These tutorials are a simplified introduction, and are not sufficient on their own to achieve system safety. You are responsible for the safety of your system.

“Engineering is achieving function while avoiding failure.”

– Henry Petroski
Anti-Patterns for Embedded System Safety:

- Requirements do not address safety
- Not using an appropriate safety standard
- Safety analysis assumes perfect software
- Redundancy management inadequate

Actually know system is safe

- Correctness is only a starting point
  - Requirements and other aspects matter
- Fault responses must be safe
  - Hardware faults (permanent; transient)
  - Software faults

Is Your System Appropriately Safe?

General Motors recalls 4 million vehicles after software linked to 1 death

http://goo.gl/EgxHEo

The company said Friday that in rare cases, the car's sensing and diagnostic module — a tiny computer that senses what the vehicle is doing and controls air bag deployment — can go into test mode. If that happens, the front air bags won't inflate in a crash and the seat belts may not work either.

http://goo.gl/vnqH7G
Defense-In-Depth For Safety

Each mitigation level attempts to prevent escalation to next level:

- **Avoid faults occurring**
  - Careful design of software to avoid software defects
  - Use robust hardware to avoid hardware run-time faults

- **Detect and contain faults**
  - Error correction HW, redundant CPUs
  - Watchdog timers for failed tasks, exception handling

- **Use Fail Safe strategies to mitigate hazards**
  - For example, automatic safety shutdown mechanisms

- **Incidents require operator intervention (or luck)**
  - Operator may be able to react correctly and quickly
  - Incident will be a mishap some fraction of time

- **Want to avoid escalation as much as possible**
  - E.g., fail safe approaches that work to avoid incidents

(For more information, see *Safeware*, Leveson 1986, pp. 149-150)
Basic Safety Principles

- Safety must be seen to be present
  - System presumed unsafe unless convincing safety argument made
  - Outsider must be able to determine safety purely from documents
- The greater the risk, the greater the need for information
  - Riskier systems require more engineering rigor
- Safety must be built in, not added on
  - If code is created without a safety process, throw it away; start over
- Systematic, random, and malicious faults all matter
  - Consider design errors and transient faults (e.g., soft errors)
  - If it’s not secure, it’s not safe
- Safety must be argued in writing and demonstrated
  - Failure-free testing isn’t enough
- Safety is a lifecycle concern
  - “Mission critical failures” can be considered “safety” as well
Safety Culture: Everyone Is Sure It’s Safe

- **Space Shuttle Challenger Mishap**
  - January 1986 launch explosion; 7 fatalities
  - Dual O-rings keep hot gases inside solid booster
    - History of sometimes failing if too cold
    - At launch, joint temperature was below freezing
  - Booster team told: “prove launch is unsafe”
    - Should have been: “no launch unless proven safe”
    - Getting lucky is not the same thing as being safe
Overview of Embedded System Safety

Safety Topics:
- Safety Plan & Safety Standards
- Safety Requirements
- Critical System Design
- Dependability
- Single Points of Failure
- Redundancy Management
- Isolation Mechanisms
- Safety Architectural Patterns

Pitfall:
- Safety isn’t just about whether you think it’s safe …
- … it’s about whether you can prove it is appropriately safe

Software-Controlled Radiation Therapy Mishaps