- You have a test on Tuesday
- Open book, open note
- No electronics: Phones, computers, calculators, etc.

Last Time

- Testing embedded software
  - Kinds of tests
  - When to test
  - How to test
  - Test coverage metrics
- Take home messages
  - Run each test as early and as often as time permits
  - Make tests cheap to run
  - Tests must be aggressive and must thoroughly exercise the system
    - Test coverage metrics help
  - Integration testing of independently developed components is the worst

Today

- Safety critical systems
  - What they are
  - Notorious examples
  - Ways to create them

Definition

- A system is critical when
  - Correctness is more important than cost, size, weight, power usage, time-to-market, etc.
  - Malfunction may result in unacceptable economic loss
  - Malfunction may result in unacceptable human loss
- Historical goal of software engineering: Increase developer productivity
- New goal: Increase software quality

Examples

- Now
  - UAVs fire missiles at people when operators push buttons
- Future
  - Autonomous fire control
- Now
  - Cars have ABS, traction control, stability control, automatic parking, automatic braking during cruise control
  - Electric and hybrid vehicles are highly computer-controlled
- Future
  - Automatic convoy formation
  - Automatic driving?

The Trend

- Humans are relinquishing direct control of more and more safety-critical systems
- Humans are flawed (forgetful, inattentive, etc.) but basically have good judgment and can grasp the big picture
  - Training helps a lot
- When the human is removed from the loop there is no oversight
  - Things may go badly, quickly
- In general: Larger systems are much worse because…
  - They are more complicated
  - They can do more harm when they go wrong

Critical Systems

- Obviously not just a software problem!
- Other angles that need to be considered
  - Faulty specification
  - Faulty hardware
  - Human error
  - Malicious users
  - Malicious non-users
- Today we focus on software issues

Risk

- Minimizing overall risk is the general strategy
- However:
  - Risk is fundamental and unavoidable
  - Risk management is about managing tradeoffs
  - Risk is a matter of perspective
    - Standpoint of individual exposed to hazard
    - Standpoint of society – total risk to general public
    - Standpoint of institution responsible for the activity
  - Quantifying risk does not ensure safety

Design for Safety

- Order of Precedence:
  1. Design for minimum risk
  2. Incorporate safety devices
  3. Provide warning devices
  4. Develop procedures and training

Case Study 1: Missile Timing

- Military aircraft modified from hardware-controlled to software-controlled missile launch
- After design, implementation, and testing, plane was loaded with a live missile and flown
- The missile engine fired, but the missile never released from the aircraft
  - A "wild ride" for the test pilot
- Problem: Design did not specify for how long the "holdback" should be unlocked
  - Programmer made an incorrect assumption
  - Missile was not unlocked for long enough to leave the rack
  - Oops!

Case Study 2: Missile Choice

- Weapon system supporting both live and practice missiles used in field practice
- Operator "fires" a practice missile at a real aircraft during an exercise
- Software is programmed to choose the best available weapon for a given target
  - Deselects the practice missile
  - Selects a live missile
  - Fires
- From the report: "Fortunately the target pilot was experienced … but the missile tracked and still detonated in close proximity."

Case Study 3: Therac-25

- Computer-controlled radiation therapy
- Goal: Destroy tumors with minimal impact on surrounding tissue
- 11 systems deployed during the 1980s
- Many new features over previous versions
- Lots of software control
- Increased power
- From a report: "The software control was implemented in a DEC model PDP 11 processor using a custom executive and assembly language. A single programmer implemented virtually all of the software. He had an uncertain level of formal education and produced very little, if any, documentation on the software."

More Therac-25

- Outcome: Six massive radiation overdoses, five deaths
- Compounding the problem: Company didn't believe software could be at fault
- "Records show that software was deliberately left out of an otherwise thorough safety analysis performed in 1983 which used fault-tree methods. Software was excluded because 'software errors' have been eliminated because of extensive simulation and field testing. Also, software does not degrade due to wear, fatigue or reproduction process. Other types of software failures were assigned very low failure rates with no apparent justification."

Therac-25 Facts

- Hardware interlocks were replaced by software without any supporting safety analysis
- There was no effective reporting mechanism for field problems involving software
- Software design practices (contributing to the accidents) did not include basic, shared-data and contention management mechanisms normal in multi-tasking software
- The design was unnecessarily complex for the problem. For instance, there were more parallel tasks than necessary. This was a direct cause of some of the accidents

Specific Therac-25 Bugs

- "Equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly. This was evidently missed during testing, since it took some practice before operators were able to work quickly enough for the problem to occur."
- "The software set a flag variable by incrementing it. Occasionally an arithmetic overflow occurred, causing the software to bypass safety checks."

Two More Examples

- Ariane 5 (1996)
  - Reused Ariane 4 software
  - Higher horizontal velocity of Ariane 5 caused a 16-bit variable to overflow
  - Resulting chain of failures necessitated destroying the rocket
- Mars Pathfinder (1997)
  - "…very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread."
  - Classic priority inversion
  - We'll cover these (and how to avoid them) later

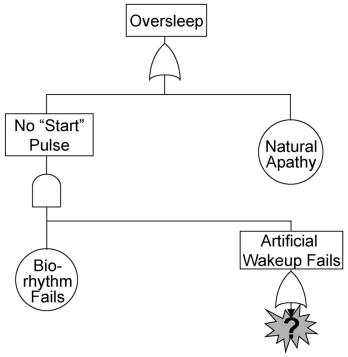
Software V&V

- Verification and validation – making sure that software does the right thing
  - Obvious problem – figuring out what "the right thing" is
- V&V typically uses 40-60% of total effort
- Parts of the solution:
  - Development process
  - Specification
  - Testing
  - Inspection
  - Fault-tree analysis
  - Language choice
  - Software techniques
  - Static analysis / formal verification

Fault Tree Analysis

- Fault: Abnormal or undesirable system state
- Failure: Loss of functionality
- Primary failure: Failing element operating within its specifications
- Secondary failure: Failing element operating outside of its specifications
  - What was the primary failure for the Challenger?
- Fault tree idea:
  - Connect faults and failures using logical operations in order to understand the consequences of faults individually and in combination

Fault Tree Example



Language Choice

- We looked at MISRA C
- A better example is SPARK Ada
  - Pervasive support for static analysis
    - All rule violations statically detectable
  - No implementation-defined behavior
  - Tons of tool support
  - "SPARK code was found to have only 10% of the residual errors of full Ada and Ada was found to have only 10% of the residual errors of C"

Software Safety Techniques

- Interlocks
  - Important actions require signals from two or more independent sources
- Firewalls
  - Hardware / software protection boundary between critical and non-critical functionality
  - Example software firewalls:
    - Address spaces
    - CPU reservations
    - Module tainting
- Redundancy
  - Correct result is computed by having multiple processes / processors vote

Static Analysis / Verification

- Goal: Rule out classes of errors across all possible executions
  - Potentially much faster than testing
  - Potentially more thorough than testing
  - Stack depth analysis is one example – can "prove" that a system is invulnerable to stack overflow
- Other examples – absence of:
  - Array / pointer misuse
  - Integer and FP overflow
  - Use of uninitialized data
  - Numerical errors
  - Exceptions
  - Deviation from specification

Static Analysis Properties

- In general static analysis:
  - Finds many trivial problems
  - Does not find the most important problems
  - Has a hard time finding the deepest problems
  - Has high computational complexity
- Even so:
  - My opinion (and lots of other people's) is that static analysis is the only way forward for creating large safety-critical systems

Example: PolySpace Verifier

- Pure static analysis
  - No testcases, no execution
- Analyzes Ada, C, C++
  - Anecdotal evidence suggests that Ada version is by far the best
  - Deep analysis of large C, C++ programs is still barely feasible
- Found the Ariane bug
  - (After the fact)
- Lots of big customers

Summary

- Safety-critical software is extremely hard to create
  - Can't be done by organizations that do not take this very seriously
  - No shortage of horror stories
- Risk mitigation requires many different techniques
  - Software-based
  - Hardware-based
  - Process-based
  - Etc.