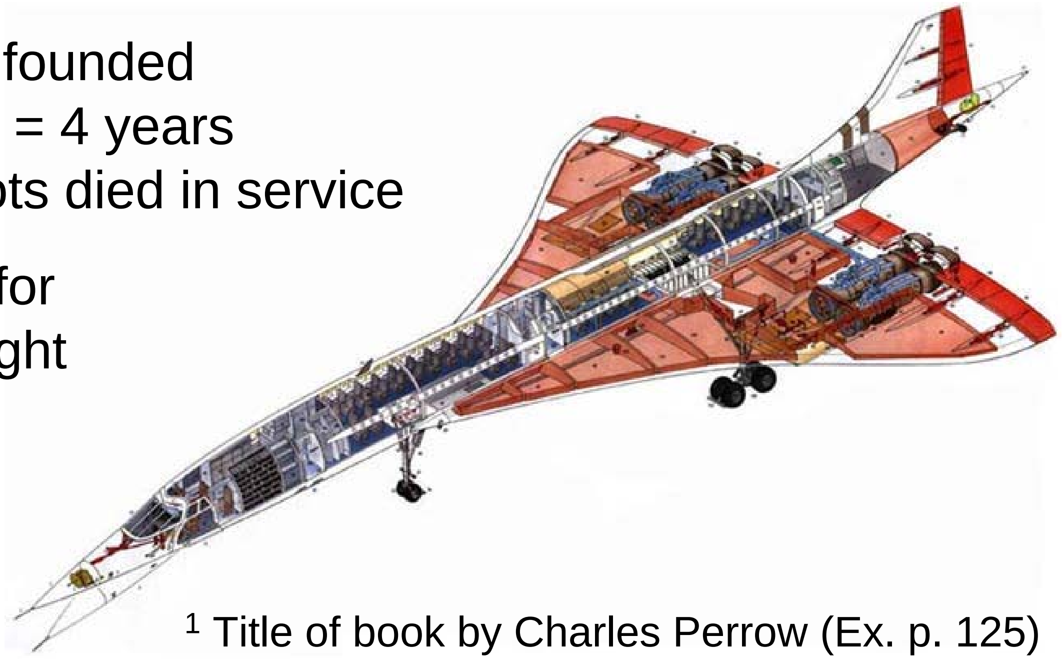
[1] Accounts of the reasons vary

Learning Curve: "Normal Accidents"[1]

Example: Risk of piloting a plane



1903	First powered flight
1908	First fatal accident
1910	Fatalities = 32 (≈2000 pilots worldwide)
1918	US Air Mail Service founded Pilot life expectancy = 4 years 31 of the first 40 pilots died in service
1922	One forced landing for every 20 hours of flight
Today	Commercial airline pilots pay normal life insurance rates



[1] Title of book by Charles Perrow (Ex. p. 125)

Mishaps, Accidents, and Catastrophes

Mishap: misfortune; unfortunate accident

Accident: unexpected (no-fault) happening causing loss or injury

Catastrophe: final, momentous event of drastic action; utter failure

At one time (following the initial years of highly unreliable hardware), computer mishaps were predominantly the results of human error

Now, most mishaps are due to complexity (unanticipated interactions)

THE RISKS DIGEST

Forum on Risks to the Public in Computers and Related Systems
http://catless.ncl.ac.uk/risks (Peter G. Neumann, moderator)

Example from THE RISKS DIGEST

On August 17, 2006, a class-two incident occurred at the Swedish atomic reactor Forsmark. A short-circuit in the electricity network caused a problem inside the reactor and it needed to be shut down immediately, using emergency backup electricity.

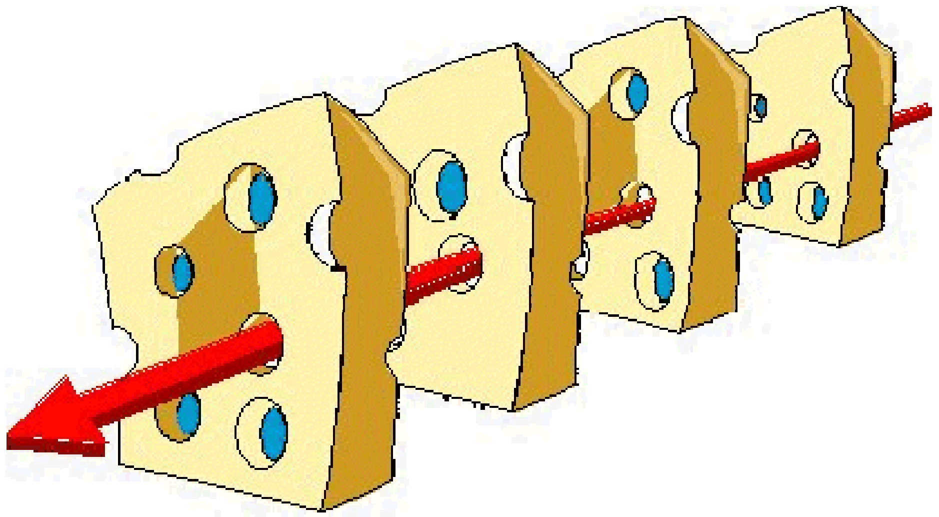
However, in two of the four generators, which run on AC, the AC/DC converters died. The generators disconnected, leaving the reactor in an unsafe state and the operators unaware of the current state of the system for approximately 20 minutes.

A meltdown, such as the one in Tschernobyl, could have occurred.

Coincidence of problems in multiple protection levels seems to be a recurring theme in many modern-day mishaps -- emergency systems had not been tested with the grid electricity being off

Layers of Safeguards

With multiple layers of safeguards, a system failure occurs only if warning symptoms and compensating actions are missed at each layer, which is quite unlikely



Is it really?

The computer engineering literature is full of examples of mishaps when two or more layers of protection failed at the same time

Multiple layers increase the reliability significantly only if the “holes” in the representation above are fairly randomly distributed, so that the probability of their being aligned is negligible

Dec. 1986: ARPANET had 7 dedicated lines between NY and Boston; A backhoe accidentally cut all 7 (they went through the same conduit)

A Problem to Think About

In a passenger plane, the failure rate of the cabin pressurizing system is 10^{-5}/ hr (loss of cabin pressure occurs once per 10^{5} hours of flight)

Failure rate of the oxygen-mask deployment system is also 10^{-5}/ hr

Assuming failure independence, both systems fail at a rate of 10^{-10}/ hr

Fatality probability for a 10-hour flight is about $10^{-10} \times 10 = 10^{-9}$ (10^{-9} or less is generally deemed acceptable)

Probability of death in a car accident is $\approx$1/6000 per year ($>10^{-7}$/ hr)

Alternate reasoning

Probability of cabin pressure system failure in 10-hour flight is 10^{-4}
Probability of oxygen masks failing to deploy in 10-hour flight is 10^{-4}
Probability of both systems failing in 10-hour flight is 10^{-8}
Why is this result different from that of our earlier analysis (10^{-9})?
Which one is correct?

Causes of Human Errors in Computer Systems

1. Personal factors (35%): Lack of skill, lack of interest or motivation, fatigue, poor memory, age or disability

2. System design (20%): Insufficient time for reaction, tedium, lack of incentive for accuracy, inconsistent requirements or formats

3. Written instructions (10%): Hard to understand, incomplete or inaccurate, not up to date, poorly organized

4. Training (10%): Insufficient, not customized to needs, not up to date

5. Human-computer interface (10%): Poor display quality, fonts used, need to remember long codes, ergonomic factors

6. Accuracy requirements (10%): Too much expected of operator

7. Environment (5%): Lighting, temperature, humidity, noise

Because "the interface is the system" (according to a popular saying), items 2, 5, and 6 (40%) could be categorized under user interface

Properties of a Good User Interface

1. Simplicity: Easy to use, clean and unencumbered look

2. Design for error: Makes errors easy to prevent, detect, and reverse; asks for confirmation of critical actions

3. Visibility of system state: Lets user know what is happening inside the system from looking at the interface

4. Use of familiar language: Uses terms that are known to the user (there may be different classes of users, each with its own vocabulary)

5. Minimal reliance on human memory: Shows critical info on screen; uses selection from a set of options whenever possible

6. Frequent feedback: Messages indicate consequences of actions

7. Good error messages: Descriptive, rather than cryptic

8. Consistency: Similar/different actions produce similar/different results and are encoded with similar/different colors and shapes

Operational Errors in Computer Systems

Hardware examples

Permanent incapacitation due to shock, overheating, voltage spike
Intermittent failure due to overload, timing irregularities, crosstalk
Transient signal deviation due to alpha particles, external interference

Software examples

These can also be classified as design errors

Counter or buffer overflow
Out-of-range, unreasonable, or unanticipated input
Unsatisfied loop termination condition

Dec. 2004: “Comair runs a 15-year old scheduling software package from SBS International (www.sbsint.com). The software has a hard limit of 32,000 schedule changes per month. With all of the bad weather last week, Comair apparently hit this limit and then was unable to assign pilots to planes.”
It appears that they were using a 16-bit integer format to hold the count.

June 1996: Explosion of the Ariane 5 rocket 37 s into its maiden flight was due to a silly software error. For an excellent exposition of the cause, see:
http://www.comp.lancs.ac.uk/computing/users/dixa/teaching/CSC221/ariane.pdf)

About the Name of This Course

Fault-tolerant computing: a discipline that began in the late 1960s – 1st Fault-Tolerant Computing Symposium (FTCS) was held in 1971

In the early 1980s, the name "dependable computing" was proposed for the field, to account for the fact that tolerating faults is but one approach to ensuring reliable computation. The terms "fault tolerance" and "fault-tolerant" were so firmly established, however, that people started to use "dependable and fault-tolerant computing."

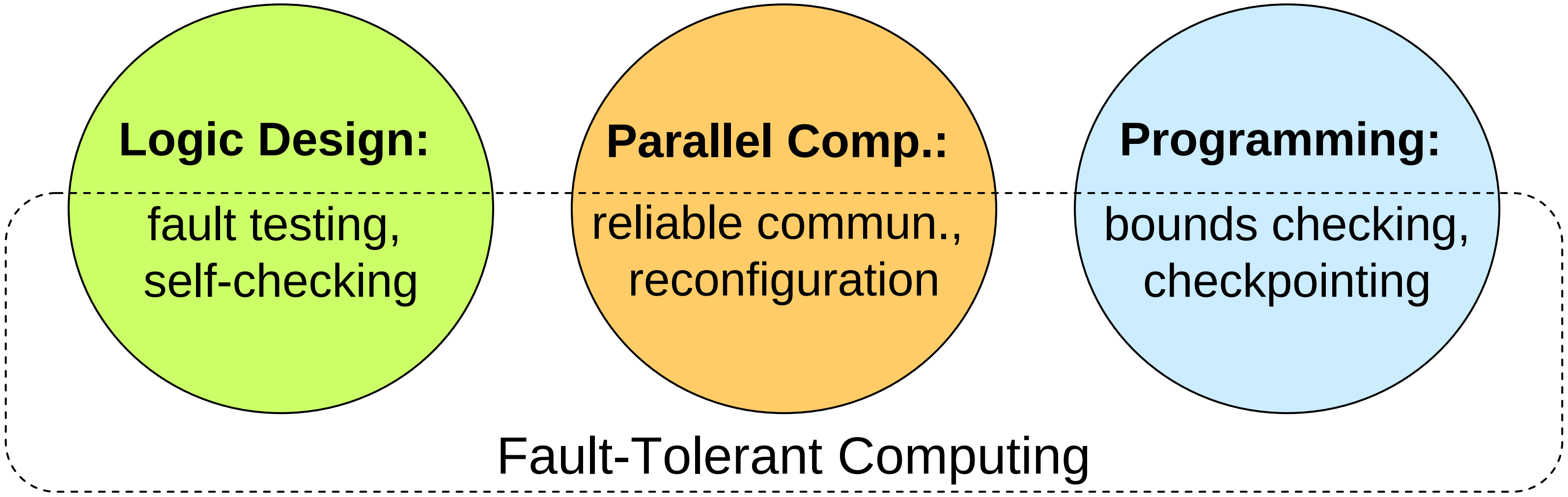
In 2000, the premier conference of the field was merged with another and renamed "Int'l Conf. on Dependable Systems and Networks" (DSN)

In 2004, IEEE began the publication of *IEEE Trans. On Dependable and Secure Systems* (inclusion of the term "secure" is for emphasis, because security was already accepted as an aspect of dependability)

Why This Course Shouldn't Be Needed

In an ideal world, methods for dealing with faults, errors, and other impairments in hardware and software would be covered within every computer engineering course that has a design component

Analogy: We do not teach structural engineers about building bridges in one course and about their safety and structural integrity during high winds or earthquakes in another (optional) course


Logic Design:
fault testing, self-checking
Parallel Comp.:
reliable commun., reconfiguration
Programming:
bounds checking, checkpointing
Fault-Tolerant Computing