Software Safety Basics

(Herrmann, Ch. 2)

Patriot missile defense system failure

- On February 25, 1991, a Patriot missile defense system operating at Dhahran, Saudi Arabia, during Operation Desert Storm failed to track and intercept an incoming Scud. This Scud subsequently hit an Army barracks, killing 28 Americans. [GAO]

http://news.bbc.co.uk/1/shared/spl/hi/middle_east/03/v3_iraq_timeline/html/scuds.stm

Patriot: A software failure

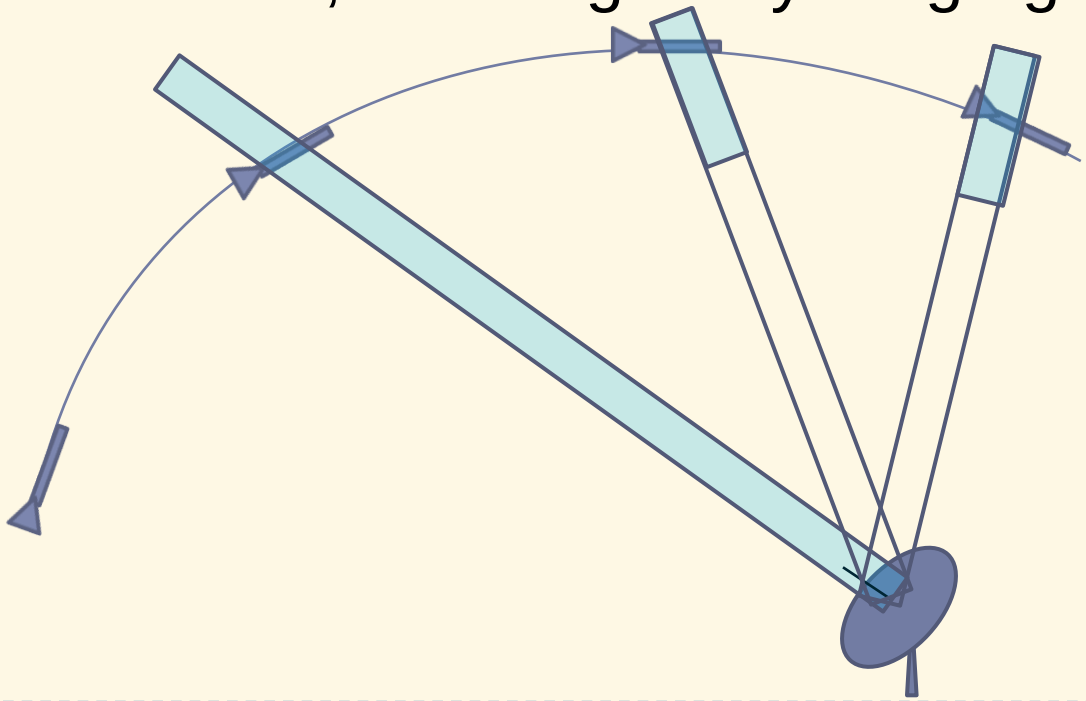
- [A] software problem in the system's weapons control computer… led to an inaccurate tracking calculation that became worse the longer the system operated.
- At the time of the incident, the battery had been operating continuously for over 100 hours. By then, the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud. [GAO]

Tracking a missile: what should happen

- Search: Wide range scanned

  When missile detected,

  range gate calculates the next area to scan
- Validation, Tracking: Only range gated area scanned



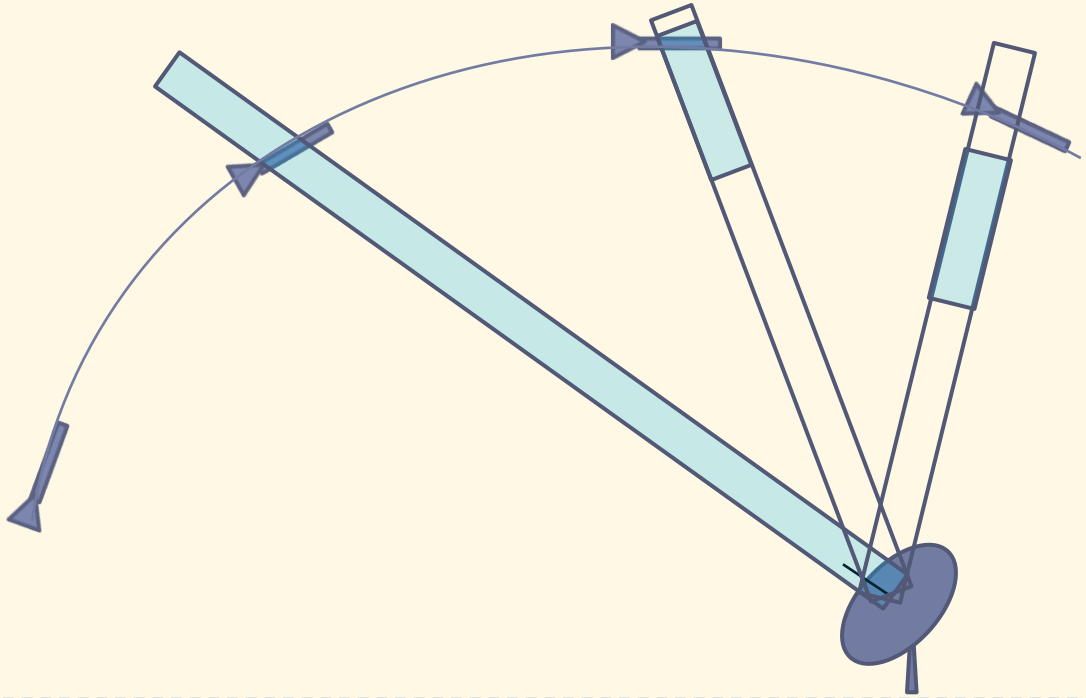
Software design flaw

- Range gate calculates predicated position from
  - Time of last radar detection:

    integer, measuring tenths of seconds
  - Known velocity of missile: floating-point value
- Problem:
  - Range gate used 24-bit registers, and each 0.1-second time increment added a little error
  - Over time, this error became significant enough to cause range gate to miscalculate missile position

What actually happened

- Range gated area shifted, no longer accurate



Sources of the problem

- Patriot designed for use against slower (Mach 2) missiles, not Scuds (Mach 5)
  - Proper calibration not performed – largely due to fear that adding an external recorder could crash the system(!)
- Patriot system typically used in short intervals – no longer than 8 hours
  - Supposed to be mobile, quick on/off, to avoid detection

Ariane 5 failure

- On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700m, the launcher veered off its flight path, broke up and exploded.

http://www.vuw.ac.nz/staff/stephen_marshall/SE/Failures/SE_Ariane.html

Ariane 5: A software failure

Unexpectedly large values encountered during alignment of "inertial platform"

↓

Attempt to convert overly large 64-bit value into a 16-bit value

↓

Software exception → Guidance system (hardware) shutdown

Sources of the problem

- Alignment code reused from (smaller, less powerful) Ariane 4
  - Velocity values of Ariane 5 were out of range of Ariane 4
- Ironically, alignment not even needed after lift-off!
- Why was alignment code running?
  - Engineers decided to leave it running for 40 seconds after planned lift-off time –
  - Permitting easy restart if launch was put on hold briefly

Panama Cancer Institute accidents

(Gage & McCormick, 2004)

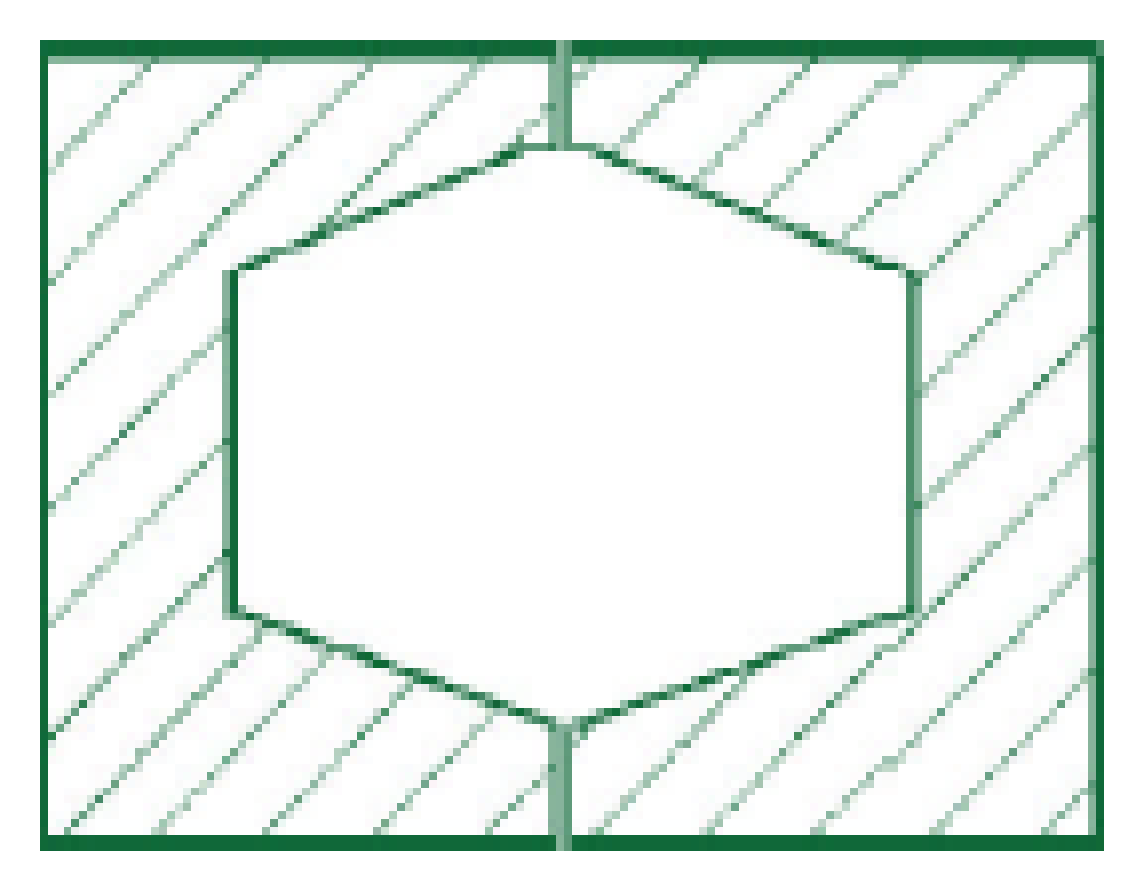
- November 2000: 27 cancer patients given massive doses of radiation
  - Partly due to flaws in Multidata software
  - Medical physicists who used the software were found guilty of 2nd degree murder in Panama
  - Note: In the well-known “Therac-25” incidents of the 1980s, software failures led to massive doses of radiation being administered to patients. Do we ever learn?...

Multidata software

- Used to plan radiation treatment
  - Operator enters patient data
  - Operator indicates placement of “blocks” (metal shields used to protect sensitive areas) through graphical editor
  - Software provides 3D prediction of where radiation would be distributed
  - From this data, dosage is determined

Block placement editor

NRC Information Notice 2001-08, Supp. 2



- Blocks drawn as separate polygons

  (There are 2 blocks in this picture)
- Software limitation: At most 4 blocks
- What if doctors want to use more blocks?

A “solution”

NRC Information Notice 2001-08, Supp. 2

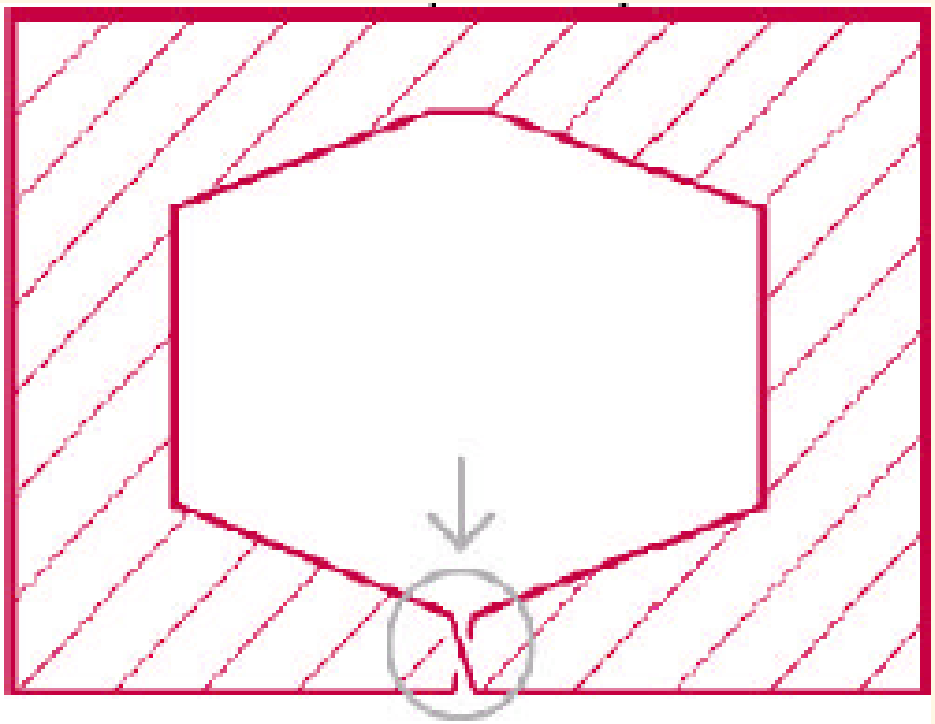


Fig. 1: Incorrect crossing block outline

- Note: This is a single unbroken line…
- Software treated it as a single block
- Now you can draw more blocks!

Fatal problem

- Dosage prediction algorithm expected blocks in the form of polygons, but graphical editor allowed non-polygons
- When run on non-polygon blocks, predictions were drastically wrong; overly high dosages prescribed

What is **software safety**?

- Features and procedures which ensure that
  - a product performs predictably under normal and abnormal conditions, and
  - the likelihood of an unplanned event occurring is minimized and its consequences controlled and contained;
- thereby preventing accidental injury or death, whether intentional or unintentional. (Herrmann)

Features and procedures

- Features: built into the software itself
  - Range checks; monitors; warnings/alarms
- Procedures: concern the proper environment for the software, and its proper use
  - Computer hardware that the software runs on
  - Physical, mechanical components of environment
  - Human users

Normal and abnormal conditions

- Abnormal conditions:
  - Failure of hardware components
  - Power outage
  - Extreme environmental conditions (temperature, velocity)
- What to do?
  - Not necessarily the best reaction, but one that has the best chance of preventing injury or death
  - Fail-safe: shut down
  - Fail-operational: continue in “simpler” degraded mode

Avoiding "unplanned events"

- To Herrmann, human users are the primary source of such events
  - Can produce unusual inputs or combinations of inputs
- User interface design, testing can be crucial to software safety
  - Understand user behavior
  - Create interfaces that guide users toward "good" input

Terminology alert #1

- There are many definitions of “safety”…
- Herrmann thinks of safety as a set of features and procedures
  - Something you can actually see in the software
- Leveson: “freedom from accidents or losses”
  - This is an idealized property of the software – something to aim for rather than actually achieve
- Storey distinguishes “safety” from “adequate safety”
  - Here, “safety” is close to Leveson’s definition;
  - “adequate safety” is closer to Herrman’s definition

Fault, error and failure

Fault: cause of error

- Misinterpreted software requirements
- Code defects: code doesn't implement required behavior

Error: dangerous state,
from which failure is possible

Failure: unacceptable system behavior

Fault, error and failure: Example

Fault: memory device fails
- Note: failure of component means fault at higher level
- If memory device not used, fault remains hidden

Error: attempt at memory access is unsuccessful
- Now, presence of fault is evident
- But, software may be able to handle it “gracefully”

Failure: unsuccessful memory access results in computing an “out of bounds” value

Faults: Hardware vs. software

- Some hardware faults may be random
  - Due to manufacturing defects or simple “wear and tear”
  - Probability can be estimated statistically
  - Well-known techniques to minimize random faults: error-correcting codes, redundant systems
- Software faults are always systematic – not random
  - Generated during design or specification – not execution
  - Software is not manufactured and doesn’t “wear out”
  - Techniques for minimizing random faults don’t work with systematic faults
    - Ariane 5 had redundant systems – all running the same software!

Fault management options

- Avoidance: Prevent faults from entering the system during the design phase
  - “good practices” in design – e.g. programming standards
- Removal: Find faults in the system before release
  - Testing – costly and not always very effective

Fault management options

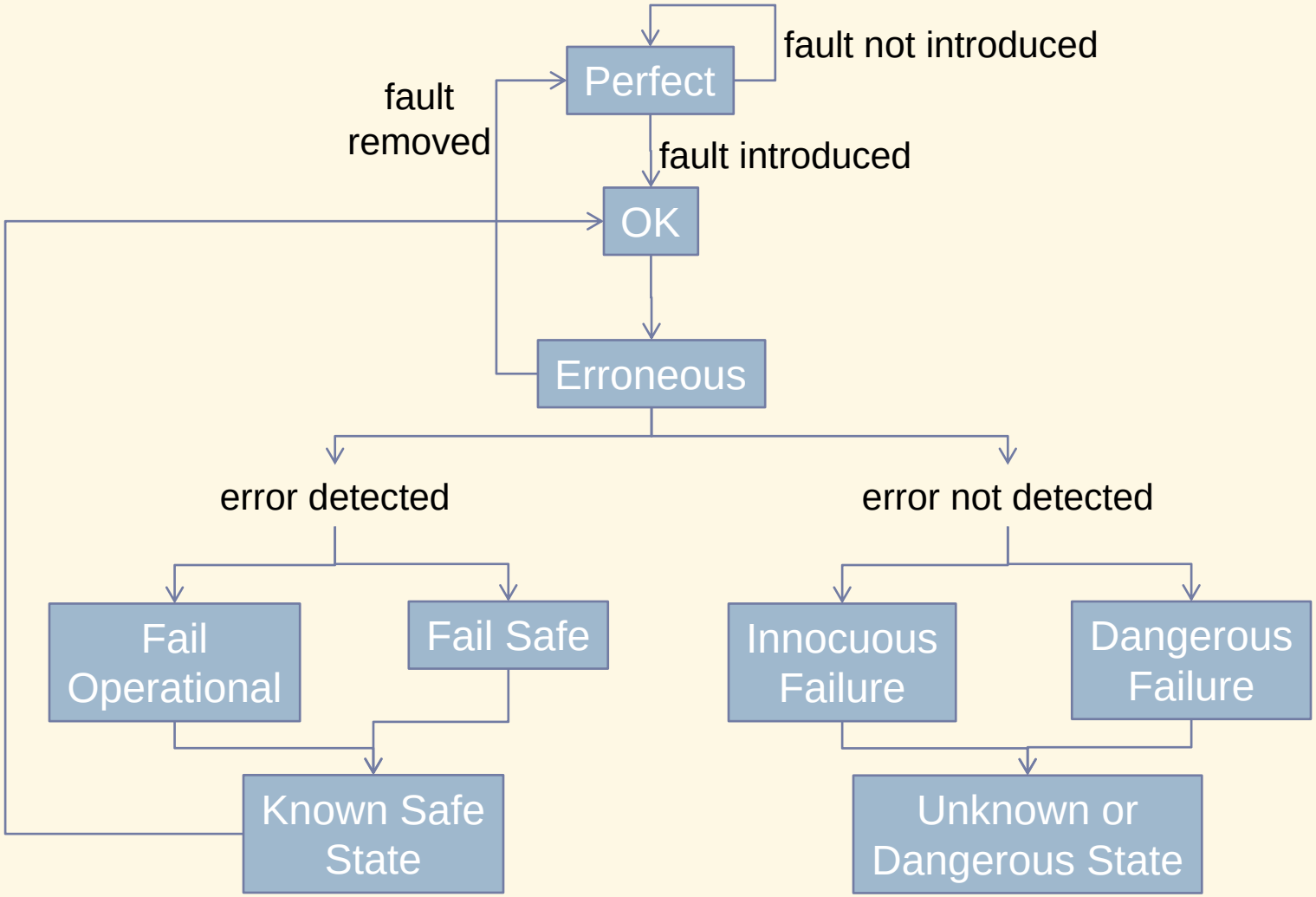
- Tolerance: Find faults in operational system after release, allow system to proceed correctly
  - Recovery blocks:
    - Create duplicate code modules
    - Run “primary module”, then run an “acceptance test”
    - If test fails, roll back changes and run an “alternative module”
  - *N*-version programming:
    - several independent implementations of a program

    Goal: ensure “design diversity”, avoid common faults

  Both approaches are costly, and may not be very effective

  For a study on whether *N*-version programming really achieves “design diversity”, read Knight & Leveson’s article.

Model of system failure behavior



Terminology alert #2

- “fault” and “error” have many alternative definitions
  - Sometimes, “error” is a synonym for what we’re calling “fault”, and “fault” means “behavior that may trigger a failure”
  - Following these alternative definitions, we have:

    error → fault → failure

References

- United States General Accounting Office. Report IMTEC-92-26, February 4, 1992. http://www.fas.org/spp/starwars/gao/im92026.htm
- Ariane 5 Flight 501 Failure – Report by the Inquiry Board. July 19, 1996. http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html
- U.S. Nuclear Regulatory Commission. Update on radiation therapy overexposures in Panama. NRC Information Notice 2001-08, Supp. 2, November 20, 2001. http://www.hsrd.ornl.gov/nrc/special/IN200108s2.pdf
- D. Gage and J. McCormick. Why software quality matters. *Baseline*, March 2004, 33-56. http://www.baselinemag.com/print_article2/0,1217,a=120920,00.asp
- Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, 1995.
- Neil Storey. *Safety-Critical Computer Systems*. Prentice Hall, 1996.
- J.C. Knight and N.G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. IEEE Transactions on Software Engineering 12(1), 1986, 96-109.