NC STATE UNIVERSITY

Dependable and Safety-Critical Systems

NC STATE UNIVERSITY

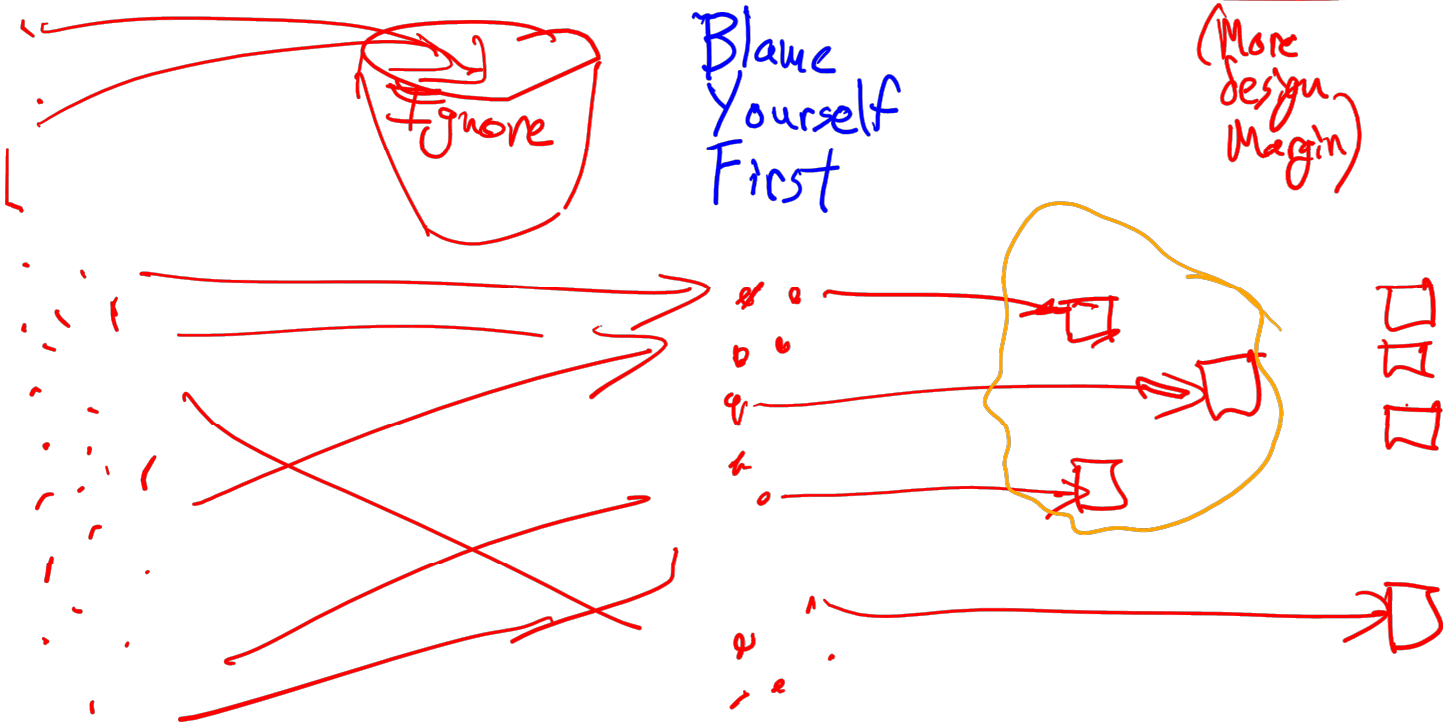
References

- **Practical Design of Safety-Critical Computer Systems**, William R. Dunn, Reliability Press, 2002
- MIL-STD-882E: Standard Practice for System Safety

NC STATE UNIVERSITY

BASIC CONCEPTS

Faults | Categorize | Prioritize | Design Modifications: Prevent | Reduce Prob. (More design Margin) | Handle | Handle at Run-Time



NC STATE UNIVERSITY

Fault Model: Which Faults Are We Considering?

- Hardware examples
  - 3.3 V regulator fails.
    - Fail short (5 V), fail open (0 V), fail at some other voltage?
  - Inductor not soldered fully (open circuit)
  - I^2C pullup resistor open
  - LCD module connection has some open circuits
  - RAM, Flash ROM bit flips
  - *CPU multiply instruction fails?*
- Software examples
  - Bugs in code: Generic, could do almost anything!
  - Need to be more specific (Project)
    - Corrupt important shared data
    - Reconfigure or disable the ADC or TPM
    - Overwrite TPM PWM duty cycle
    - Reconfigure the port pins.
    - Disable CPU clock, drop the CPU clock speed so current rises.
    - Disable all interrupts
    - Keep ADC from generating interrupt
    - Put an infinite loop in a high-priority ISR
    - Keep key threads from running (lower their priority, hog processor, take but don't release resources they need)
    - Misuse queues (make full or empty)
    - Use up RTOS object memory preventing queues from working
    - Overflow the stack
    - Crash the RTOS

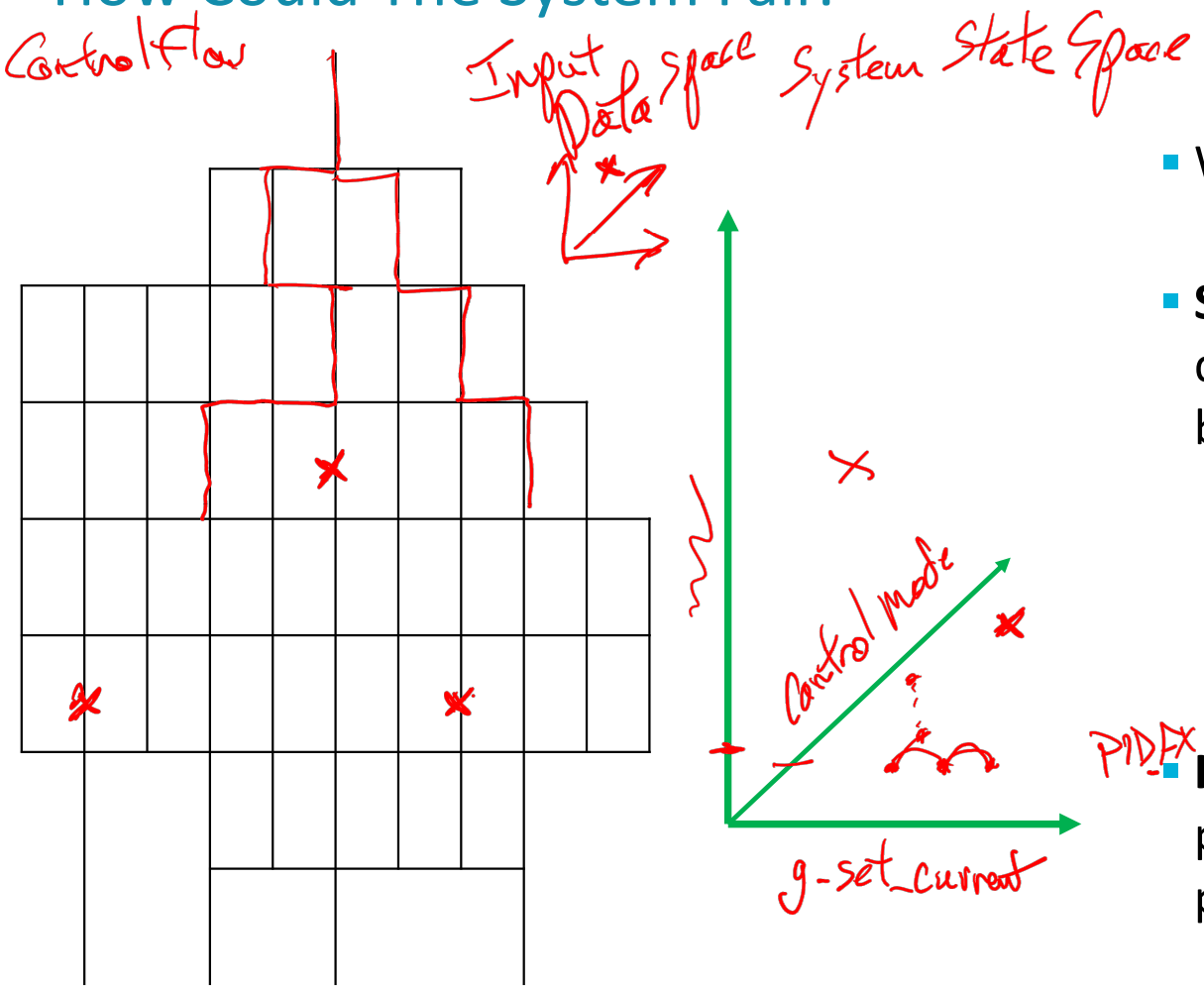
NC STATE UNIVERSITY

Software doesn't wear out. It's already broken.

- Embedded systems should always work, but it's not easy to get the software perfect
  - Must handle many different scenarios and error conditions
  - Hard to anticipate all of them when creating requirements and code
  - Hard, slow and expensive to test all of them (if even possible)

- So what should we do?

How Could The System Fail?



- What could possibly go wrong?
- **Software** isn't perfect, so it has latent defects lurking, waiting to be triggered by a fault (input and/or condition)
  - Bad inputs or bad system state → defect activated → device fails to perform properly
  - Good inputs and system state, but not considered in design → defect activated → device fails to perform properly
- **Hardware** failure (transient or permanent) → device fails to perform properly

Responses

- How should system respond to each fault?
  - Ignore the problem?
    - "Problem? What problem?"
  - Declare out of scope?
    - "Not my problem!"
  - Stop working?
    - Shut down.
    - **Fail Stop**.
  - Stop working in a way that minimizes further damage?
    - Move to safe mode, then stop.
    - **Fail Safe** (subset of **Fail Stop**)
  - Keep working (as able)?
    - Handle failure and continue (possibly in degraded mode)
    - **Fail Operate**. Much harder to provide this, but very useful.
  - Restart and hope it doesn't happen again?
    - May want to save a checkpoint.
- How should system respond to **multiple concurrent faults?**
  - Non-trivial (hard)

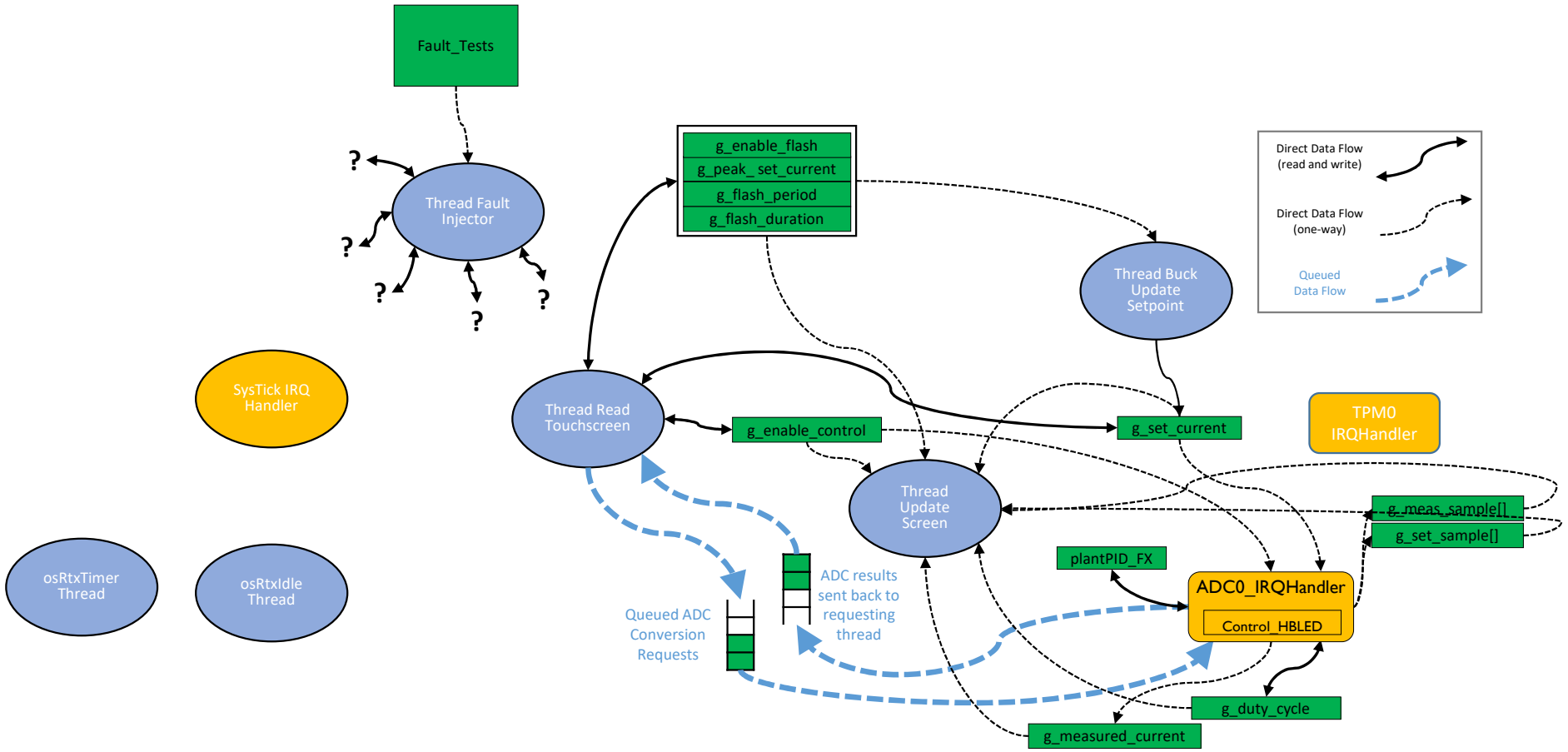
Common Types of Solutions

- Design the fault out of the system
  - May be feasible to modify design so that fault is impossible
- Don't design-in or code-in the fault
  - Be more careful!
- Design-in or code-in the fault, but find it as early as possible during development
  - Testing, peer review
- Accept the fault may happen, so design system to tolerate it during operation
  - System masks the fault. E.g. RAM protected by hardware ECC (error-correcting code).
  - System detect the faults and responds (manages it)

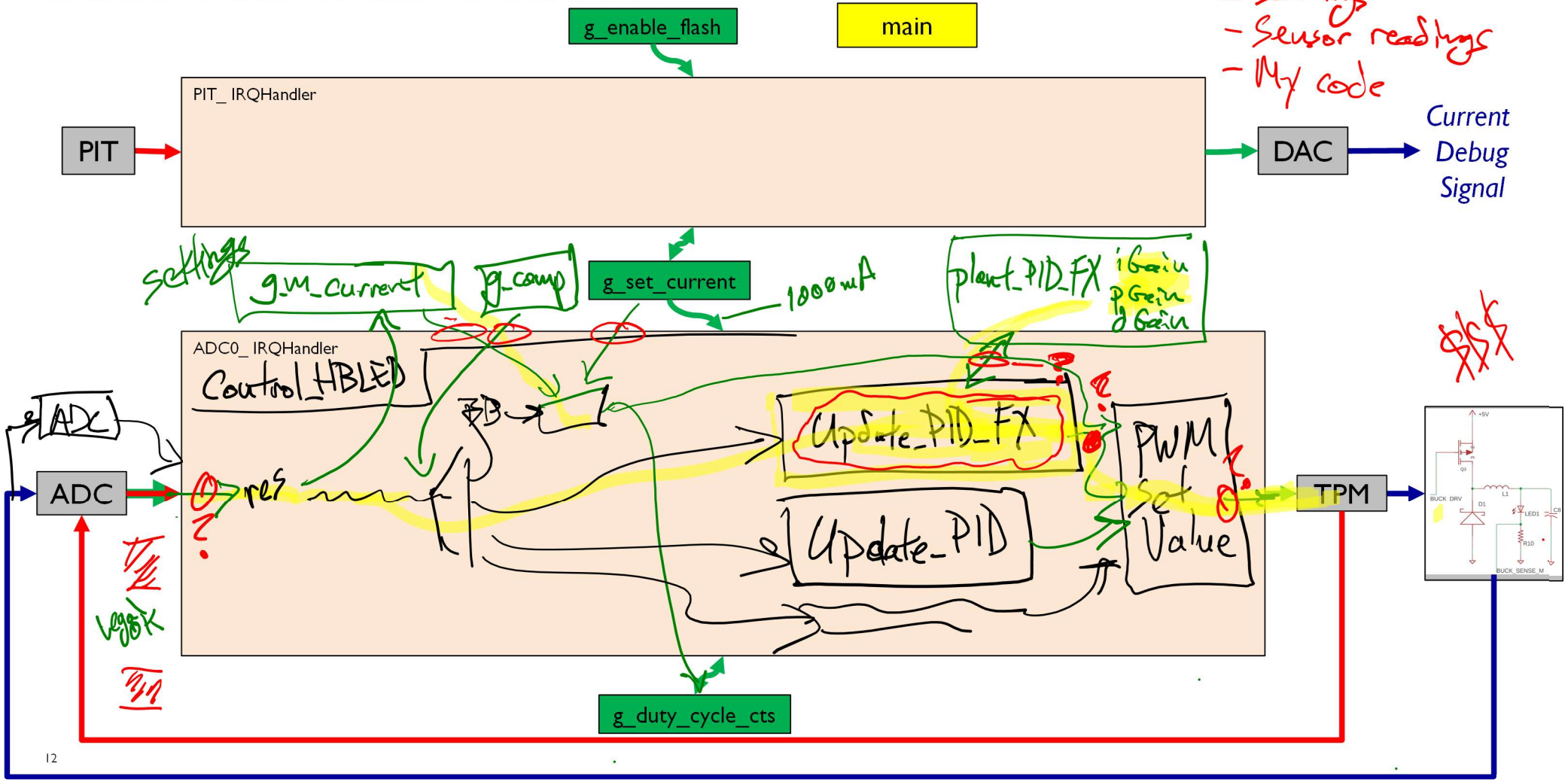
NC STATE UNIVERSITY

Basic Fault Management Concepts

- Start with
  - System diagram
  - Fault model
- Identify where system is vulnerable to faults
- Prioritize faults
- For each fault
  - Define a fault-management strategy (more later in process discussion)
    - Prevent during development?
    - Handle during operation?
- Fault handling during operation
  - Develop detection method
  - Develop response (continue operating? restart?)
  - Discuss, implement, test


Fault_Tests
Thread Fault Injector
?
?
?
?
?
g_enable_flash
g_peak_ set_current
g_flash_period
g_flash_duration
Thread Buck Update Setpoint
Direct Data Flow (read and write)
Direct Data Flow (one-way)
Queued Data Flow
SysTick IRQ Handler
Thread Read Touchscreen
g_enable_control
g_set_current
TPM0 IRQHandler
Thread Update Screen
g_meas_sample[]
g_set_sample[]
plantPID_FX
ADC0_IRQHandler
Control_HBLED
ADC results sent back to requesting thread
Queued ADC Conversion Requests
g_duty_cycle
g_measured_current
osRtxTimer Thread
osRtxIdle Thread

Buck Converter Controller

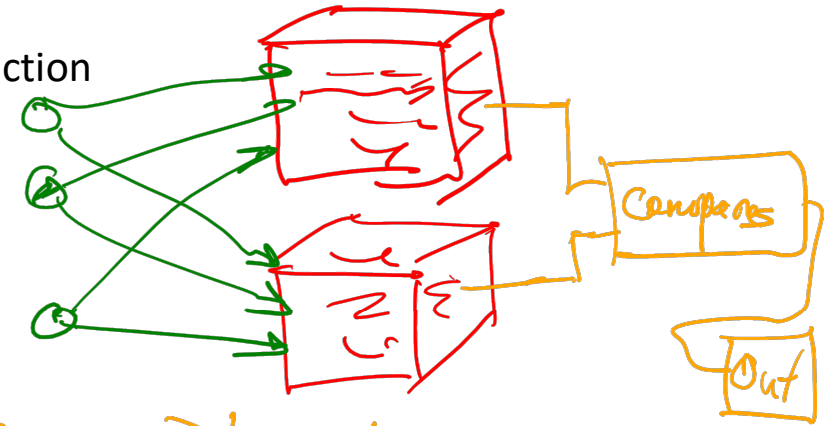


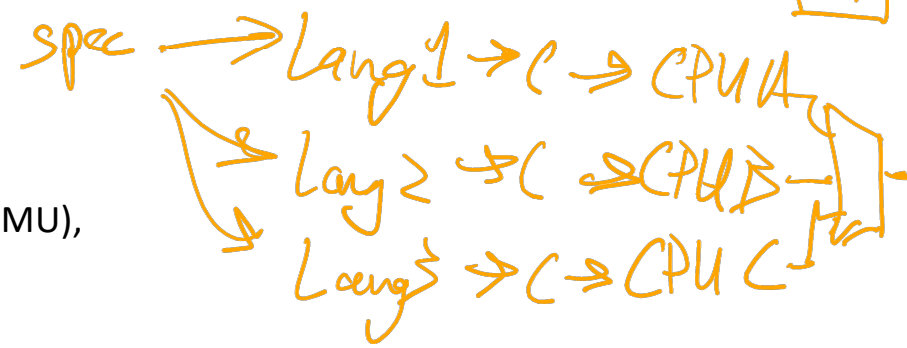
NC STATE UNIVERSITY

FAULT DETECTION AND RESPONSE

Range of Fault Detection Methods

- As you know more about the application, more precise fault detection becomes possible
- Coarse-Grain Example: Apply redundancy at very high level
  - Duplicate the controller and compare the outputs
  - Mismatched outputs indicate fault has occurred
  - Misses common-mode faults
- Apply generic Built-In Test (BIT)
  - Memory tests, CPU tests, etc.
- Safety Monitors
  - Watchdog timer (WDT)
  - Memory protection unit (MPU) or memory management unit (MMU),
  - Low voltage detector
  - Others possible…
- More detailed application knowledge → more accurate fault detection
  - Input range checking, variable protection with CRC, etc.



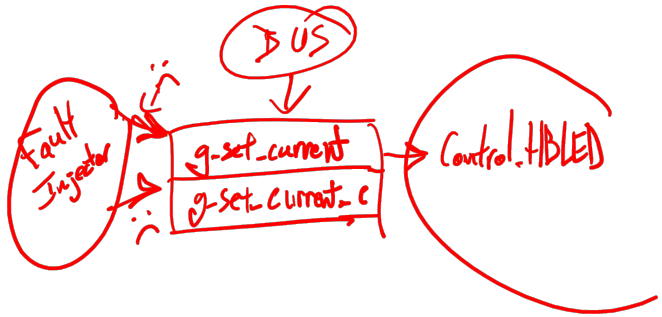


Detecting Data Faults

- Has data been updated correctly?
  - Store extra information to confirm valid updates
  - Use extra memory (fine-grain to coarse-grain)
    - Parity bit, EDC/ECC
    - Data copy/copies
    - Data complement
    - CRC/hash of data
  - Read: Confirm validity before use
  - Write: Update redundancy with data
- Is data reasonable, within expected range?
  - Simple bounds
  - Near previous value (or trend)

5: 000101
58: 111010
Xor 111111 = 63

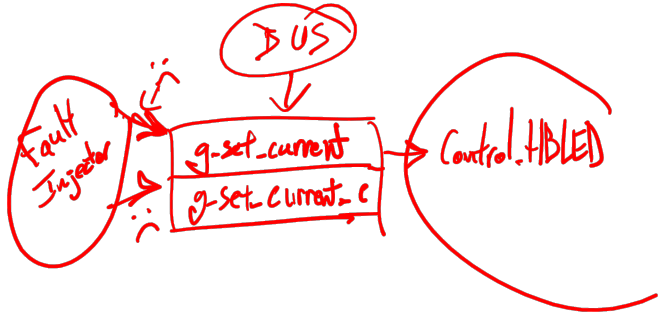
- Near expected value from model
- Other ways?



pGain[3]
Use: Check all all 3 copies match?
Else, if 2 match, use that value and update the other

Responding to Data Faults

- Logging
  - Log a fault code?
- Processing flow
  - Restart system?
    - Save checkpoint first?
  - Switch to degraded mode?
    - Which data value to use? Corrected, old, estimated…
  - Continue normally
    - Which data value to use? Corrected, old, estimated…



NC STATE UNIVERSITY

Protecting the HBLED Buck Driver (High and Low)

Fault	Detection Methods	Possible Responses
Corrupt important shared data.	*Hide data (make it file local), add access functions, integrity check (CRC/complement), range checking*	*Use previous value? Clip to valid range?*
Reconfigure or disable the ADC or TPM.	*Check configuration periodically?*	*Scrub peripheral configuration*
Overwrite TPM PWM duty cycle	*?*	*?*
Reconfigure the port pins (if not lockable)	*Check configuration periodically?*	*Scrub peripheral configuration*
Disable CPU clock or drop the CPU clock speed so peak current rises.	*Check configuration periodically?*	*Scrub clock configuration*
Disable ADC interrupt (or all), put an infinite loop in a high-priority ISR.	*WDT*	*?*
Keep key threads from running (via priority or shared resources).	*WDT*	*?*
Misuse queues (make full or empty),	*Test status returned by OS calls*	*?*
Use up RTOS object memory preventing queues from working.	*Test status returned by OS calls*	*?*
Overflow the stack.	*Perform stack overflow detection*	*?*
Crash the RTOS	*WDT*	*Reset*

NC STATE UNIVERSITY

SYSTEM ARCHITECTURE OPTIONS

System Architecture Options – Independent of Application

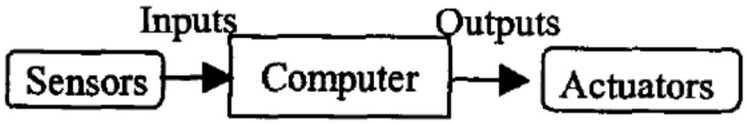


Fig. 4. Simplex, no fault tolerance

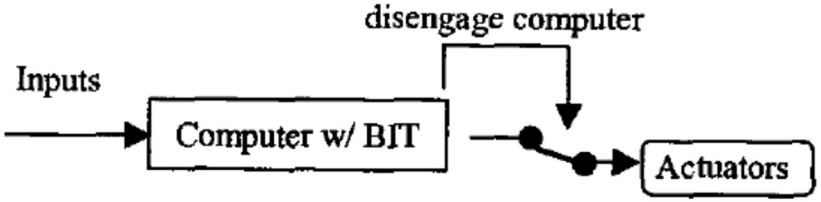


Fig. 5. Simplex, disengagement features

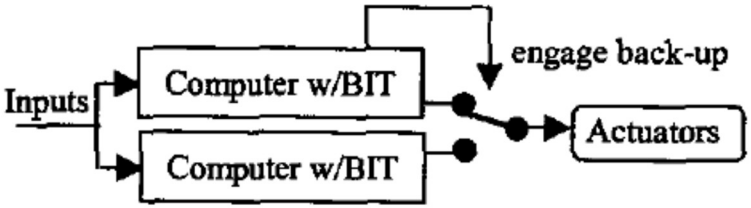


Fig. 6. Dual standby FT architecture

- Built-in Test (BIT) used to detect faults
- Functional Safety requirements (e.g. IEC 60730) specify running BIT
  - At start-up
  - Periodically

More System Architecture Options

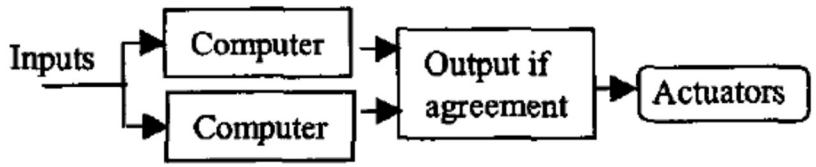


Fig. 7. Self-checking pair

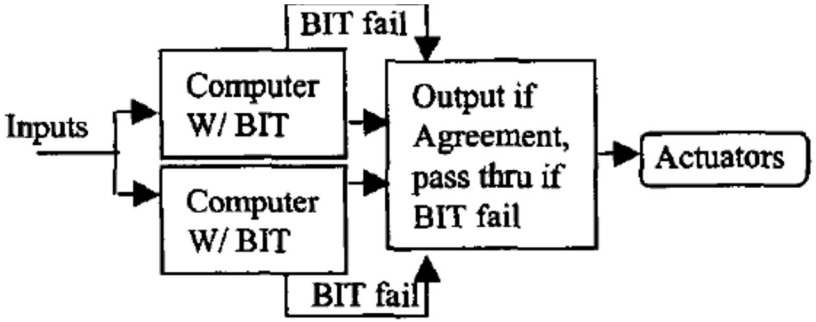


Fig. 8. Self-checking pair with simplex fault down

- Note that inputs are still shared between computers (channels)
  - Single point of failure. If the oven's thermal sensor fails, then both channels will be wrong (but agree)

More System Architecture Options

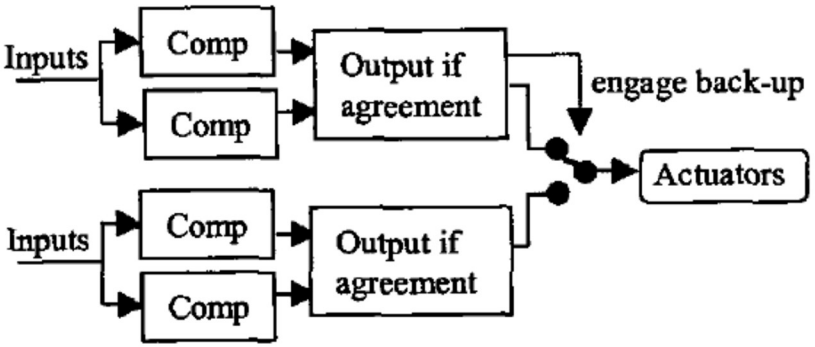


Fig. 9. Dual self-checking pair

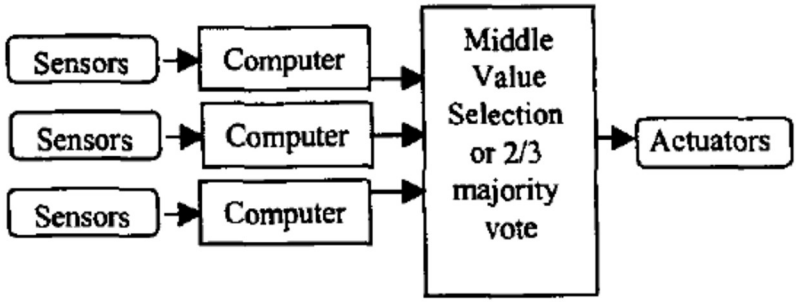


Fig. 10. Triple modular redundancy

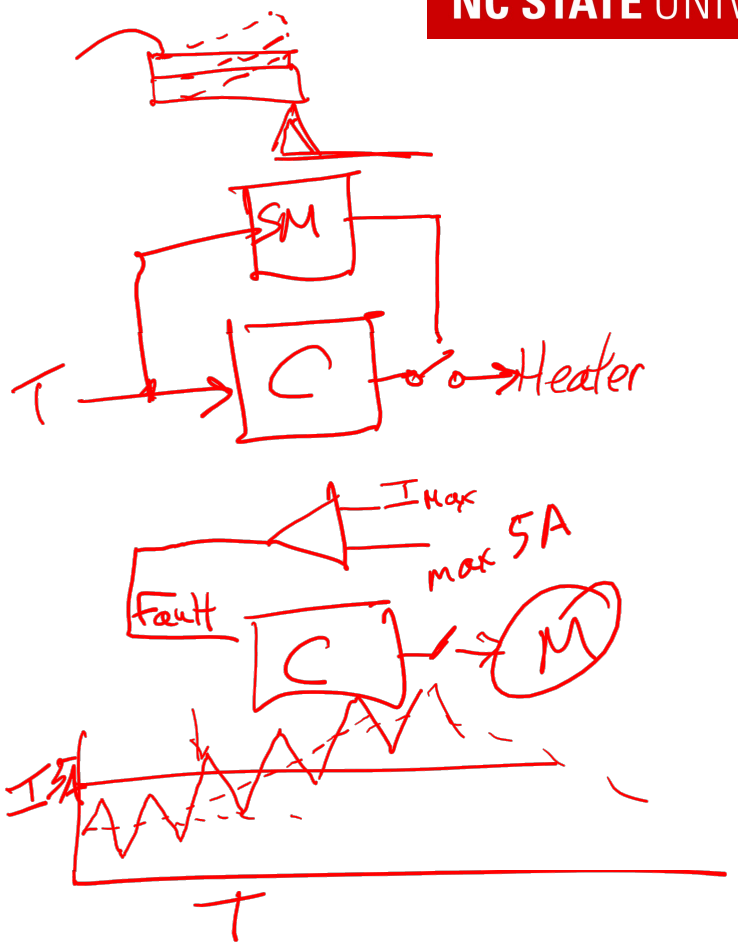
- Duplicate sensors to eliminate single point of failure
- Reduces some common mode failures, but some may remain

Safety Invariant Conditions

- Can we define simple tests which tell us system **may be** operating **unsafely?**
- Then we can add monitor to disable system when we enter the **maybe unsafe** zone
- Where do we draw the line?
  - False positive: System is safe, but test says maybe unsafe
  - False negative: System is unsafe, but test says not unsafe (i.e. safe)

External Safety Monitor

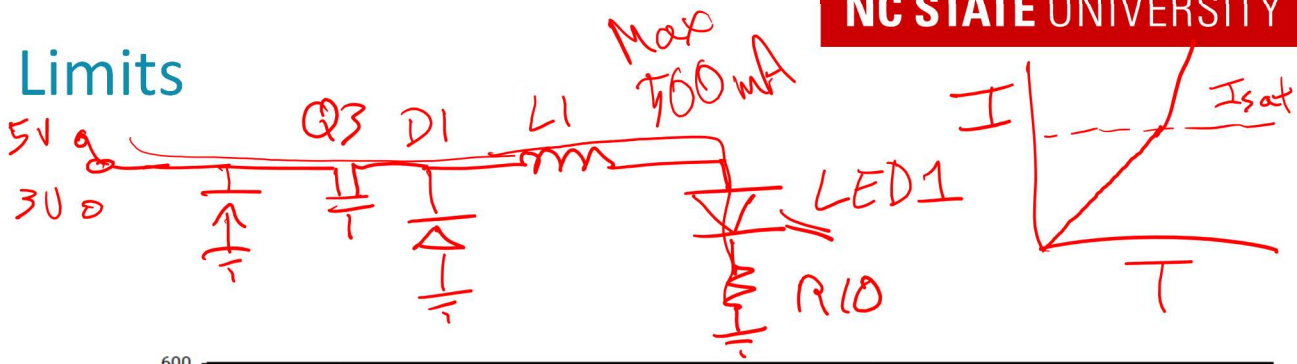
- Use separate simple device to monitor system, disable it if safety invariant conditions are violated
- Over-temperature protection for oven
  - Temperature must never rise above 500º F
  - Thermal switch (bimetallic) opens circuit above max. allowed temperature, removing power to heating element (or controller?)
- Over-current protection for switch-mode power conversion and motor control
  - Motor current never above 5 A
  - Control system will eventually detect and respond to over-current, but may be too late to prevent damage
  - Use dedicated hardware: analog comparator triggers when overcurrent, turns off PWM output via fault input



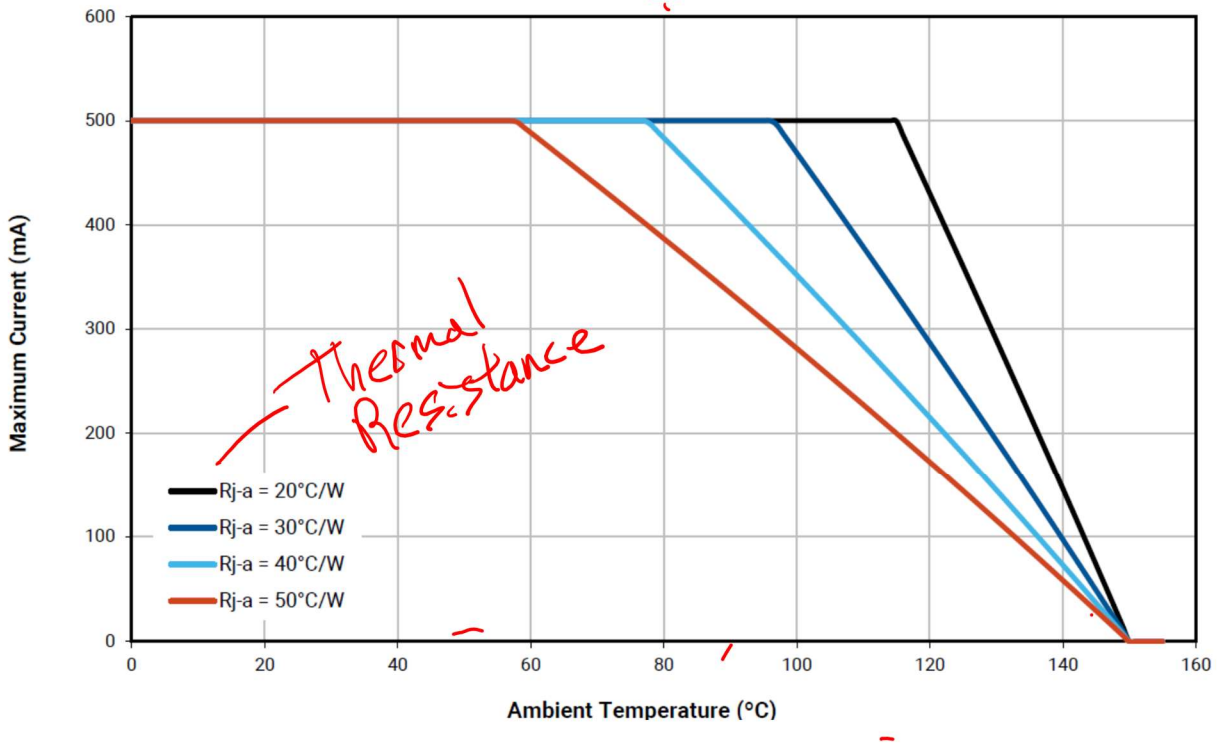
NC STATE UNIVERSITY

HBLED DRIVER ANALYSIS

Current and Temperature Limits



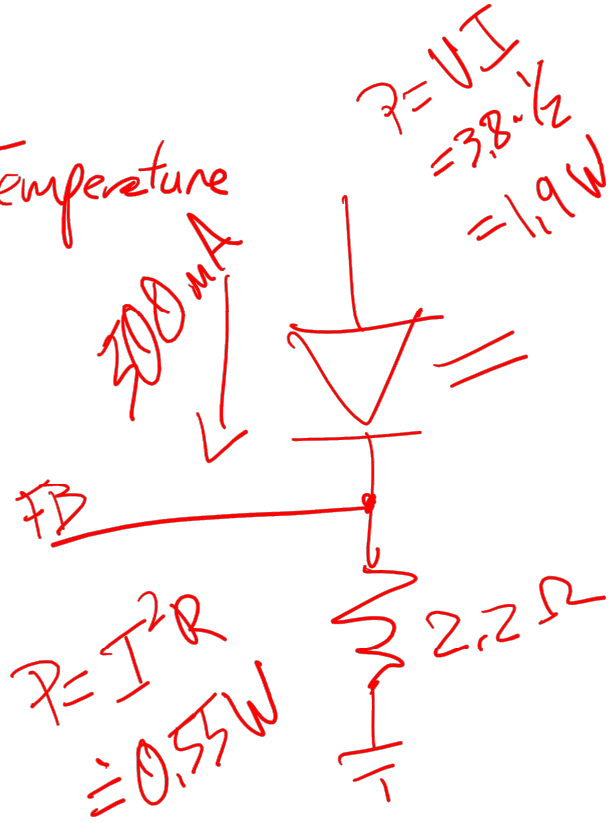
- Transistor Q3: BSS215P.
  - Max I_D @ 25C = -1.5 A.
  - $R_{DS(on)}$ = 105-280 mΩ
- Diode D1: DB2J20900LSMD
  - I_{FAve} = 0.5 A, I_{FSurge} = 3 A
- Inductor L1: SDR0604-680KLCT-ND.
  - $I_{RMS\ Max}$ = 0.62 A
  - I_{sat} = 0.84 A
  - R_{DCMax} = 0.52Ω
- LED1: MLEAWT-A1-0000-0003F7
  - Nominal I_{max} = 500 mA
  - T_{max} for LED Junction = 150º C



Fault Tree Analysis for HBLED Driver

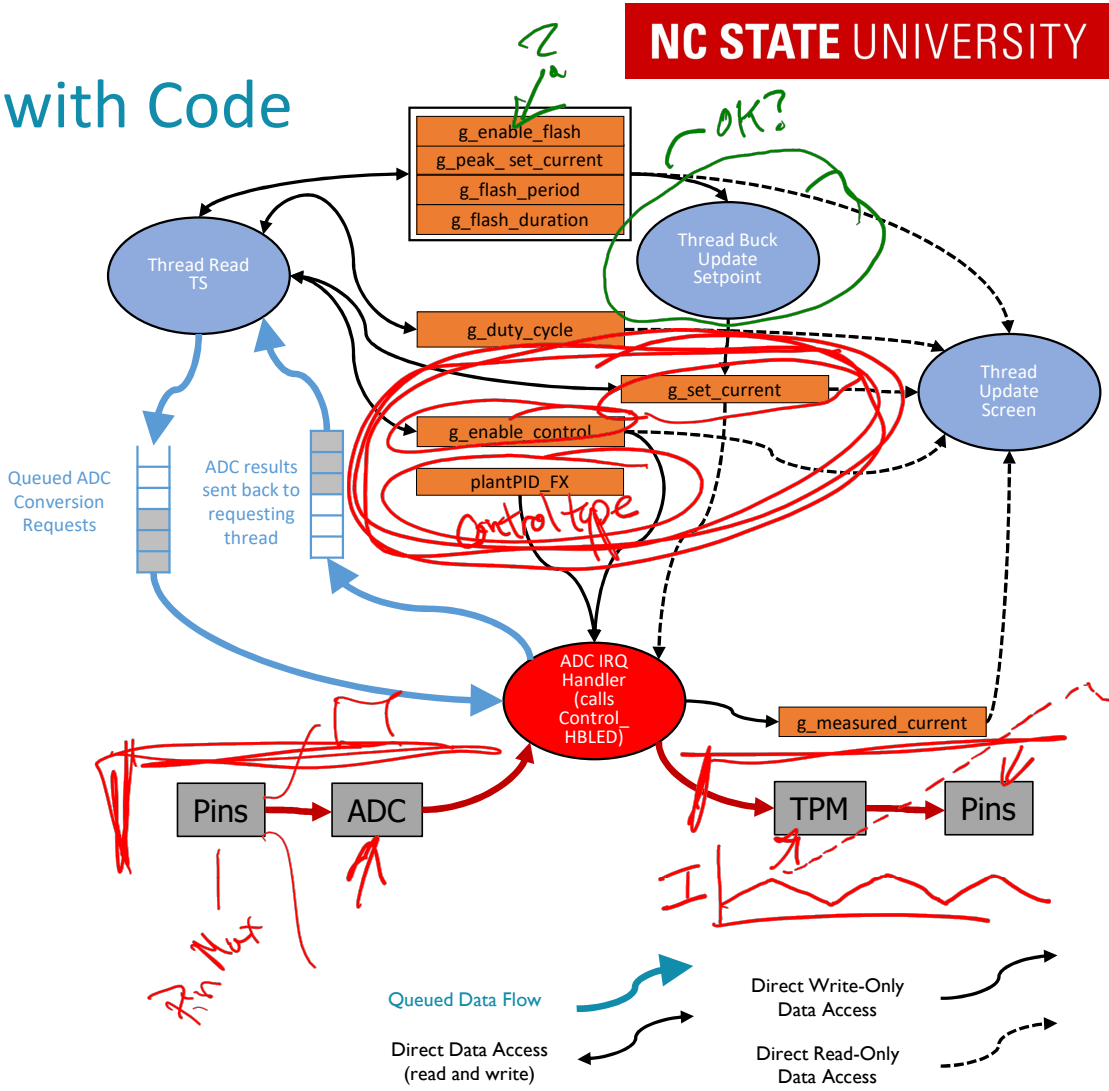
- How can the hardware be damaged?

Too much current for LED at given Temperature
- Q3 - Fail closed ckt.
- Metal shorting Q3
- Inductor fail short or Saturate
- Bad Soldering
- Bad Temp. reading



Crashing the HBLED Buck Driver with Code

- Reconfigure or disable the ADC or TPM
- Overwrite TPM PWM duty cycle
- Reconfigure the port pins
- Disable CPU clock, drop the CPU clock speed so current rises
- Disable all interrupts, keep ADC from generating interrupt, put an infinite loop in a high-priority ISR
- Keep key threads from running (lower their priority, hog processor, take but don't release resources they need)
- Misuse queues (make full or empty)
- Use up RTOS object memory preventing queues from working
- Overflow the stack
- Corrupt important shared data
- Crash the RTOS



Generic Methods to Tolerate Imperfect Software

- Software methods
  - Architecture and Implementation:
    - Structure software, hardware to **isolate** critical functions
    - Provide redundancy in time or space (multicore) for fault detection (and masking)
  - Development Process:
    - Design, implement and test that critical software **much more carefully** than the rest. Refer to Safety-Critical System processes

- Hardware support
  - Use **memory protection unit (MPU)** or MMU to prevent tasks from corrupting each other's memory or control registers
  - Detect when program is out of control, and reset it
    - **Watchdog timer** covered next
    - Safety monitor
  - Use MCU peripheral hardware limitations (e.g. only one write to register allowed after reset)
  - Apply thread privilege levels to prevent execution of critical instructions, access to key control registers
  - Etc.

Examples of Limited Privileges for Cortex-M0+

- From ARM CM0+ Device Generic User Guide
- Types
  - All handler code is privileged
  - Thread code may or may not be privileged
- Unprivileged code
  - Limited use of MSR, MRS
  - Cannot use CPS to mask interrupts
  - Cannot access system timer, NVIC, system control block
  - Possible restricted access to memory or peripherals
  - Cannot write to CONTROL register
- Unprivileged thread uses SVC (service call, SW interrupt) to request execution of privileged software
- Is option for CM0+, but not provided in our KL25Z MCU

NC STATE UNIVERSITY

Protecting the HBLED Buck Driver

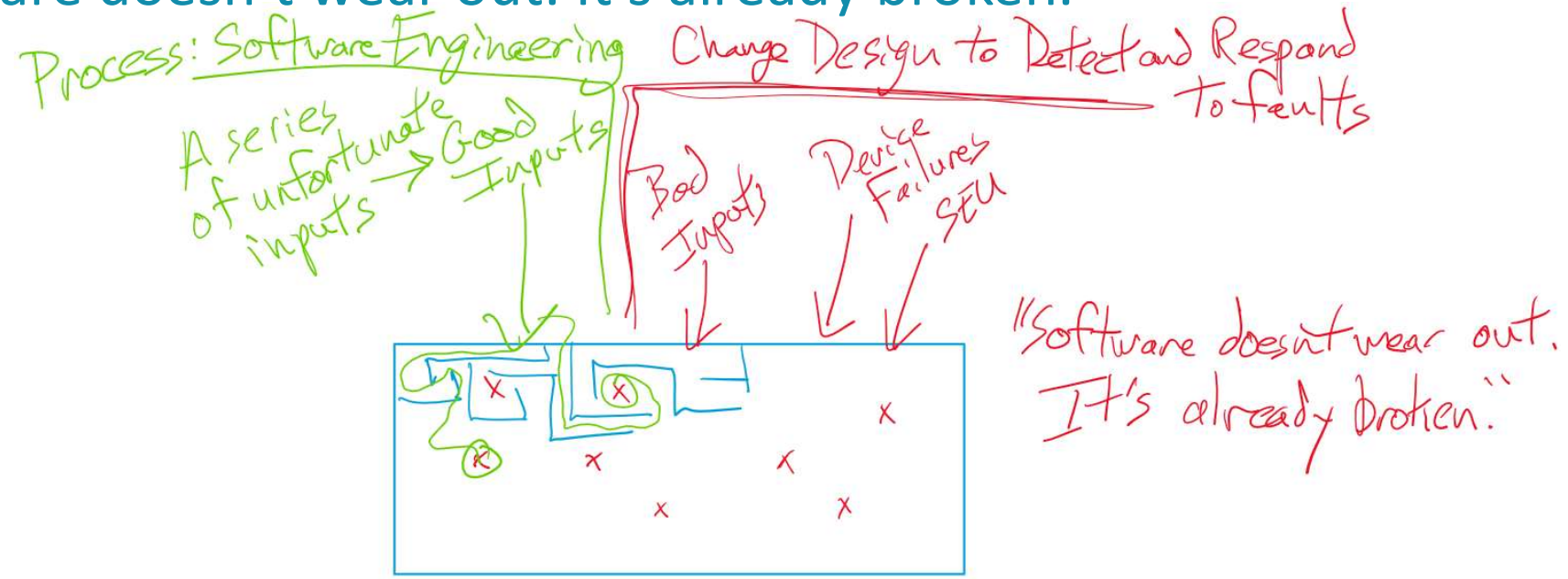
- Reconfigure or disable the ADC or TPM – *scrub system to reconfigure peripherals correctly*
- Overwrite TPM PWM duty cycle
- Reconfigure the port pins. *Scrub system.*
- Disable CPU clock, drop the CPU clock speed so current rises. *Check CPU clock speed periodically*
- Disable all interrupts, keep ADC from generating interrupt, put an infinite loop in a high-priority ISR. *WDT*
- Keep key threads from running (lower their priority, hog processor, take but don't release resources they need). *WDT.*
- Misuse queues (make full or empty)
- Use up RTOS object memory preventing queues from working. *Check return codes*
- Overflow the stack. *Perform stack overflow detection and protection.*
- Corrupt important shared data. *Hide data (make it file local), add access functions, integrity check (CRC/complement), range checking, duty cycle < period? .*
- Crash the RTOS

- *(Temperature sensor)*

NC STATE UNIVERSITY

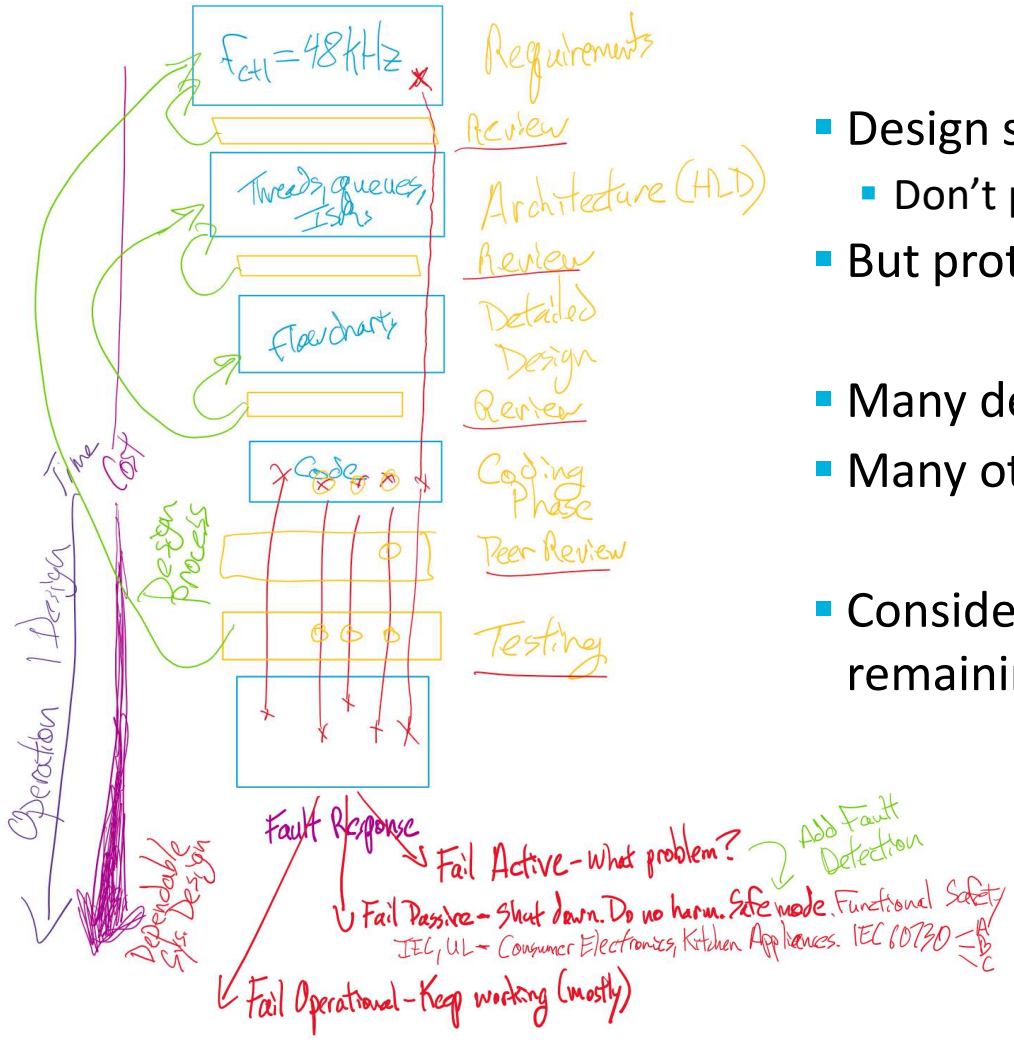
DEVELOPMENT PROCESSES

Software doesn't wear out. It's already broken.



- Systems have latent defects lurking, waiting to be triggered
  - Good but unexpected inputs and system state -> defect activated -> device fails to perform
  - Bad inputs -> defect activated -> device fails to perform
  - Hardware failures -> device fails to perform
- Can eliminate some through better development process
- Address others by designing system to detect and handle faults

Overview of System Development Process So Far



- Design system before coding it
  - Don't paint yourself into a corner
- But prototype high-risk areas

- Many defects can be caught during testing
- Many other defects can be caught by peer review

- Consider how system should detect and handle remaining defects and faults

NC STATE UNIVERSITY

DESIGN IN FEWER DEFECTS

Perform Peer Reviews Upstream of Testing

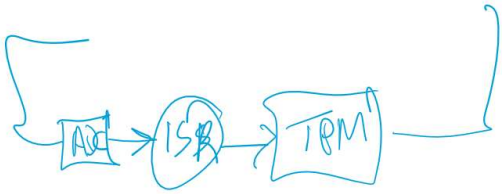
- Peer reviews of code highly effective
  - Good at finding different kinds of bugs than testing is
  - Use both as complements

Formal – Fagan – IBM
1. Distrib. code for review
2. Makes list of issues – bugs, risks, questions
3. Review: discuss issues, solutions
4. Offline Code revision.

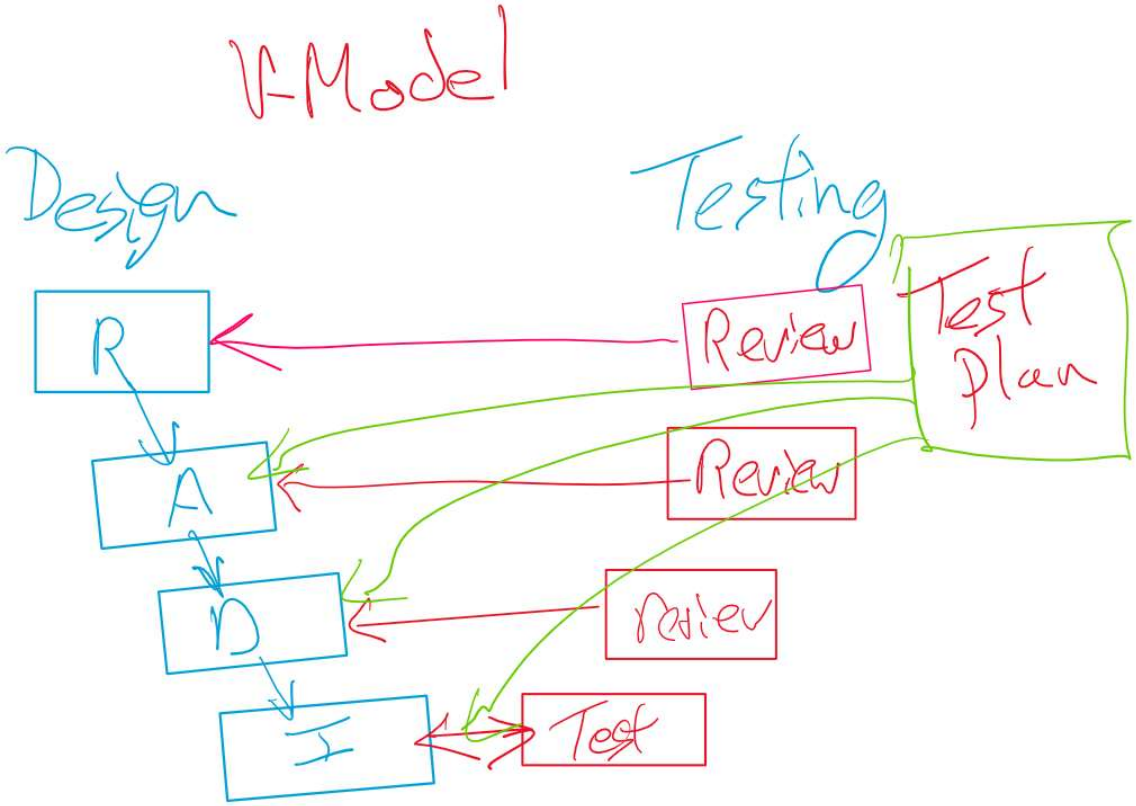
Benefits
1. More careful coding
2. Institutional wisdom

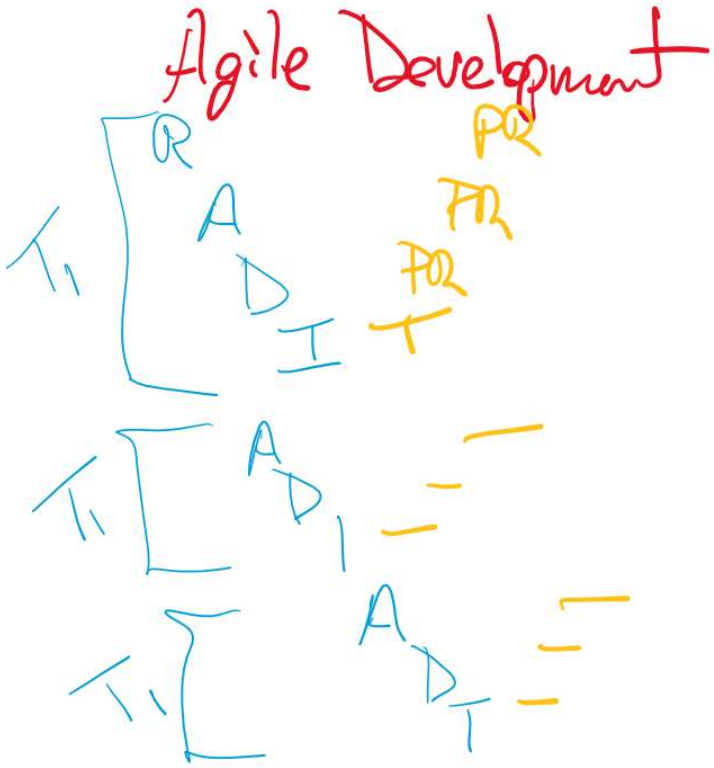
- Peer reviews of Design Documents
  - Test the ideas the system will be built upon
  - Express ideas more concisely tha with code

Peer Review of Design Documents
- vs Code. – Details hurt efficiency.
- Pseudocode
- Flowcharts
- State diagrams
- Tasks + Communications, ISRs, handlers



Software Development Models



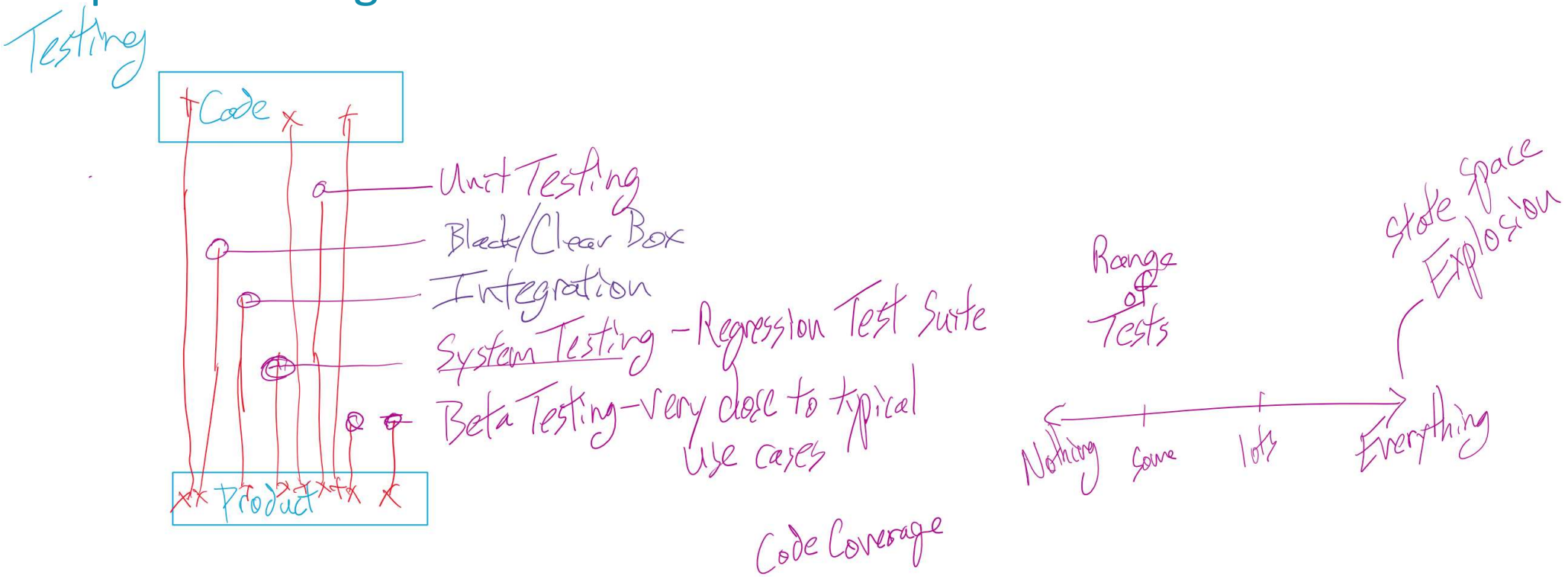


NC STATE UNIVERSITY

TEST TO FIND IMPORTANT DEFECTS EARLIER

NC STATE UNIVERSITY

Improve Testing to Find Relevant Defects Earlier



- Improve testing to identify defects earlier, reducing cost of rework
- Impossible to test all possible inputs and states
- Range of testing methods available. Apply appropriately

NC STATE UNIVERSITY

DEVELOPMENT PROCESSES FOR SAFETY-CRITICAL SYSTEMS

NC STATE UNIVERSITY

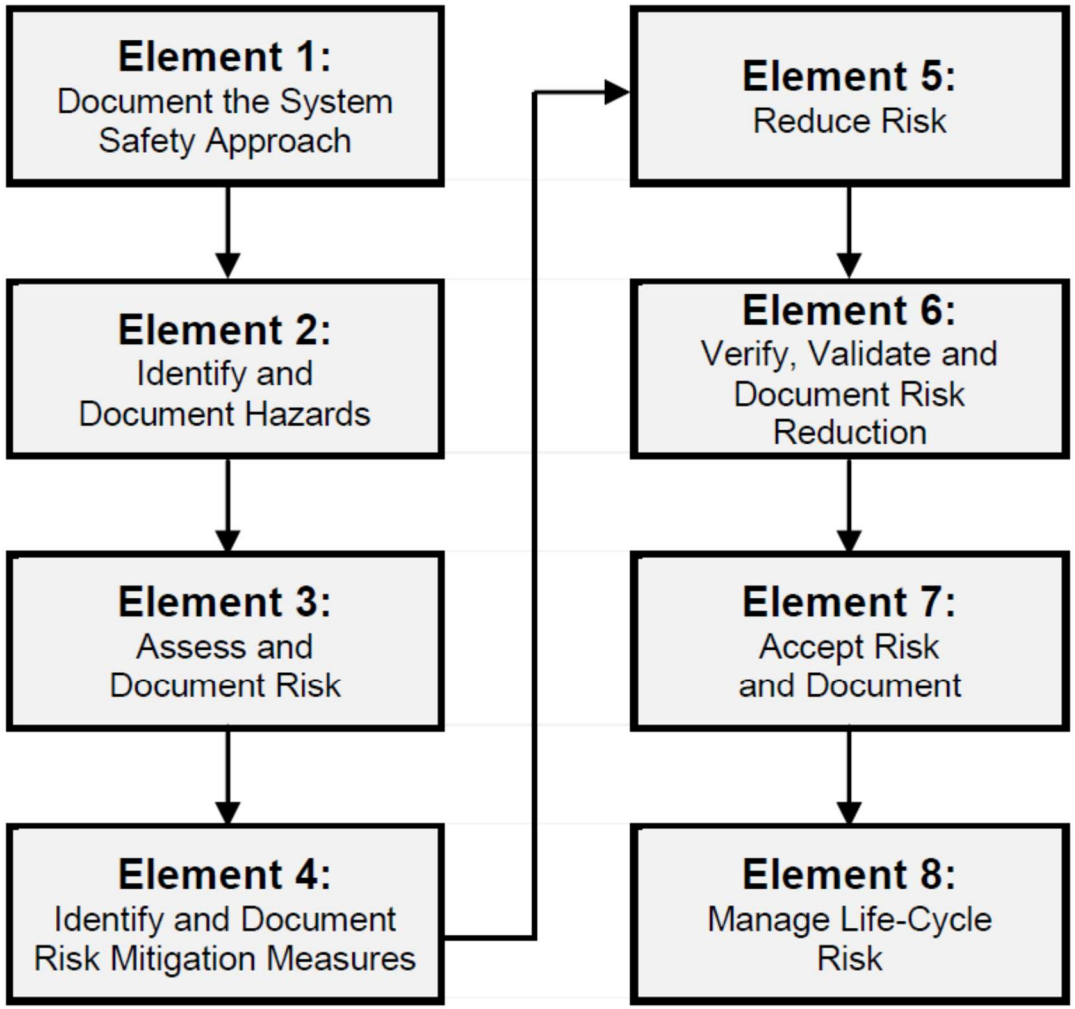
Safety-Critical System Concepts

- Safety-Critical System
  - Failure can lead to injury, death, damage to property or environment (a mishap)
- Example Applications
  - Motor control, aircraft, chemical process, medical electronics, automobile anti-lock braking system
- *Terminology*
  - *Mishap: Event (or chain of events) resulting in unintentional death, injury, damage, etc.*
  - *Hazard: Condition that could lead to a mishap*
  - *Failure: Failing to perform an expected action*
  - *Fault: A defect*
- Types of Computer Use
  - Safety System: passively monitor system, take control when application enters dangerous state
  - Control System: actively control system based on inputs (operator, sensors) and control logic
- Types of Requirements
  - Functional and Operational
    - Maintain oven temperature within 1%
  - Safety-Related
    - Even with failures, don't let temperature go over a specified limit
    - Don't electrocute the user

Activities Required for Achieving System Safety

- Must address system's **life cycle**
  - Design of system, development, testing, production, installation, operation (control, monitoring, diagnostics, repair, upgrade), disposal
- Requires a Systems Engineering management approach (**process**) specific to safety
  - Processes: documentation standards and practices, issue tracking system, system configuration management, etc.
  - Many Other Risks: Changes to design may require additional training for users, employee turn-over, component obsolescence, etc.
- Requires **multidisciplinary** effort
  - Span design from hardware through software through application through …
  - Span entire life cycle
- Apply system safety standards
  - MIL-STD-882E: Standard Practice for System Safety
  - IEC 61508: Functional Safety of electrical/electronic/programmable electronic safety-related systems

MIL-STD-882E **System Safety Process** Elements



- Just a design (hardware and code) is not nearly enough!
- Systems must be developed according to a process
  - Must explain context, the approach used, considerations, etc.
- Development activities are typically audited: “Show me the documentation for all of the steps”

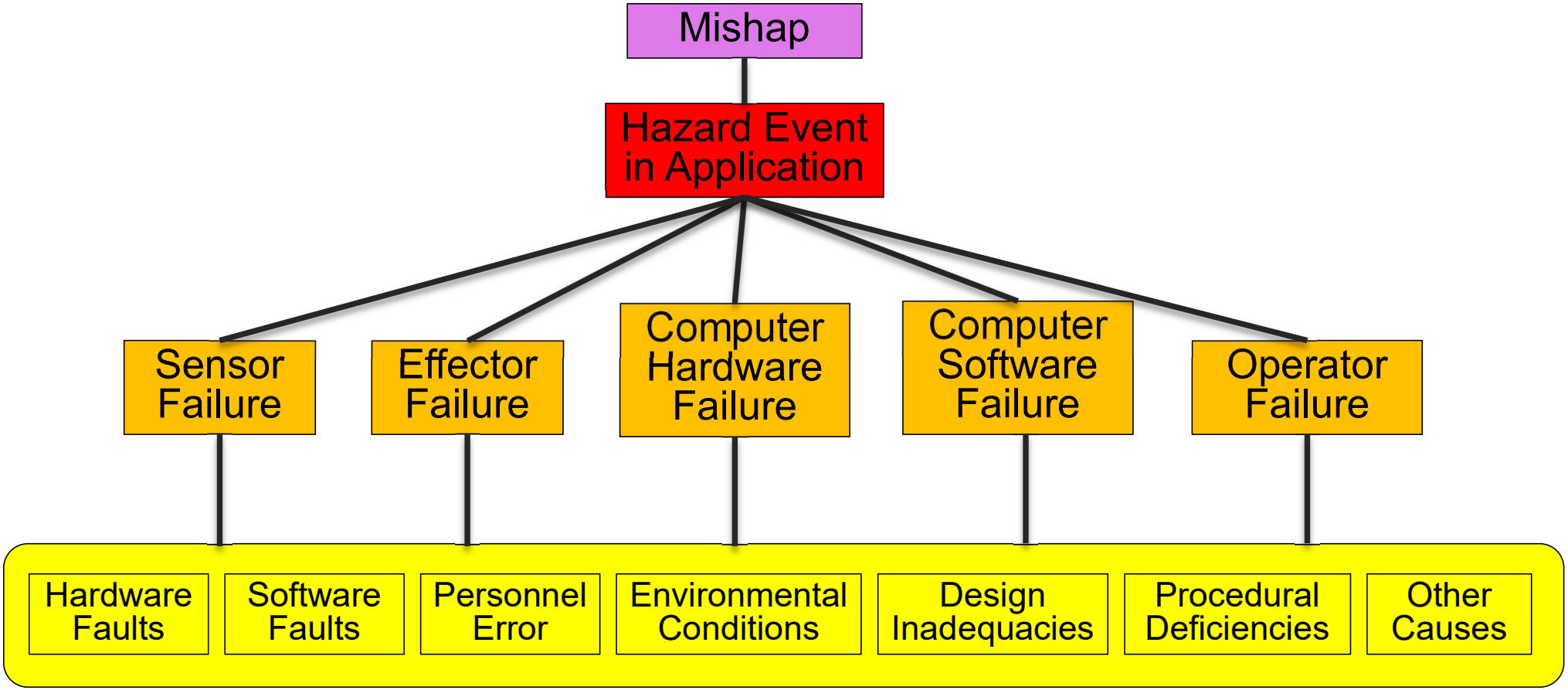
NC STATE UNIVERSITY

Risk Assessment Matrix

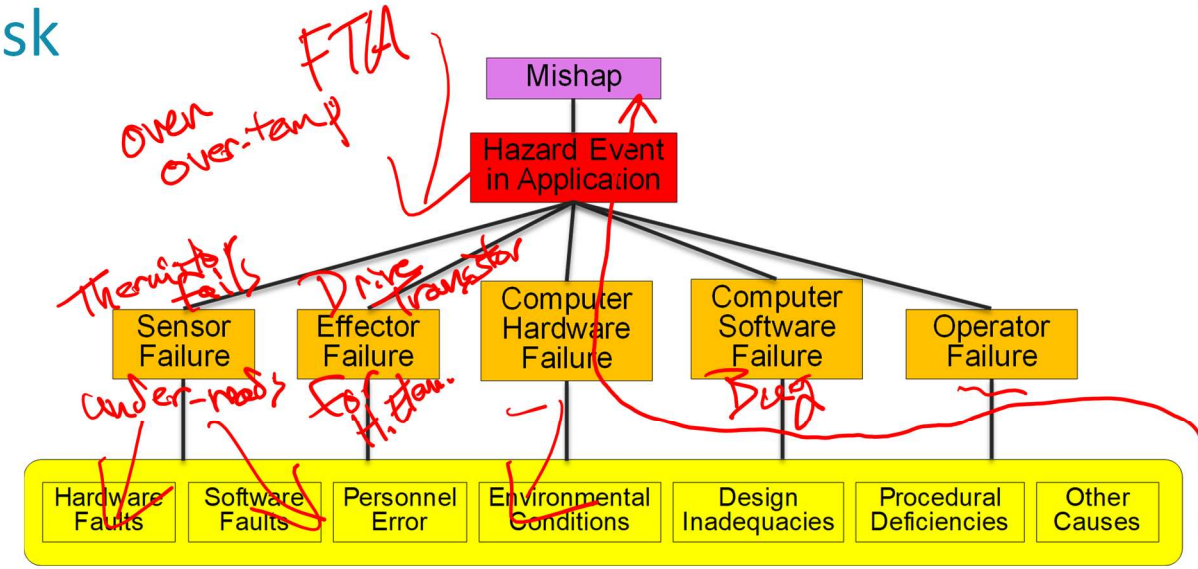
- Depends on severity and probability

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

Mishap Analysis for Basic System

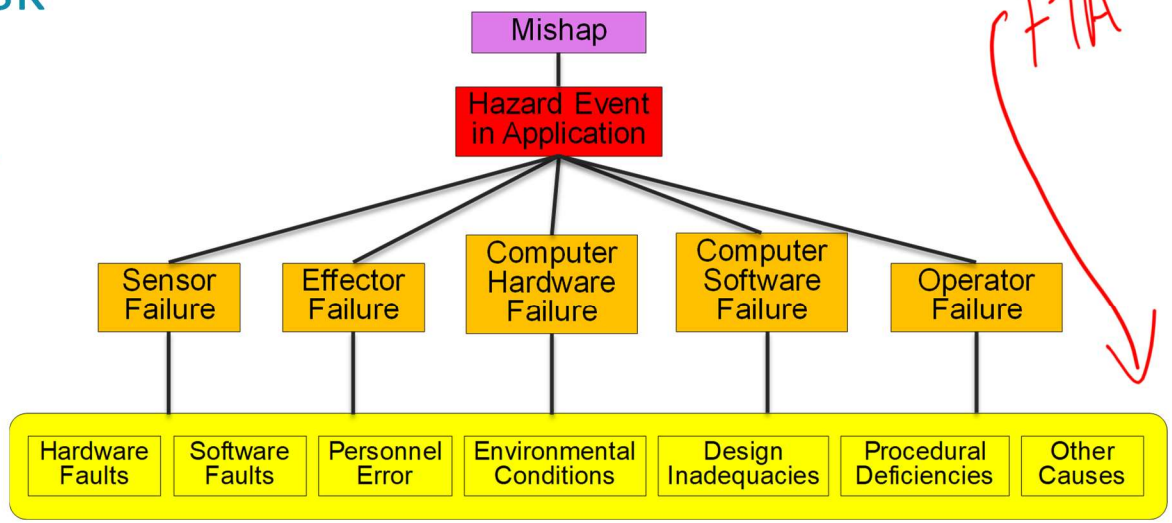


Assessing Risk



- Design evaluation and analysis
  - Failure Modes and Effects Analysis: In what ways can each component fail? What is the impact on the system of that component failing that way?
  - Failure Modes and Effects Testing: Inject component failures and evaluate system responses, confirming design mitigates failures.
  - Fault Tree Analysis: What could cause each mishap?
- Risk Analysis
  - Perform Fault Tree Analysis to identify basic component failure events
  - Calculate probability of all these failure events
  - Combine probabilities to determine mishap probability

Assessing Risk



FMEA

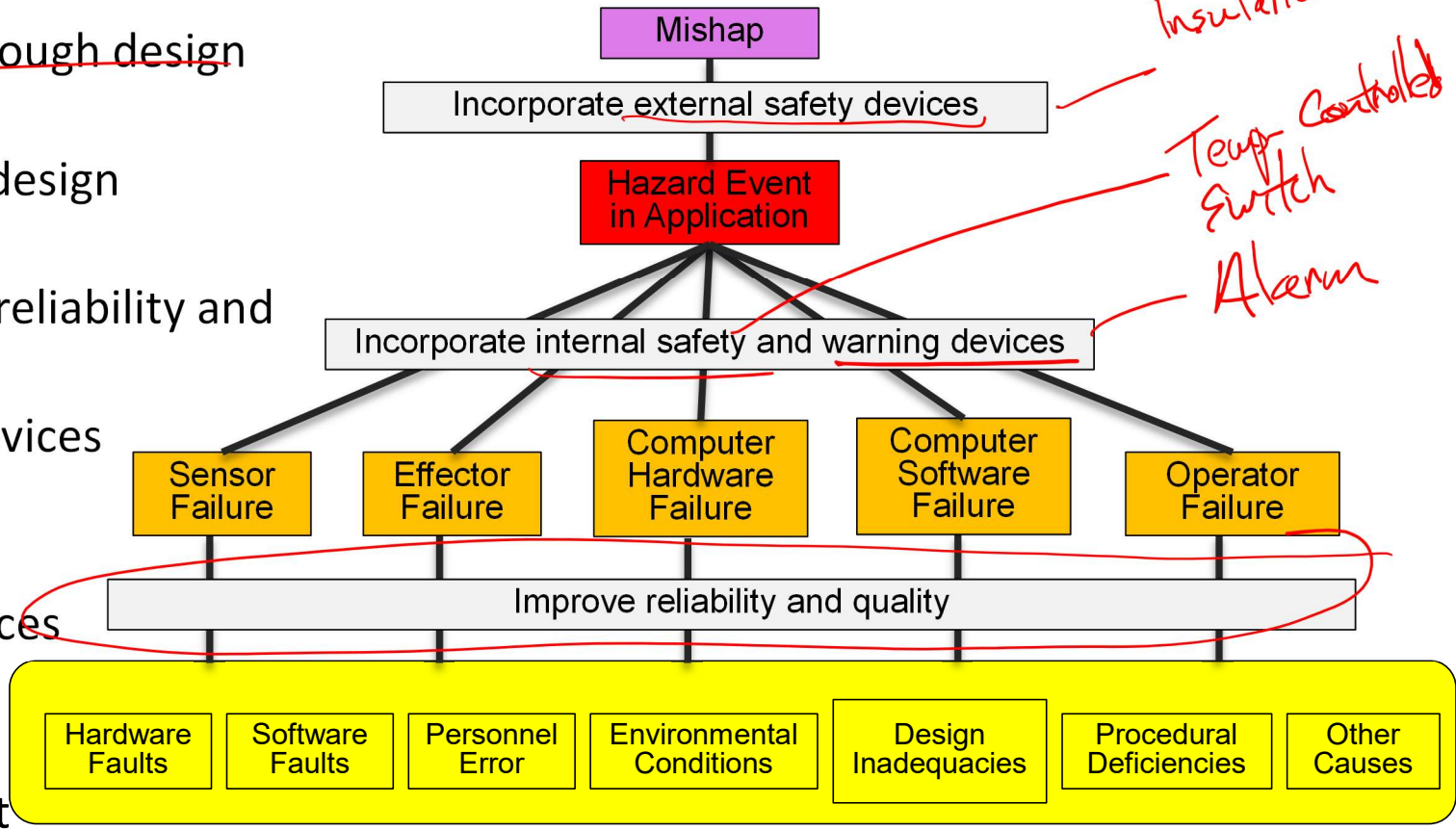
FTA

Burn out HB LED
Too much current, temperature
Duty Cycle too high
loose wire
PM fault
Software Defect

- Design evaluation and analysis
  - Failure Modes and Effects Analysis: In what ways can each component fail? What is the impact on the system of that component failing that way?
  - Failure Modes and Effects Testing: Inject component failures and evaluate system responses, confirming design mitigates failures.
  - Fault Tree Analysis: What could cause each mishap?
- Risk Analysis
  - Perform Fault Tree Analysis to identify basic component failure events
  - Calculate probability of all these failure events
  - Combine probabilities to determine mishap probability

Methods to Mitigate Mishap Risk (In Preferred Order)

1. Eliminate hazards through design selection
2. Reduce risk through design alteration
3. Improve component reliability and quality
4. Incorporate safety devices
   - External
   - Internal
5. Provide warning devices
6. Develop procedures, training and personal protective equipment



MIL-STD-882E on Software

4.4 Software contribution to system risk.

The assessment of risk for software, and consequently software-controlled or software-intensive systems, cannot rely solely on the risk severity and probability.

Determining the probability of failure of a single software function is difficult at best and cannot be based on historical data.

Software is generally application-specific and reliability parameters associated with it cannot be estimated in the same manner as hardware.

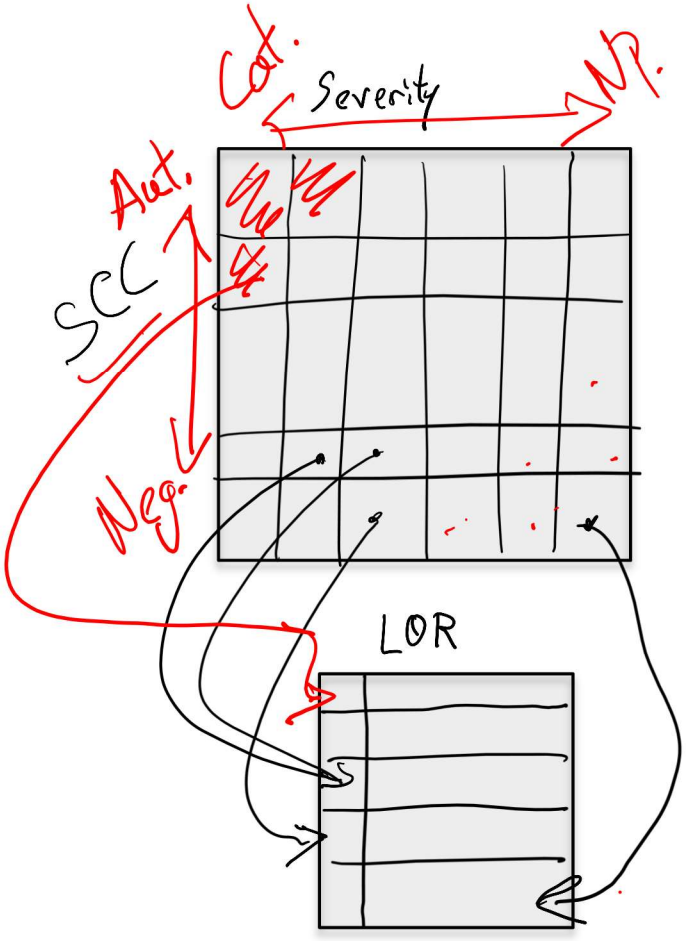
Therefore, another approach shall be used for the assessment of software's contributions to system risk that considers the potential risk severity and the degree of control that software exercises over the hardware.

Process Requires Safety Decisions to be Documented and Justified

RELATIONSHIP BETWEEN SwCI, RISK LEVEL, LOR Tasks, AND RISK		
Software Criticality Index (SwCI)	**Risk Level**	**Software LOR Tasks and Risk Assessment/Acceptance**
SwCI 1	**High**	• If SwCI 1 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as HIGH and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 1 LOR tasks or prepare a formal risk assessment for acceptance of a HIGH risk.
SwCI 2	**Serious**	• If SwCI 2 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as SERIOUS and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 2 LOR tasks or prepare a formal risk assessment for acceptance of a SERIOUS risk.
SwCI 3	**Medium**	• If SwCI 3 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as MEDIUM and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 3 LOR tasks or prepare a formal risk assessment for acceptance of a MEDIUM risk.
SwCI 4	**Low**	• If SwCI 4 LOR tasks are unspecified or incomplete, the contributions to system risk will be documented as LOW and provided to the PM for decision. The PM shall document the decision of whether to expend the resources required to implement SwCI 4 LOR tasks or prepare a formal risk assessment for acceptance of a LOW risk.
SwCI 5	**Not Safety**	• No safety-specific analyses or testing is required.

4.4.1 Software Assessments

- Software Control Categories (SCC)
  - Describes how much of the system the software controls
- Software Safety Criticality Matrix (SSCM)
  - Describes how critical the software is based on the severity of a mishap and the SCC
  - Uses a number: Software Criticality Index (SwCI)
- Level of Rigor Tasks (LoR)
  - Indicates which tasks must be performed for this SwCI



NC STATE UNIVERSITY

Software Control Categories

Level	Name	Description (Paraphrased)
1	**Autonomous (AT)**	**Autonomous control authority over potentially safety-significant components without the possibility of predetermined safe detection and intervention** by a control entity to prevent mishap or hazard.
2	**Semi-Autonomous (SAT)**	**Control authority over potentially safety-significant components, allowing time for predetermined safe detection and intervention** by independent safety mechanisms to mitigate or control the mishap or hazard. Software item that displays **safety-significant information requiring immediate operator entity to execute a predetermined action for mitigation or control over a mishap or hazard**. Software exception, failure, fault, or delay will allow mishap occurrence.
3	**Redundant Fault Tolerant (RFT)**	Software functionality that **issues commands over safety-significant hardware components requiring a control entity to complete the command function**. The system detection and functional reaction includes redundant, independent fault tolerant mechanisms for each defined hazardous condition. Software that **generates information of a safety-critical nature used to make critical decisions**. The system includes several redundant, independent fault tolerant mechanisms for each hazardous condition, detection and display.
4	**Influential**	Software **generates information of a safety-related nature** used to make decisions by the operator, but **does not require operator action to avoid a mishap**.
5	**No Safety Impact (NSI)**	No command or control authority over safety-significant hardware components, does not provide safety-significant information. Software does not provide safety-significant or time sensitive data or information that requires control entity interaction. Software does not transport or resolve communication of safety-significant or time sensitive data.

SSCM and Level of Rigor Tasks

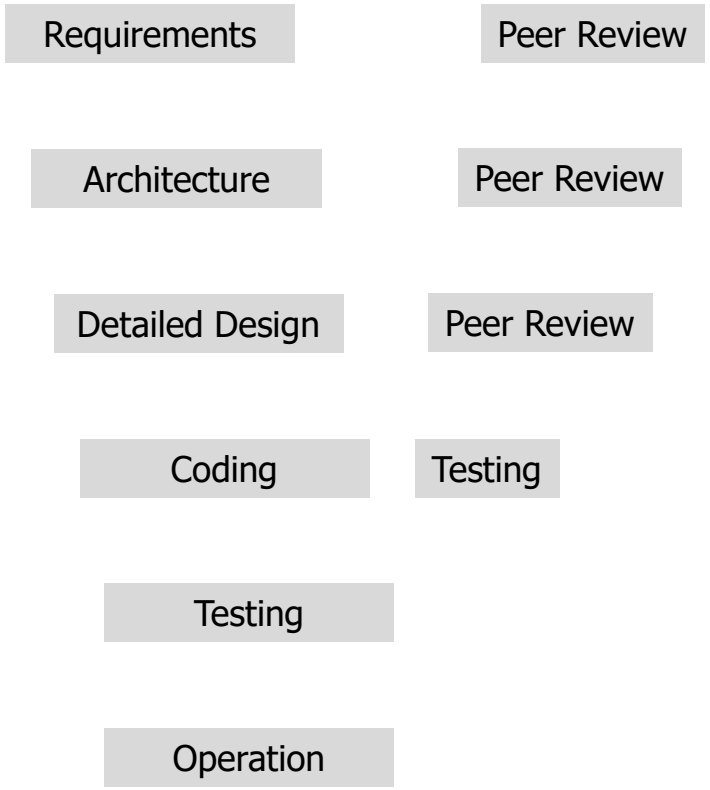
SOFTWARE SAFETY CRITICALITY MATRIX				
	SEVERITY CATEGORY			
SOFTWARE CONTROL CATEGORY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
1	SwCI 1	SwCI 1	SwCI 3	SwCI 4
2	SwCI 1	SwCI 2	SwCI 3	SwCI 4
3	SwCI 2	SwCI 3	SwCI 4	SwCI 4
4	SwCI 3	SwCI 4	SwCI 4	SwCI 4
5	SwCI 5	SwCI 5	SwCI 5	SwCI 5

Level of Rigor Tasks					
Software Criticality Index	Perform analysis of …				Conduct Safety-Specific Testing
	Requirements	Architecture	Design	Code	
1	●	●	●	●	In-depth
2	●	●	●		In-depth
3	●	●			In-depth
4					●
5					

NC STATE UNIVERSITY

APPENDIX

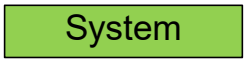
Mapping Solutions onto the System Development Process



- Design the fault out of the system
- Don't design-in or code-in the fault
  - Use a better development process
  - Prototype or model high-risk areas
  - During development, find the fault as early as possible
    - Test early and often
    - Use peer reviews for non-functional items (architectures, etc.)
- Accept the fault may happen, so design system to tolerate it during operation

Causes of Embedded System Failures

- Latent faults
  - The system was designed and built that way
- Environmental factors
  - Device failures
  - Noise, single event upsets
  - Hacker attacks
  - New unexpected inputs/conditions



NC STATE UNIVERSITY

Definitions

- Terms
  - Safety
    - System does not cause any harm
  - Reliability
    - Probability that item will operate reliably for a specified amount of time
  - Availability
    - Probability that item will operate correctly at a given time

NC STATE UNIVERSITY

Fault Detection Methods