

Software System Safety

Nancy G. Leveson

© Copyright by the author, July 2005.

Outline of the Class

- 1: Understanding the Problem
 - Accident Causes
 - Accident Models
 - Computers and Risks
 - Safety vs. Reliability
- 2: Approaches to Safety Engineering
- 3: Introduction to Systems Theory
- 4: STAMP: A New Accident Model
- 5: The System Safety Process

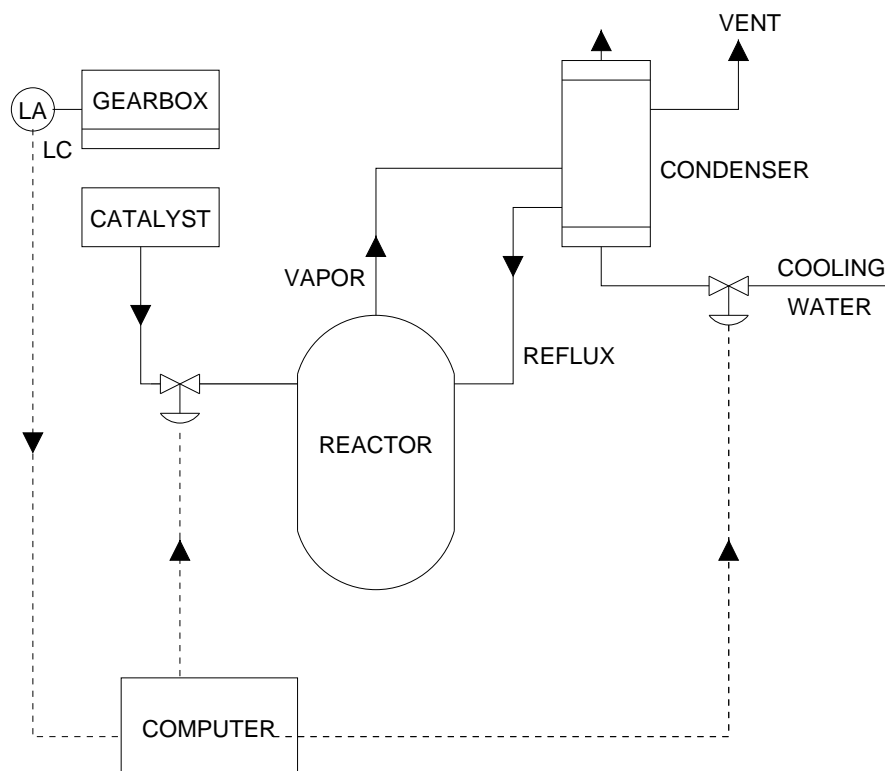
Outline (Con't)

- 6: Hazard Analysis for Software-Intensive Systems
- 7: Requirements Analysis
- 8: Design for Safety
- 9: Human-Machine Interaction and Safety
- 10: Testing and Assurance
- 11: Management and Organizational Issues
- 12: Summary and Conclusions

The Problem

The first step in solving any problem is to understand it. We often propose solutions to problems that we do not understand and then are surprised when the solutions fail to have the anticipated effect.

Accident with No Component Failures



Types of Accidents

- Component Failure Accidents
 - Single or multiple component failures
 - Usually assume random failure
- System Accidents
 - Arise in interactions among components
 - No components may have "failed"
 - Caused by interactive complexity and tight coupling
 - Exacerbated by the introduction of computers.

Interactive Complexity

- Complexity is a moving target
- The underlying factor is intellectual manageability
 1. A "simple" system has a small number of unknowns in its interactions within the system and with its environment.
 2. A system is intellectually unmanageable when the level of interactions reaches the point where they cannot be thoroughly
 - planned
 - understood
 - anticipated
 - guarded against
 3. Introducing new technology introduces unknowns and even "unk-unks."

Tight Coupling

- Tightly coupled system is one that is highly interdependent:
 - Each part linked to many other parts.
Failure or unplanned behavior in one can rapidly affect status of others.
 - Processes are time-dependent and cannot wait.
Little slack in system
 - Sequences are invariant, only one way to reach a goal.
- System accidents are caused by unplanned interactions.
- Coupling creates increased number of interfaces and potential interactions.

Other Types of Complexity

Decompositional complexity

- Structural decomposition not consistent with functional decomposition

Non-linear complexity

- Cause and effect not related in obvious way

Dynamic complexity:

- Related to changes over time

Accident Causes

My company has had a safety program for 150 years. The program was instituted as a result of a French law requiring an explosives manufacturer to live on the premises with his family.

Crawford Greenwalt
(former president of Dupont)

Most accidents are not the result of unknown scientific principles, but rather of a failure to apply well-known, standard engineering practices.

Trevor Kletz

Causality

- Accident causes are often oversimplified:

The vessel Baltic Star, registered in Panama, ran aground at full speed on the shore of an island in the Stockholm waters on account of thick fog. One of the boilers had broken down, the steering system reacted only slowly, the compass was maladjusted, the captain had gone down into the ship to telephone, the lookout man on the prow took a coffee break, and the pilot had given an erroneous order in English to the sailor who was tending the rudder. The latter was hard of hearing and understood only Greek.

LeMonde

- Larger organizational and economic factors?

Issues in Causality

- Filtering and subjectivity in accident reports
- Root cause seduction
 - Idea of a singular cause is satisfying to our desire for certainty and control.
 - Leads to fixing symptoms
- The "fixing" orientation
 - Well understood causes given more attention
 - Component failure
 - Operator error
 - Tend to look for linear cause–effect relationships
 - Makes it easier to select corrective actions (a "fix")

NASA Procedures and Guidelines: NPG 8621 Draft 1

Root Cause:

"Along a chain of events leading to a mishap, the first causal action or failure to act that could have been controlled systematically either by policy/practice/procedure or individual adherence to policy/practice/procedure."

Contributing Cause:

"A factor, event, or circumstance that led directly or indirectly to the dominant root cause, or which contributed to the severity of the mishap."

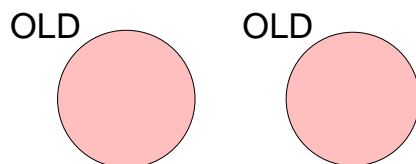
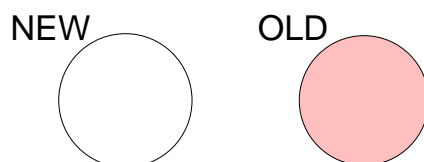
Do Operators Cause Most Accidents?

- Data may be biased and incomplete
- Positive actions usually not recorded
- Blame may be based on premise that operators can overcome every emergency
- Operators often have to intervene at the limits.
- Hindsight is always 20/20
- Separating operator error from design error is difficult and perhaps impossible.

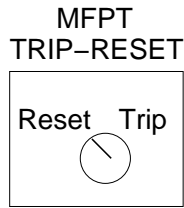
Example accidents from chemical plants:



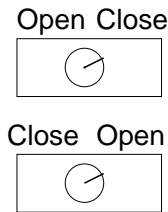
Operator told to fix pump 7.



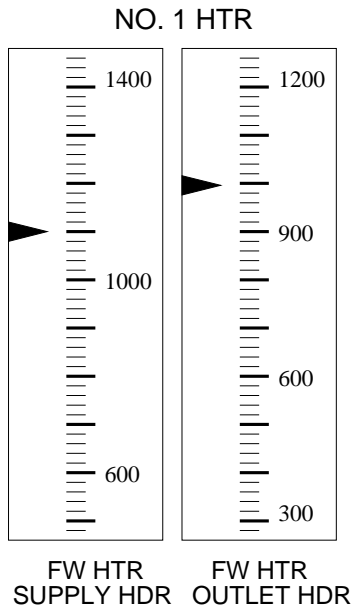
Operator told to replace crystallizer A



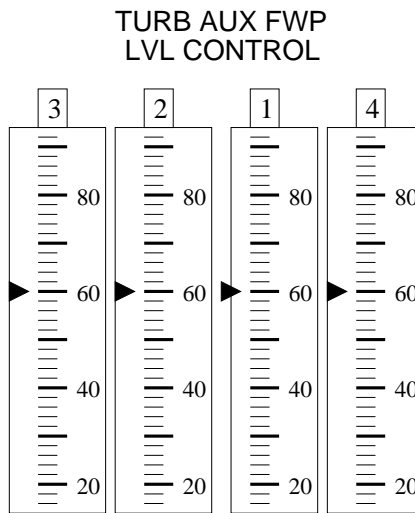
a. Note reversal of trip-reset positions



b. Another Inconsistency



c. Heater pressure gauges. A hurried operator under stress might believe the outlet pressure is higher than the supply, even though it is lower.



d. A strange way to count.

A-320 accident while landing at Warsaw:

Blamed on pilots for landing too fast.

Was it that simple?

- Pilots told to expect windshear. In response, landed faster than normal to give aircraft extra stability and lift.
 - Meteorological information out of date -- no windshear by time pilots landed.
 - Polish government's meteorologist supposedly in toilet at time of landing.
- Thin film of water on runway that had not been cleared.
 - Wheels aquaplaned, skimming surface, without gaining enough rotary speed to tell computer braking systems that aircraft was landing.
 - Computers refused to allow pilots to use aircraft's braking systems. So did not work until too late.
- Still would not have been catastrophic if had not built a high bank at end of runway.
 - Aircraft crashed into bank and broke up.

Blaming pilots turns attention away from:

- Why pilots were given out-of-date weather information
- Design of computer-based braking system
 - Ignored pilots commands
 - Pilots not able to apply braking systems manually
 - Who has final authority?
- Why allowed to land with water on runway
- Why decision made to build a bank at end of runway

Cited probable causes of Cali American Airlines crash:

- Flightcrew's failure to adequately plan and execute the approach to runway 19 at Cali and their inadequate use of automation.
- Failure of flight crew to discontinue the approach into Cali, despite numerous cues alerting them of the inadvisability of continuing the approach.
- Lack of situational awareness of the flightcrew regarding vertical navigation, proximity to terrain, and the relative location of critical radio aids.
- Failure of the flightcrew to revert to basic radio navigation at the time when the FMS-assisted navigation became confusing and demanded an excessive workload in a critical phase of flight.

Exxon-Valdez

- Shortly after midnight, March 24, 1989, tanker Exxon Valdez ran aground on Bligh Reef (Alaska).
 - 11 million gallons of crude oil released.
 - Over 1500 miles of shoreline polluted.
- Company and government put responsibility on tanker Captain.
- Captain Hazelwood was disciplined and fired.
- Was he to "blame"?
 - State-of-the-art iceberg monitoring equipment promised by oil industry, but never installed. Exxon Valdez traveling outside normal sea lane in order to avoid icebergs thought to be in the area.
 - Radar station in city of Valdez, which was responsible for monitoring the location of tanker traffic in Prince William Sound, had replaced its radar with much less powerful equipment. Location of tankers near Bligh Reef could not be monitored with this equipment.

- Congressional approval of Alaska oil pipeline and tanker transport network included an agreement by oil corporations to build and use double-hulled tankers. Exxon Valdez did not have a double hull.
- Crew fatigue was typical on tankers.
 - In 1977, average oil tanker operating out of Valdez had a crew of 40 people. By 1989, crew size had been cut in half.
 - Crews routinely worked 12–14 hour shifts, plus extensive overtime.
 - Exxon Valdez had arrived in port at 11 pm the night before. The crew rushed to get the tanker loaded for departure the next evening.
- Coast Guard at Valdez assigned to conduct safety inspections of tankers. It did not perform these inspections. It's staff had been cut by one-third.

- Tanker crews relied on the Coast Guard to plot their position continually.
 - Coast Guard operating manual required this.
 - Practice of tracking ships all the way out to Bligh reef had been discontinued.
 - Tanker crews were never informed of the change.
- Spill response teams and equipment were not readily available. Seriously impaired attempts to contain and recover the spilled oil.
- Summary:
 - Safeguards designed to avoid and mitigate effects of an oil were not in place or were not operational.
 - By focusing exclusively on blame, the opportunity to learn from mistakes is lost.
- Postscript: Captain Hazelwood was tried for being drunk the night the Exxon Valdez went aground. He was found "not guilty."

Accident Models

Models

- Provide a means for
 - Understanding phenomena
 - Recording that understanding so can communicate to others
- All models are abstractions
 - Omit assumed irrelevant details
 - Focus on features of phenomenon assumed most relevant
 - Selection process usually arbitrary and dependent on choice of modeler
 - Selection is critical in determining usefulness and accuracy of model

Accident models

- Underlie all attempts to engineer for safety.
 - Used to explain how accidents occur.
 - Assume common patterns in accidents; not just random events
 - Imposing pattern on accidents influences factors considered in safety analysis
 - Model may act as filter and bias toward considering only some events and conditions
- or
- May force consideration of factors often omitted.

Accident models (2)

- Forms basis for:
 - Investigating and analyzing accidents
 - Preventing accidents
 - Hazard analysis
 - Design for safety
 - Assessing risk (determining whether systems are suitable for use)
 - Performance auditing and defining safety metrics
- So influences causes identified, countermeasures taken, and risk evaluation
- May not be aware using model, but always exists

Chain-of-Events Models

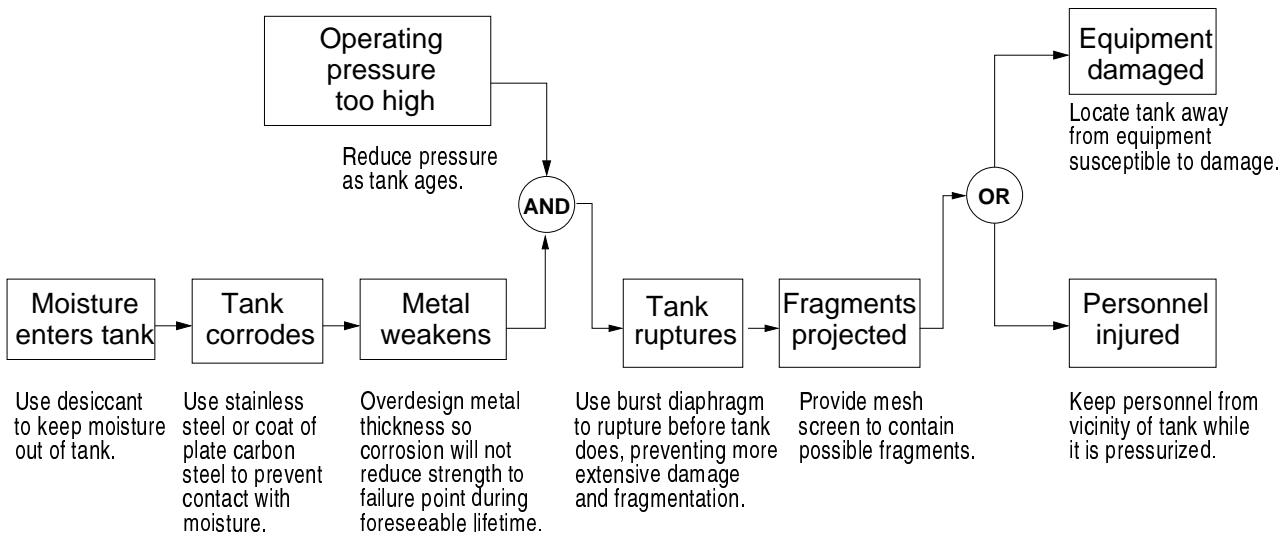
- Explain accidents in terms of multiple events, sequenced as a forward chain over time.
 - Simple, direct relationships between events in chain
 - Contrapositive (if A hadn't occurred, then B would not have)
- Events almost always involve component failure, human error, or energy-related event
- Form the basis of most safety-engineering and reliability engineering analysis:

e.g., Fault Tree Analysis, Probabilistic Risk Assessment, FMEA, Event Trees

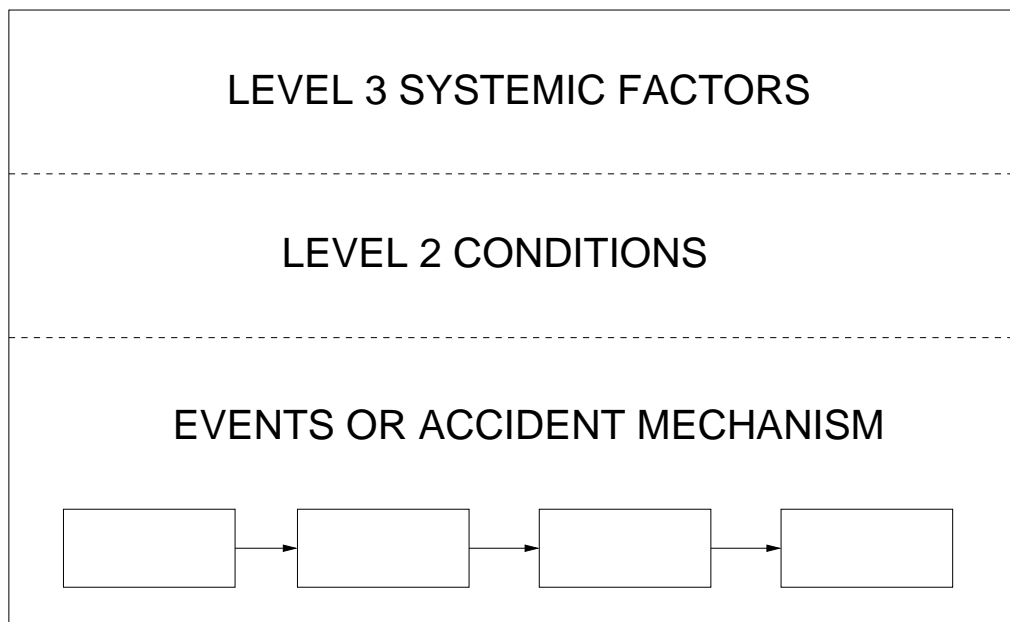
and design:

e.g., redundancy, overdesign, safety margins, ...

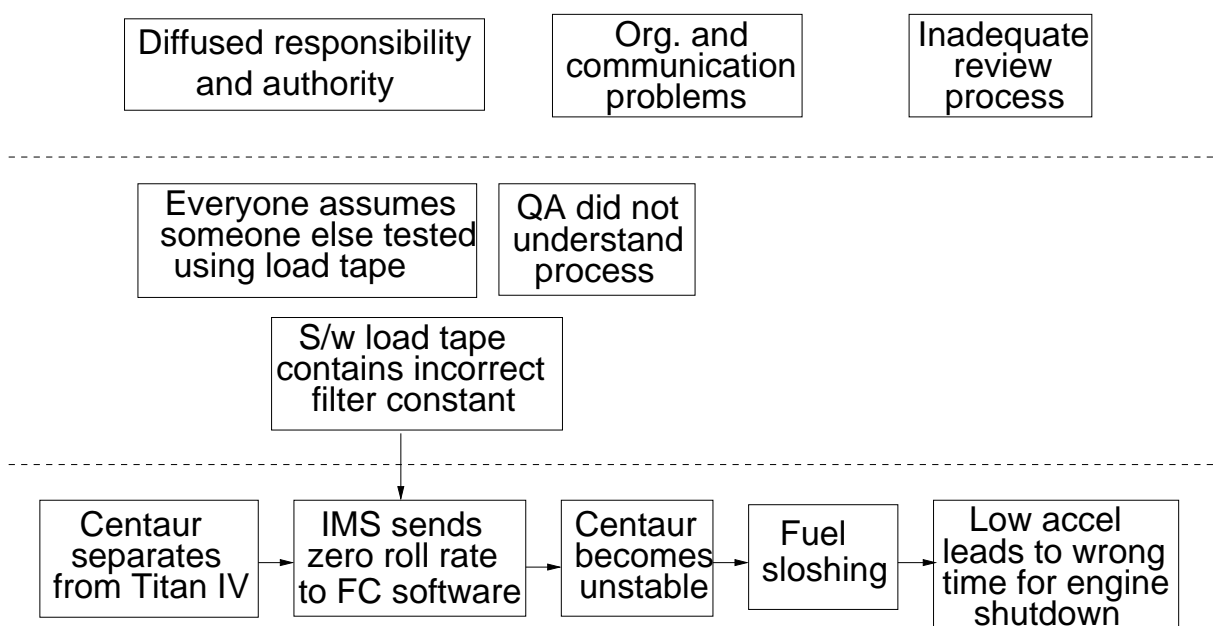
Chain-of-Events Example



Hierarchical Models



Hierarchical Analysis Example



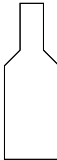


Limitations of Event Chain Models

- Selecting events
 - Subjective except for physical events immediately preceding or directly involved in accident
 - Root cause dependent on stopping rule (difficult to go "through" operators)

Possible Bhopal event chain:

E1: Worker washes pipes without inserting slip blind
 E2: Water leaks into MIC tank
 E3: Explosion occurs
 E4: Relief valve opens
 E5: MIC vented into air
 E6: Wind carries MIC into populated area around plant

Bhopal

<p>Management: Blamed on sabotage worker not properly shutting valve</p>	<p>(personnel problems cuts in training and maintenance shift replacement policies)</p>
<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; border-radius: 10px; padding: 5px; margin-right: 10px;">Tank 610</div> <div> <p>Pressure in tank 610 builds up due to chemical reaction. MIC vapor escapes, rupturing safety valve.</p> </div> </div>	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; border-radius: 10px; padding: 5px; margin-right: 10px;">Tank 619</div> <div> <p>Tank 619 was empty but nobody opened valves to relieve pressure. Instruments indicated tank was not empty.</p> </div> </div>
<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Refrigeration Systems</div> <div> <p>Turned off so tank 610 could not be cooled down to slow reaction.</p> </div> </div>	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">  </div> <div> <p>Vent gas scrubber supposed to spray caustic soda on escaping vapors to neutralize them. Scrubber shut down for maintenance. Design inadequate anyway.</p> </div> </div>
<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">  </div> <div> <p>Water curtain, which could have neutralized some MIC, designed to reach height of 40 to 50 feet. MIC vapor vented 100 feet above ground.</p> </div> </div>	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">  </div> <div> <p>Flare tower could not be used because a length of pipe was corroded and had not been replaced. Design inadequate anyway.</p> </div> </div>

PLUS: Instruments did not work
Poor emergency procedures

Limitations of Event Chain Models (2)

- Selecting conditions
 - Links between events, chosen to explain them, are subjective

Cali AA B-757 accident:

- E1: Pilot asks for clearance to take ROZO approach
- E2: Pilot types R into the FMS

What is the link between these two events?

- Pilot Error?*
- Crew Procedure Error?*
- Approach Chart and FMS inconsistencies?*
- American Airlines training deficiency?*
- Manufacturer deficiency?*
- International standards deficiency?*

Selection of linking condition will greatly influence accident cause identified

Limitations of Event Chain Models (3)

- Selecting countermeasures
 - Leads to overreliance on redundancy
- Risk assessment
 - Usually assumes independence between events
 - Events chosen will affect accuracy but subjective
 - Usually concentrates on failure events
- Treating events and conditions as causes
 - Can miss systemic causes
 - Root cause analysis limited if use event chain models
(fault trees, fishbone diagrams, barrier analysis, etc.)

Risk Measurement

- Risk = f (likelihood, severity)
- Impossible to measure risk accurately.
- Instead, use risk assessment:
 - Accuracy of such assessments is controversial.

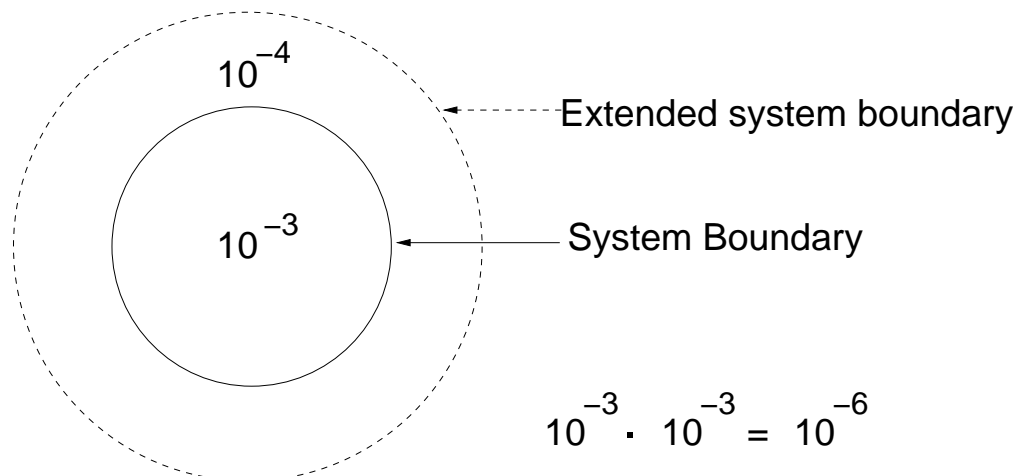
“To avoid paralysis resulting from waiting for definitive data, we assume we have greater knowledge than scientists actually possess and make decisions based on those assumptions.”

William Ruckleshaus

- Cannot evaluate probability of very rare events directly.
- So use models of the interaction of events that can lead to an accident.

Misinterpreting Risk

Risk assessments can easily be misinterpreted:



Risk Modeling

- In practice, models only include events that can be measured.
- Most causal factors involved in major accidents are unmeasurable.
- Unmeasurable factors tend to be ignored or forgotten.
- Can we measure software? (what does it mean to measure design?)

*Risk assessment data can be like the captured spy;
if you torture it long enough, it will tell you anything
you want to know.*

William Ruckelshaus
Risk in a Free Society

Limitations of Event Chain models (4)

- Social and organizational factors in accidents.

*Underlying every technology is at least one basic science,
although the technology may be well developed long before the
science emerges. Overlying every technical or civil system is
a social system that provides purpose, goals, and decision criteria.*

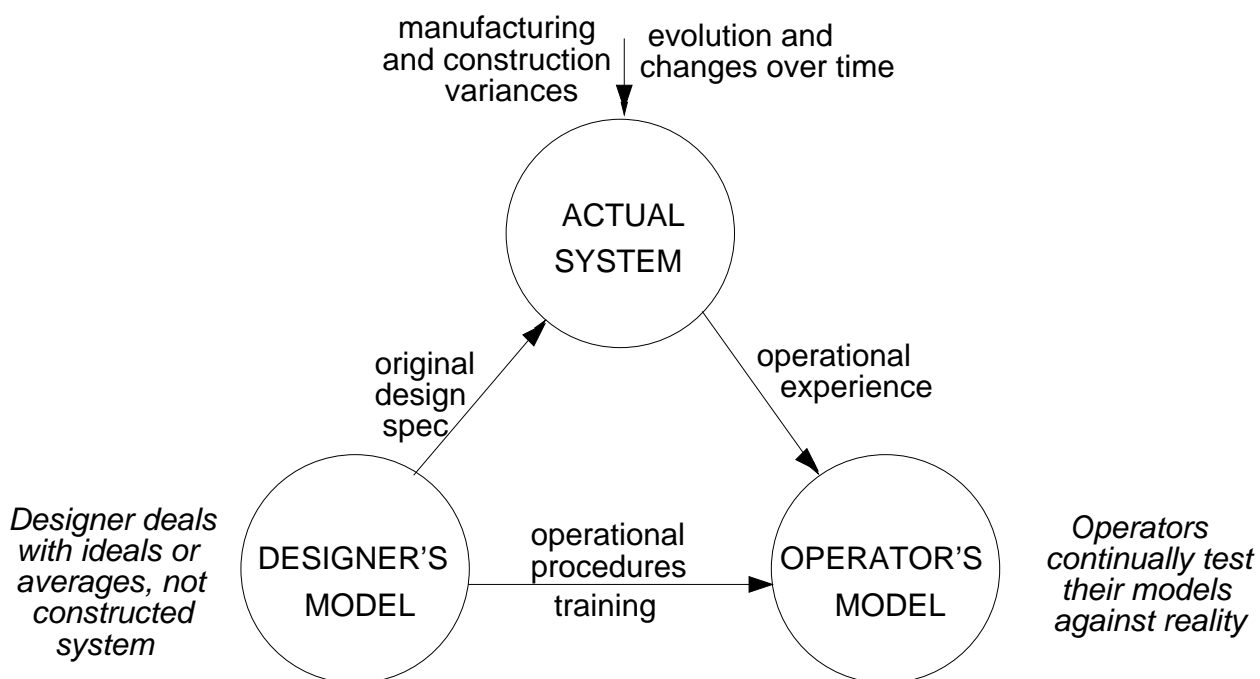
Ralph Miles Jr.

- Models need to include the social system as well as the technology and its underlying science.
- System accidents
- Software

Limitations of Event Chain Models (5)

- Human error
 - Define as deviation from normative procedure, but operators always deviate from standard procedures.
 - normative procedures vs. effective procedures
 - sometimes violation of rules has prevented accidents
 - Cannot effectively model human behavior by decomposing it into individual decisions and acts and studying it in isolation from the
 - physical and social context
 - value system in which takes place
 - dynamic work process
 - Less successful actions are natural part of search by operator for optimal performance

Mental Models



System changes and so must operator's model

Limitations of Event Chain Models (6)

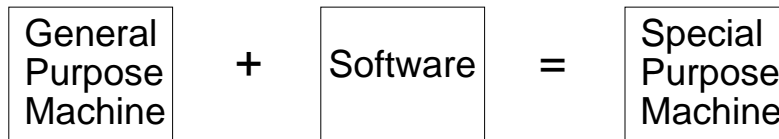
- Adaptation
 - Systems are continually changing
 - Systems and organizations migrate toward accidents (states of high risk) under cost and productivity pressures in an aggressive, competitive environment.

Computers and Risk

We seem not to trust one another as much as would be desirable. In lieu of trusting each other, are we putting too much trust in our technology? . . . Perhaps we are not educating our children sufficiently well to understand the reasonable uses and limits of technology.

Thomas B. Sheridan

The Computer Revolution



- Software is simply the design of a machine abstracted from its physical realization.
- Machines that were physically impossible or impractical to build become feasible.
- Design can be changed without retooling or manufacturing.
- Can concentrate on steps to be achieved without worrying about how steps will be realized physically.

Advantages = Disadvantages

- Computer so powerful and so useful because it has eliminated many of physical constraints of previous machines.
- Both its blessing and its curse:
 - + No longer have to worry about physical realization of our designs.
 - No longer have physical laws that limit the complexity of our designs.

The Curse of Flexibility

- Software is the resting place of afterthoughts
- No physical constraints
 - To enforce discipline on design, construction and modification
 - To control complexity
- So flexible that start working with it before fully understanding what need to do
- “And they looked upon the software and saw that it was good, but they just had to add one other feature ...”

Abstraction from Physical Design

- Software engineers are doing system design



- Most errors in operational software related to requirements
 - Completeness a particular problem
- Software "failure modes" are different
 - Usually does exactly what you tell it to do
 - Problems occur from operation, not lack of operation
 - Usually doing exactly what software engineers wanted

Software Myths

1. Good software engineering is the same for all types of software.
2. Software is easy to change.
3. Software errors are simply “teething” problems.
4. Reusing software will increase safety.
5. Testing or “proving” software correct will remove all the errors.

Black Box Testing

Test data derived solely from specification (i.e., without knowledge of internal structure of program).

- Need to test every possible input

$x := y * 2$
if $x=5$ then $y := 3$ (since black box, only way to be sure to detect this is to try every input condition)

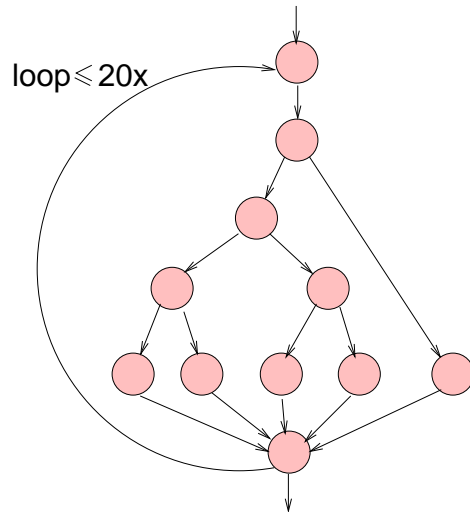
- Valid inputs up to max size of machine (not astronomical)
 - Also all invalid input (e.g., testing Ada compiler requires all valid and invalid programs)
 - If program has “memory”, need to test all possible unique valid and invalid sequences.
- So for most programs, exhaustive input testing is impractical.

White Box Testing

Derive test data by examining program's logic.

Exhaustive path testing: Two flaws

1) Number of unique paths through program is astronomical.



(control-flow graph)

$$5^{20} + 5^{19} + 5^{18} + \dots + 5 = 10^{14} \\ = 100 \text{ trillion}$$

If could develop/execute/verify one test case every five minutes = 1 billion years

If had magic test processor that could develop/execute/evaluate one test per msec = 3170 years.

White Box Testing (con't)

2) Could test every path and program may still have errors!

- Does not guarantee program matches specification, i.e., wrong program.
- Missing paths: would not detect absence of necessary paths
- Could still have data-sensitivity errors.

e.g. program has to compare two numbers for convergence

if $(A - B) < \text{epsilon} \dots$

is wrong because should compare to $\text{abs}(A - B)$

Detection of this error dependent on values used for A and B and would not necessarily be found by executing every path through program.

Stages in Process Control System Evolution

1. Mechanical systems

- Direct sensory perception of process
- Displays are directly connected to process and thus are physical extensions of it.
- Design decisions highly constrained by:
 - Available space
 - Physics of underlying process
 - Limited possibility of action at a distance

Stages in Process Control System Evolution (2)

2. Electromechanical systems

- Capability for action at a distance
- Need to provide an image of process to operators
- Need to provide feedback on actions taken.
- Relaxed constraints on designers but created new possibilities for designer and operator error.

Stages in Process Control System Evolution (3)

3. Computer-based systems

- Allow multiplexing of controls and displays.
- Relaxes even more constraints and introduces more possibility for error.
- But constraints shaped environment in ways that efficiently transmitted valuable process information and supported cognitive processes of operators.
- Finding it hard to capture and present these qualities in new systems.

A Possible Solution

- Enforce discipline and control complexity
 - Limits have changed from structural integrity and physical constraints of materials to intellectual limits
- Improve communication among engineers
- Build safety in by enforcing constraints on behavior
 - Control software contributes to accidents by:
 1. Not enforcing constraints on behavior
 2. Commanding behavior that violates constraints

Example (batch reactor)

System safety constraint:

Water must be flowing into reflux condenser whenever catalyst is added to reactor.

Software safety constraint:

Software must always open water valve before catalyst valve

The Problem to be Solved

- The primary safety problem in software-intensive systems is the lack of appropriate constraints on design.
- The job of the system safety engineer is to identify the design constraints necessary to maintain safety and to ensure the system and software design enforces them.

Safety \neq Reliability

Accidents in high-tech systems are changing their nature, and we must change our approaches to safety accordingly.

Confusing Safety and Reliability

From an FAA report on ATC software architectures:

"The FAA's en route automation meets the criteria for consideration as a safety-critical system. Therefore, en route automation systems must possess ultra-high reliability."

From a blue ribbon panel report on the V-22 Osprey problems:

"Safety [software]: ...

Recommendation: Improve reliability, then verify by extensive test/fix/test in challenging environments."

Reliability Engineering Approach to Safety

Reliability: The probability an item will perform its required function in the specified manner over a given time period and under specified or assumed conditions.

(Note: Most accidents result from errors in specified requirements or function and deviations from assumed conditions.)

- Concerned primarily with failures and failure rate reduction
 - Parallel redundancy
 - Standby sparing
 - Safety factors and margins
 - Derating
 - Screening
 - Timed replacements

Reliability Engineering Approach to Safety (2)

- Assumes accidents are the result of component failure.
 - + Techniques exist to increase component reliability
Failure rates in hardware are quantifiable.
 - Omits important factors in accidents.
May even decrease safety.
 - Many accidents occur without any component “failure”
 - e.g. Accidents may be caused by equipment operation outside parameters and time limits upon which reliability analyses are based.
 - Or may be caused by interactions of components all operating according to specification
- Highly reliable components are not necessarily safe.

What is software failure?

What is software reliability?

Software-Related Accidents

- Are usually caused by flawed requirements
 - Incomplete or wrong assumptions about operation of controlled system or required operation of computer.
 - Unhandled controlled-system states and environmental conditions.
- Merely trying to get the software “correct” or to make it reliable will not make it safer under these conditions.

Software-Related Accidents (con't.)

- Software may be highly reliable and “correct” and still be unsafe.
 - Correctly implements requirements but specified behavior unsafe from a system perspective.
 - Requirements do not specify some particular behavior required for system safety (incomplete)
 - Software has unintended (and unsafe) behavior beyond what is specified in requirements.

SYSTEM REQUIREMENTS

1. The touchdown sensors shall be sampled at 100-Hz rate.

The sampling process shall be initiated prior to lander entry to keep processor demand constant.

However, the use of the touchdown sensor data shall not begin until 12 m above the surface.

2. Each of the 3 touchdown sensors shall be tested automatically and independently prior to use of the touchdown sensor data in the onboard logic.

The test shall consist of 2 sequential sensor readings showing the expected sensor status.

If a sensor appears failed, it shall not be considered in the descent engine termination decision.

3. Touchdown determination shall be based on 2 sequential reads of a single sensor indicating touchdown.

SOFTWARE REQUIREMENTS

- a. The lander flight software shall cyclically check the state of each of the three touchdown sensors (one per leg) at 100 Hz during EDL.

- b. The lander flight software shall be able to cyclically check the touchdown event state with or without touchdown event generation enabled.

- c. Upon enabling touchdown event generation, the lander flight software shall attempt to detect failed sensors by marking the sensor as bad when the sensor indicates "touchdown state" on two consecutive reads.

- d. The lander flight software shall generate the landing event based on two consecutive reads indicating touchdown from any one of the "good" touchdown sensors.

???

ENTRY

- Lander fails to separate from cruise stage.
- Overheating, skip-out, excessive downtrack reentry points
- Excessive angle of attack causes skip out or high-velocity impact
- Heatshield fails

TERMINAL DESCENT

- Water hammer damage to propulsion system
- Propellant line rupture
- Loss of control authority (propulsion or thermal control failure)
- Loss of control (dynamics effects)
- Loss of velocity control (Doppler radar fails;
- Radar data lockout; algorithmic singularity at zero velocity; depleted propellant)
- **Premature shutdown of descent engines.**
- Excessive horizontal velocity cause lander to tip over at touchdown.

TOUCHDOWN

- Surface conditions exceed design capabilities
- Engine plume interacts with surface.
- Landing site not survivable (slope > 10 deg; lands on >30-cm rock, etc.)

PARACHUTE PHASE

- Parachute fails to deploy or fails to open
- Heatshield fails to separate.
- Legs fail to deploy
- Radar fails (altimeter)
- Spurious radar return from heatshield causes lander to separate prematurely
- Lander fails to separate from backshell

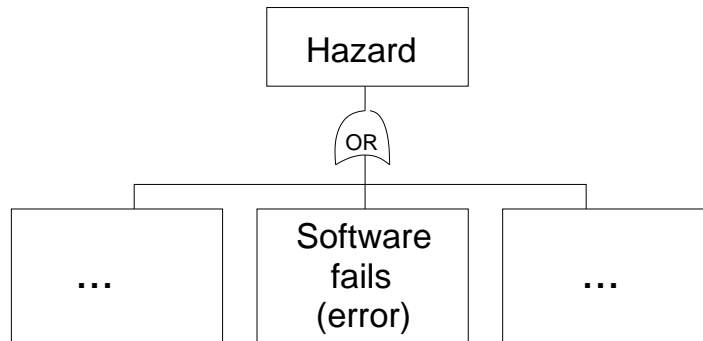
POST-LANDING

- Backshell or parachute contacts lander
- Solar array does not deploy.
- Failure to establish X-band downlink or uplink
- Failure to establish UHF link
- Medium-gain antenna fails.

COMMON TO ALL EDL PHASES

- **Flight software fails to execute properly**
- Pyrotechnic events fail.
- Propulsion component fails.
- **C&DH subsystem fails.**
- Freezing temperatures at propellant tank outlet.

Typical Fault Trees



Hazard Cause	Probability	Mitigation
Software Error	0	Test software

Reliability Approach to Software Safety

Standard engineering techniques of

- Preventing failures through redundancy
- Increasing component reliability
- Reuse of designs and learning from experience

won't work for software and system accidents.

Preventing Failures through Redundancy

- Redundancy simply makes complexity worse.
 - NASA experimental aircraft example
 - Any solutions that involve adding complexity will not solve problems that stem from intellectual unmanageability and interactive complexity.
- Majority of software–related accidents caused by requirements errors.
- Does not work for software even if accident is caused by a software implementation error.

Software errors not caused by random wearout failures.

Increasing Software Reliability (Integrity)

- Appearing in many new international standards for software safety (e.g., 61508)
 - "Safety integrity level"
 - Sometimes give reliability number (e.g., 10^{-9})
 - Can software reliability be measured? What does it even mean?
- Safety involves more than simply getting software "correct"

Example: altitude switch

1. Signal safety–increasing =>
Require any of three altimeters report below threshold
2. Signal safety–reducing =>
Require all three altimeters to report below threshold

Software Component Reuse

- One of most common factors in software-related accidents
- Software contains assumptions about its environment.

Accidents occur when these assumptions are incorrect.

- Therac-25
 - Ariane 5
 - U.K. ATC software
- Most likely to change the features embedded in or controlled by the software.
 - COTS makes safety analysis more difficult.

Safety and reliability are different qualities!

Approaches to Safety Engineering

The [FAA] administrator was interviewed for a documentary film on the [Paris DC-10] accident. He was asked how he could still consider the infamous baggage door safe, given the door failure proven in the Paris accident and the precursor accident at Windsor, Ontario. The Administrator replied—and not facetiously either—‘Of course it is safe, we certified it.’

C.O. Miller

*A Comparison of Military and Civilian
Approaches to Aviation Safety*

Three Approaches to Safety Engineering

- Civil Aviation
- Nuclear Power
- Defense

- Fly–fix–fly: analysis of accidents and feedback of experience to design and operation
- Fault Hazard Analysis:
 - Trace accidents (via fault trees) to components
 - Assign criticality levels and reliability requirements to components
- Fail–Safe Design (in Appendix B)
 - "No single failure of probable combination of failures during any one flight shall jeopardize the continued safe flight and landing of the aircraft."
- Other airworthiness requirements
- DO–178B (software certification requirements)

Nuclear Power (Defense in Depth)

- Multiple independent barriers to propagation of malfunctions
- High degree of single element integrity and lots of redundancy
- Handling single failures (no single failure of any component will disable any barrier)
- Protection ("safety") systems: automatic system shut–down
- Emphasis on reliability and availability of shutdown system and physical barriers
 - Primary approach to achieving this reliability is redundancy
- More emphasis on learning from experience since TMI

Why are these effective?

- Relatively slow pace of basic design changes
 - Use of well-understood and "debugged" designs
- Ability to learn from experience
- Conservatism in design
- Slow introduction of new technology
- Limited interactive complexity and coupling

(But software starting to change these factors)

NOTE EMPHASIS ON COMPONENT RELIABILITY

Defense (and Aerospace) – System Safety

- Emphasizes building in safety rather than adding it on to a completed design.
- Looks at systems as a whole, not just components
 - A top-down systems approach to accident prevention
- Takes a larger view of accident causes than just component failures.
 - Includes interactions among components
- Emphasizes hazard analysis and design to eliminate or control hazards.
- Emphasizes qualitative rather than quantitative approaches.

Introduction to Systems Theory

Ways to Cope with Complexity

- Analytic Reduction (Descartes)
 - Divide system into distinct parts for analysis purposes.
 - Examine the parts separately.

Physical Aspects —————> Separate physical components

Behavior —————> Events over time

- Assumes such separation feasible:
 1. The division into parts will not distort the phenomenon
 - Each component or subsystem operates independently
 - Analysis results not distorted when consider components separately
 2. Components are the same when examined singly as when playing their part in the whole.
 - Components or events not subject to feedback loops and non-linear interactions
 3. Principles governing the assembling of the components into the whole are themselves straightforward.
 - Interactions among subsystems simple enough that can be considered separate from behavior of the subsystems themselves
 - Precise nature of interactions known
 - Interactions can be examined pairwise

Organized Simplicity

Ways to Cope with Complexity (con't.)

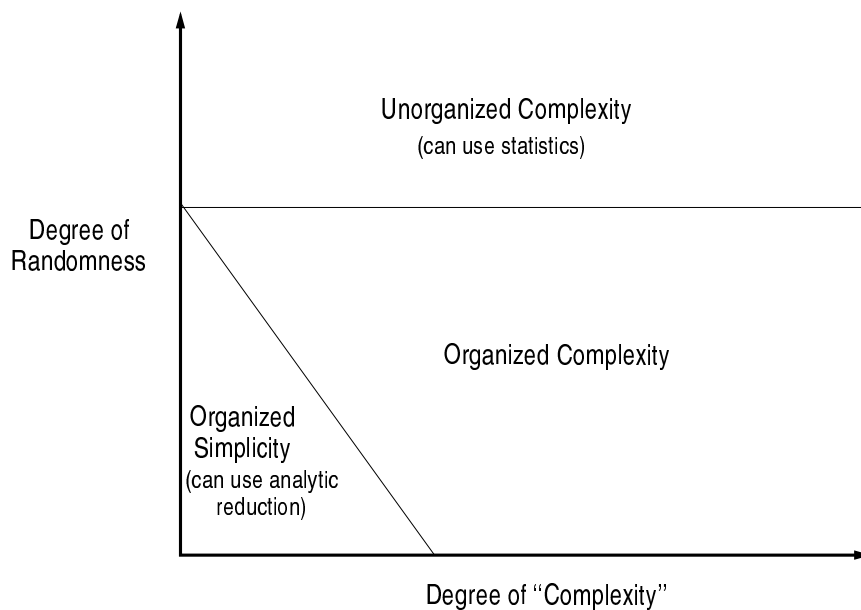
- Statistics
 - Treat as a structureless mass with interchangeable parts.
 - Use Law of Large Numbers to describe behavior in terms of averages.
- Assumes components sufficiently regular and random in their behavior that they can be studied statistically.

Unorganized Complexity

What about systems where

- Too complex for complete analysis:
 - Separation into non-interacting subsystems distorts the results.
 - The most important properties are emergent.
- Too organized for statistics
 - Too much underlying structure that distorts the statistics.
- Includes most software

Organized Complexity

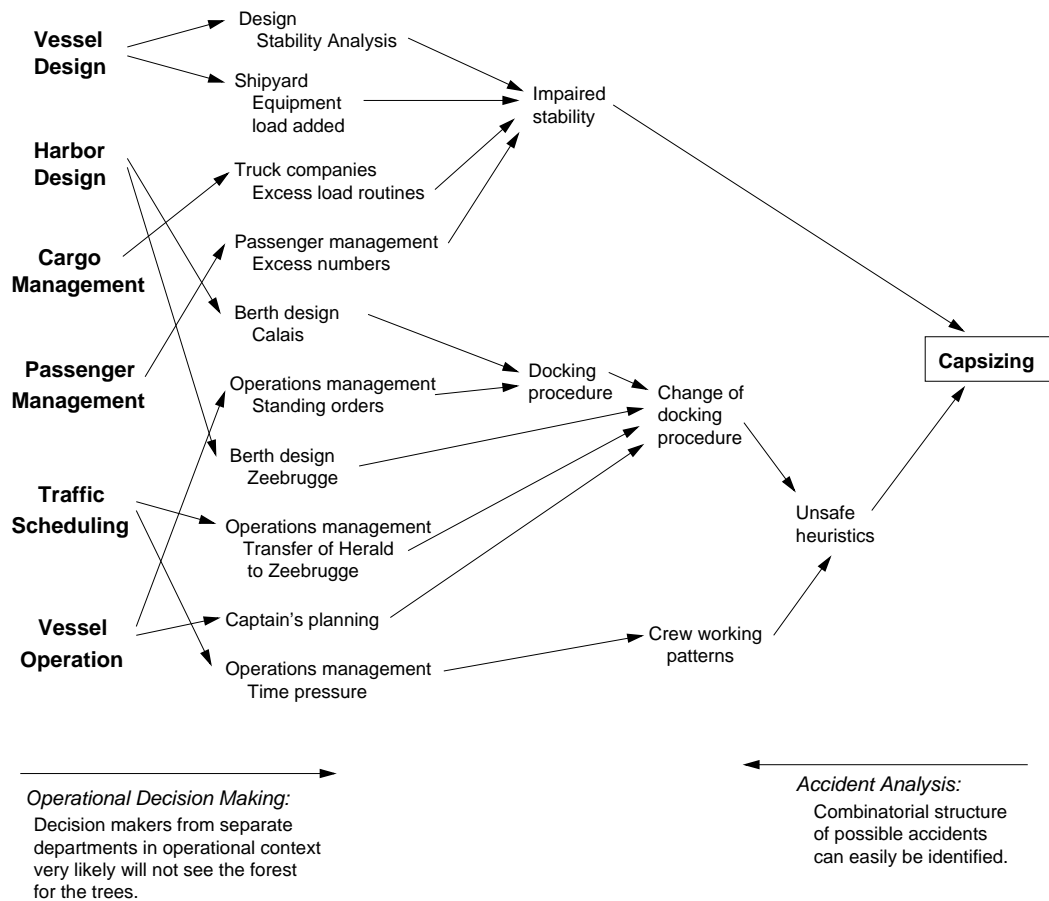


Systems Theory

- Developed for biology (von Bertalanffy) and cybernetics (Norbert Wiener)
- Basis of system engineering and system safety (ICBM systems of 50s)
 - Developed for systems with "organized complexity"
- Focuses on systems taken as a whole, not on parts taken separately
 - Some properties can only be treated adequately in their entirety, taking into account all social and technical aspects
 - These properties derive from relationships between the parts of systems — how they interact and fit together
- Two pairs of ideas:
 1. Hierarchy and emergence
 2. Communication and control

Hierarchy and Emergence

- Complex systems can be modeled as a hierarchy of levels of organization
 - Each level more complex than one below.
 - Levels characterized by emergent properties
 - Irreducible
 - Represent constraints upon the degree of freedom of components a lower level.
 - Safety is an emergent system property
 - It is NOT a component property.
 - It can only be analyzed in the context of the whole.



Communication and control

- Hierarchies characterized by control processes working at the interfaces between levels.
- A control action imposes constraints upon the activity at a lower level of the hierarchy.
- Open systems are viewed as interrelated components kept in a state of dynamic equilibrium by feedback loops of information and control.
- Control in open systems implies need for communication

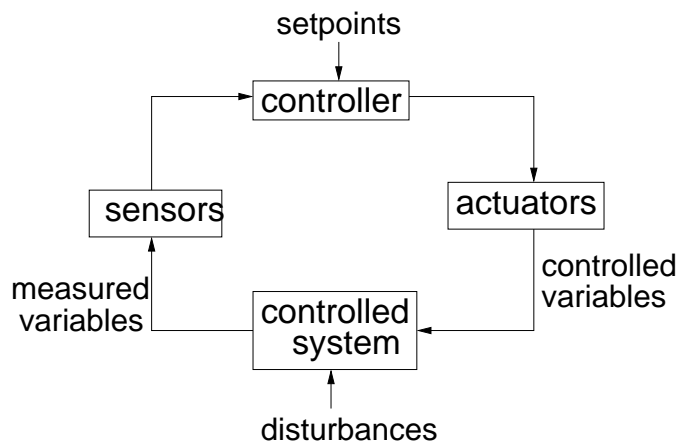
To effect control over a system requires four conditions:

Goal Condition: The controller must have a goal or goals
(e.g., to maintain a setpoint)

Action Condition: The controller must be able to affect the system state.

Model Condition: The control must be (or contain) a model of the system

Observability Condition: The controller must be able to ascertain the
state of the system.



A New Accident Model

A Systems Theory Model of Accidents

- Accidents arise from interactions among humans, machines, and the environment.
 - Not simply chains of events or linear causality, but more complex types of causal connections.
- Safety is an emergent property that arises when components of system interact with each other within a larger environment.
 - A set of constraints related to behavior of components in system enforces that property.
 - Accidents when interactions violate those constraints (a lack of appropriate constraints on the interactions).
 - Software as a controller embodies or enforces those constraints.

STAMP (Systems–Theoretic Accident Model and Processes)

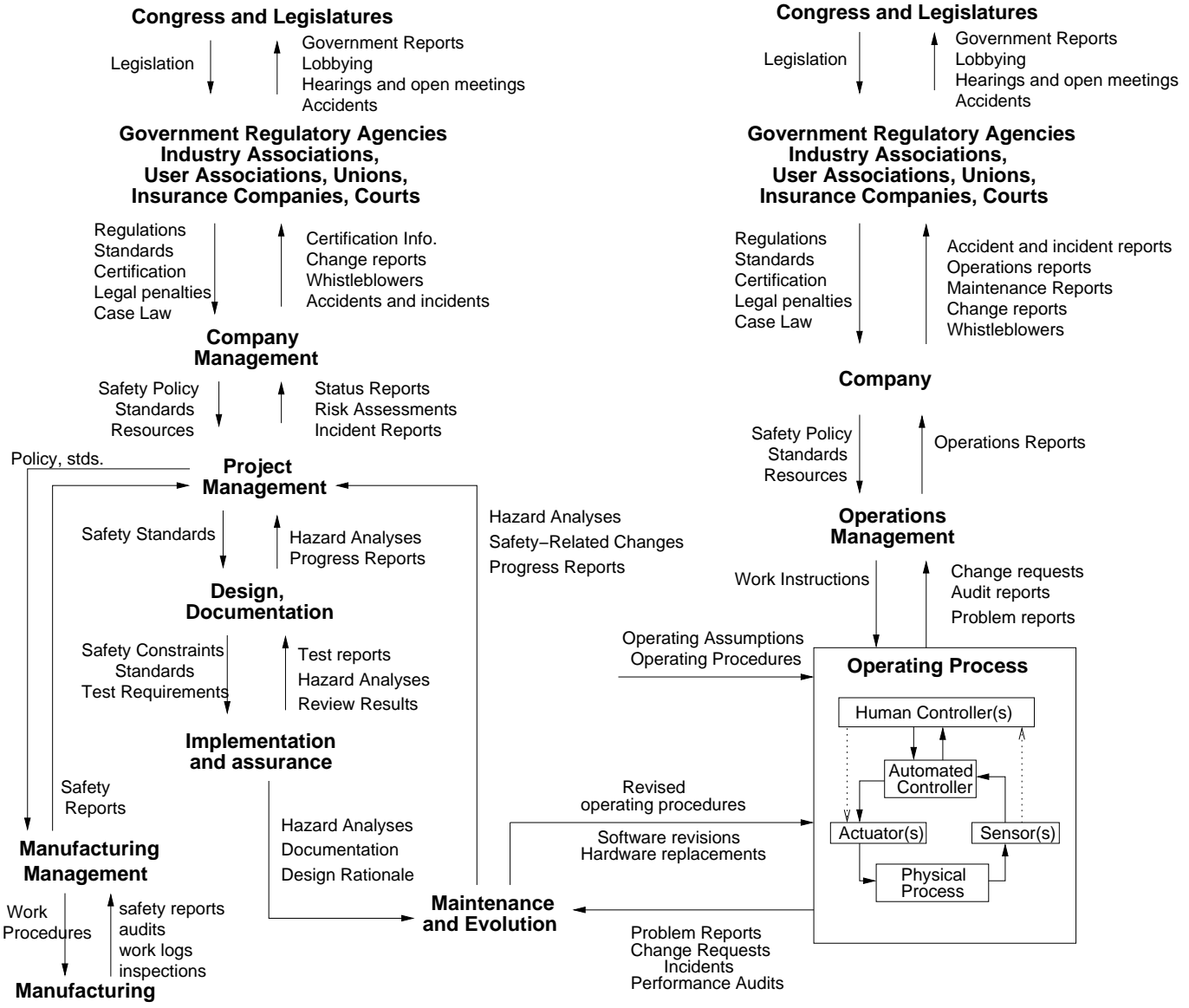
- Based on systems and control theory
- Systems not treated as a static design
 - A socio–technical system is a dynamic process continually adapting to achieve its ends and to react to changes in itself and its environment
 - Preventing accidents requires designing a control structure to enforce constraints on system behavior and adaptation.

STAMP (2)

- Views accidents as a control problem
 - e.g., O–ring did not control propellant gas release by sealing gap in field joint
 - Software did not adequately control descent speed of Mars Polar Lander.
- Events are the result of the inadequate control
 - Result from lack of enforcement of safety constraints
- To understand accidents, need to examine control structure itself to determine why inadequate to maintain safety constraints and why events occurred.

SYSTEM DEVELOPMENT

SYSTEM OPERATIONS

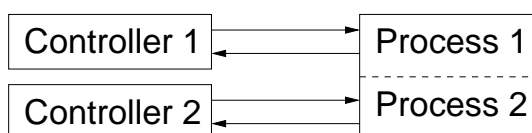


Note:

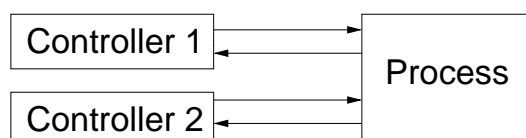
- Does not imply need for a "controller"
 - Component failures may be controlled through design
 - e.g., redundancy, interlocks, fail-safe design
 - or through process
 - manufacturing processes and procedures
 - maintenance procedures
- But does imply the need to enforce the safety constraints in some way.
- New model includes what do now and more

Accidents occur when:

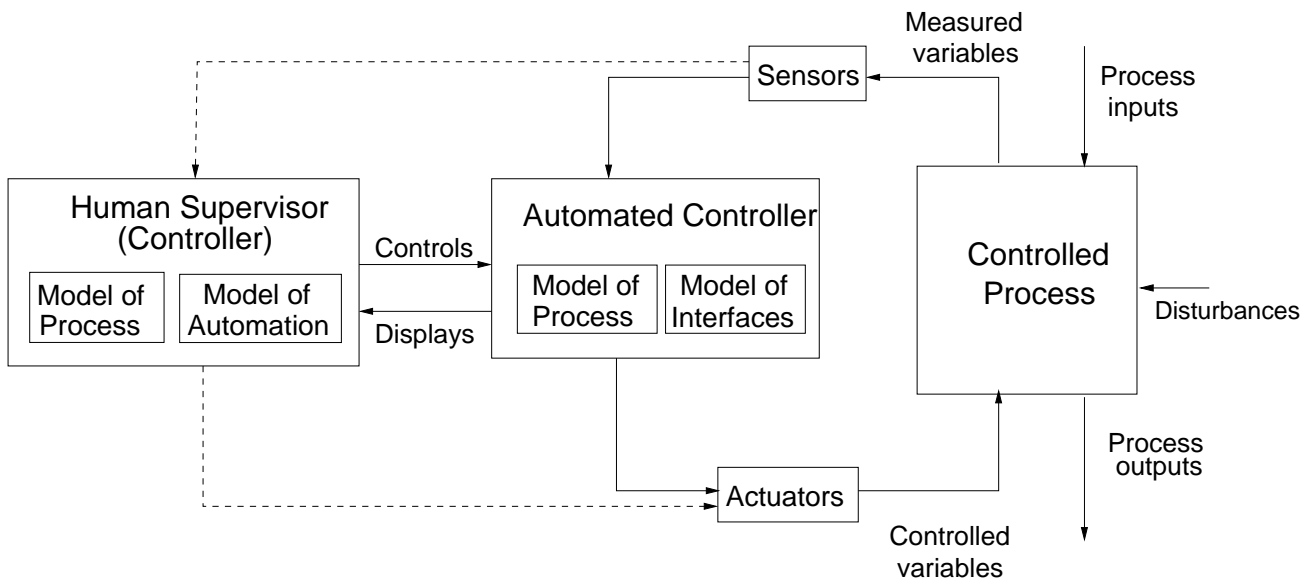
- Design does not enforce safety constraints
 - unhandled disturbances, failures, dysfunctional interactions
- Inadequate control actions
- Control structure degrades over time, asynchronous evolution
- Control actions inadequately coordinated among multiple controllers.
 - Boundary areas



- Overlap areas (side effects of decisions and control actions)

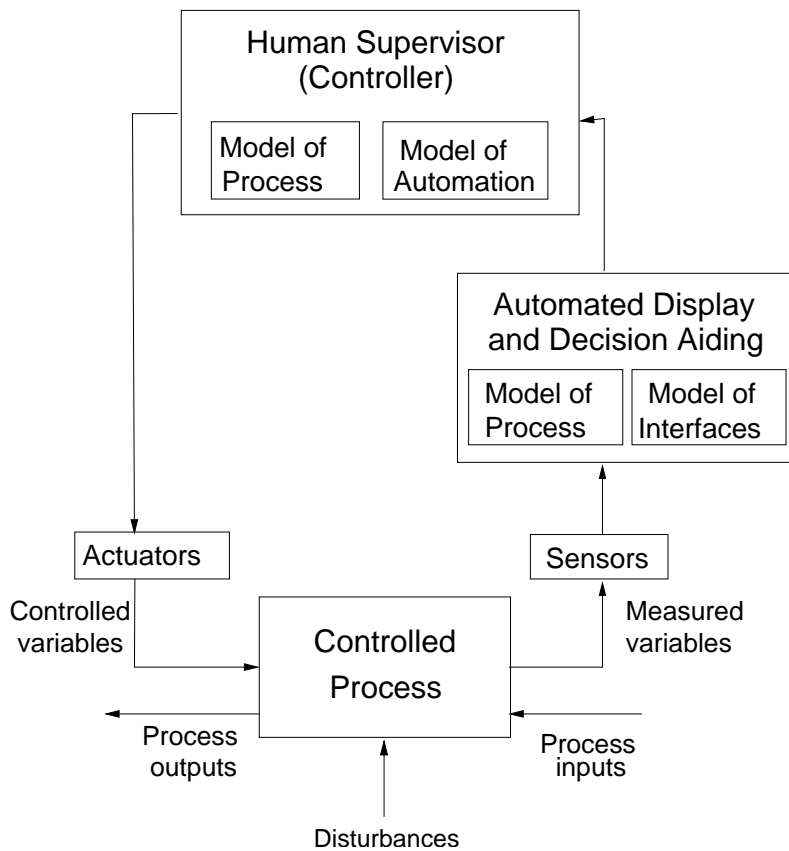


Process Models



Process models must contain:

- Required relationship among process variables
- Current state (values of process variables)
- The ways the process can change state



Relationship between Safety and Process Model

- Accidents occur when the models do not match the process and incorrect control commands are given (or correct ones not given)
- How do they become inconsistent?
 - Wrong from beginning
 - e.g. uncontrolled disturbances
 - unhandled process states
 - inadvertently commanding system into a hazardous state
 - unhandled or incorrectly handled system component failures
 - [Note these are related to what we called system accidents]
 - Missing or incorrect feedback and not updated correctly
 - Time lags not accounted for
- Explains most software-related accidents

Safety and Human Mental Models

- Explains developer errors
 - May have incorrect model of
 - required system or software behavior
 - development process
 - physical laws
 - etc.
- Also explains most human/computer interaction problems
 - Pilots and others are not understanding the automation

What did it just do?	Why won't it let us do that?
Why did it do that?	What caused the failure?
What will it do next?	What can we do so it does not
How did it get us into this state?	happen again?
How do I get it to do what I want?	
 - Or don't get feedback to update mental models or disbelieve it

1. Inadequate Enforcement of Constraints (Control Actions)

- Unidentified hazards
- Inappropriate, ineffective, or missing control actions for identified hazards
 - Design of control algorithm (process) does not enforce constraints
 - Flaw(s) in creation process
 - Process changes without appropriate change in control algorithm (asynchronous evolution)
 - Incorrect modification or adaptation
 - Process models inconsistent, incomplete, or incorrect (lack of linkup)
 - Flaw(s) in creation process
 - Flaws(s) in updating process (asynchronous evolution)
 - Time lags and measurement inaccuracies not accounted for
 - Inadequate coordination among controllers and decision makers (boundary and overlap areas)
 - Inadequate or missing feedback
 - Not provided in system design
 - Communication flaw
 - Time lag
 - Inadequate sensor operation (incorrect or no information provided)

2. Inadequate Execution of Control Action

- Communication flaw
- Inadequate actuator operation
- Time lag

Validating and Using the Model

- Can it explain (model) accidents that have already occurred?
- Is it useful?
 - In accident and mishap investigation
 - In preventing accidents
 - Hazard analysis
 - Designing for safety
- Is it better for these purposes than the chain-of-events model?

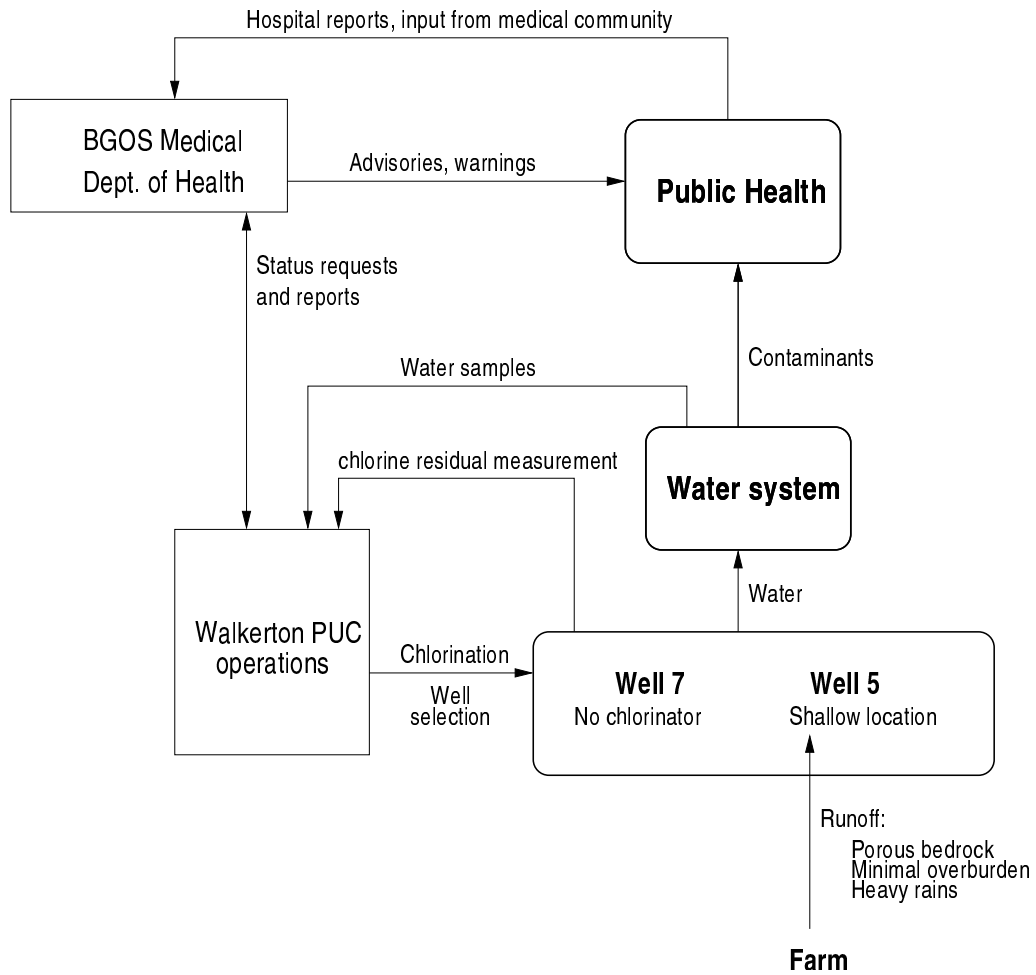
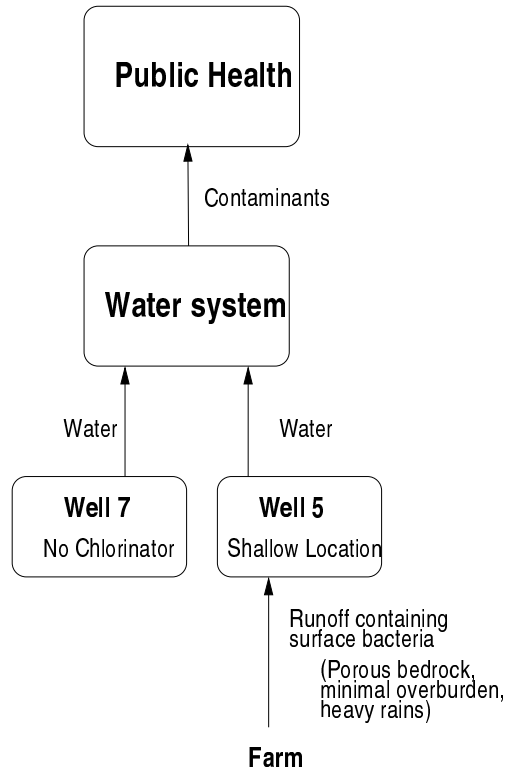
Using STAMP in Accident and Mishap Investigation and Root Cause Analysis

Modeling Accidents Using STAMP

Three types of models are needed:

1. Static safety control structure
2. Dynamic structure
 - Shows how the safety control structure changed over time
3. Behavioral dynamics
 - Dynamic processes behind the changes, i.e., why the system changes

Walkerton Physical Process



Hospital reports, input from medical community

BGOS Medical Dept. of Health

Safety Requirements and Constraints:

- Provide oversight of drinking water quality.
- Follow up on adverse drinking water quality reports.
- Issue boil water and other advisories if public health at risk.

Context in Which Decisions Made:

- Most recent water quality reports over 2 years old.
- Illness surfacing in communities outside Walkerton
- E. coli most commonly spread through meat.

Inadequate Control Actions:

- Advisory delayed.
- Advisory should have been more widely disseminated.
- Public health inspector did not follow up on 1998 inspection report.

Mental Model Flaws:

- Thought were receiving adverse water quality reports.
- Unaware of reports of E. coli linked to treated water.
- Thought Stan Koebel was relaying the truth.
- Unaware of poor state of local water operations.

Coordination:

- Assumed MOE was ensuring inspection report problems were resolved.

Public Health

Walkerton PUC Operations Management

Safety Requirements and Constraints:

- Monitor operations to ensure that sample taking and reporting is accurate and adequate chlorination is being performed.
- Keep accurate records.
- Update knowledge as required.

Context in Which Decisions Made:

- Complaints by citizens about chlorine taste in drinking water.
- Improper activities were established practice for 20 years.
- Lacked adequate training and expertise.

Inadequate Control Actions:

- Inadequate monitoring and supervision of operations
- Adverse test results not reported when asked.
- Problems discovered during inspections not rectified.
- Inadequate response after first symptoms in community
- Did not maintain proper training or operations records.

Mental Model Flaws:

- Believed sources for water system were generally safe.
- Thought untreated water safe to drink.
- Did not understand health risks posed by underchlorinated water.
- Did not understand risks of bacterial contaminants like E. coli.
- Did not believe guidelines were a high priority.

Water samples

Contaminants

Water system

chlorine residual measurement

Water

Local Operations

Safety Requirements and Constraints:

- Apply adequate doses of chlorine to kill bacteria.
- Measure chlorine residuals.

Context in Which Decisions Made:

- Lacked adequate training.

Inadequate Control Actions:

- Did not measure chlorine residuals on most days. Only started measuring in 1998.
- Made fictitious entries for residuals in daily operating sheets.
- Misstated locations from which samples had been collected.
- Did not use adequate doses of chlorine.
- Did not take measurements of chlorine residuals for Well 5 May 13 and May 15 (after symptoms of problems appeared).
- Operated Well 7 without a chlorinator.

Mental Model Flaws:

- Inadequate training led to inadequate understanding of job responsibilities.
- Thought convenience was acceptable basis for sampling.
- Believed untreated water safe to drink.

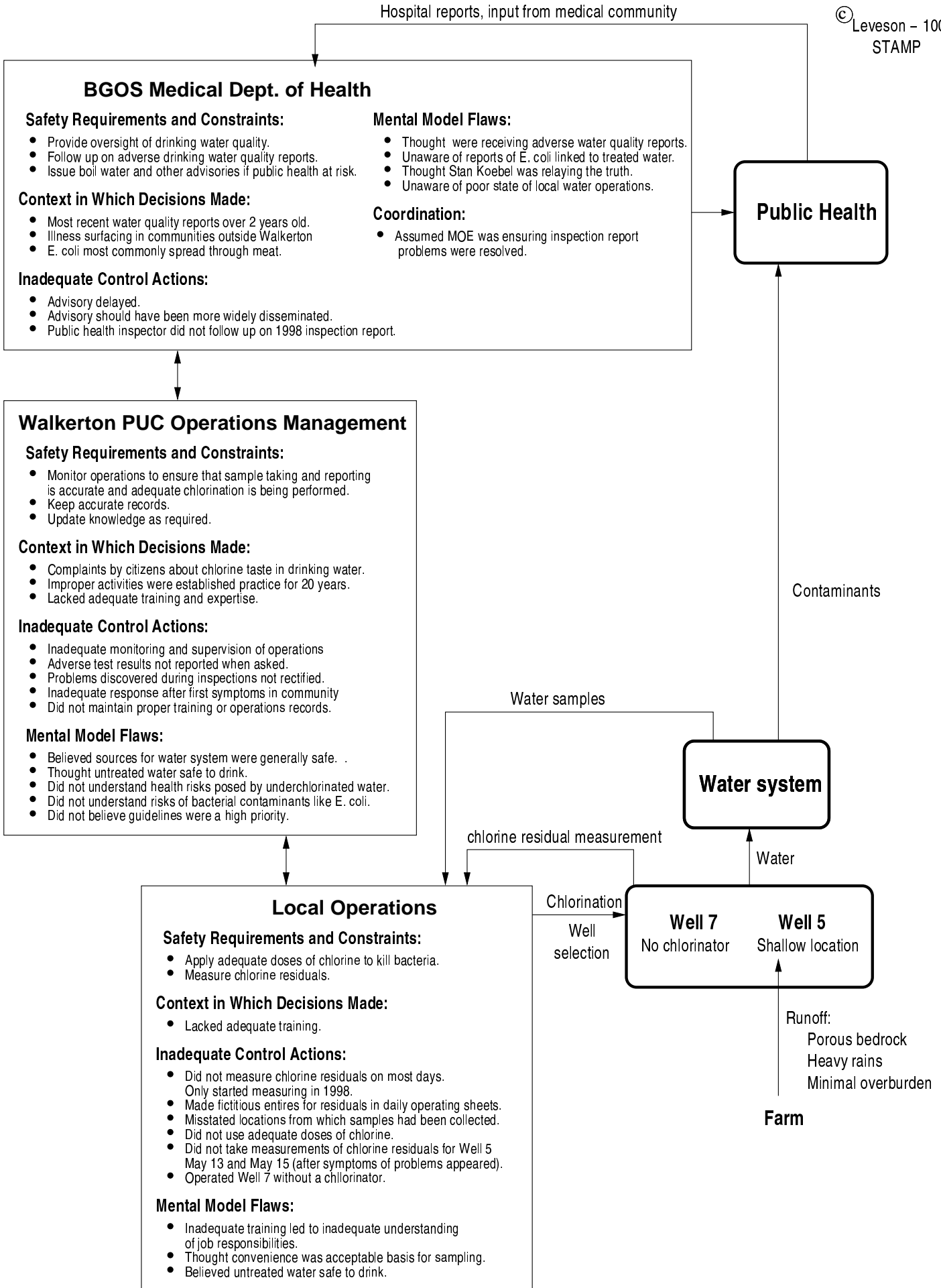
Chlorination
Well selection

Well 7
No chlorinator

Well 5
Shallow location

Runoff:
Porous bedrock
Heavy rains
Minimal overburden

Farm

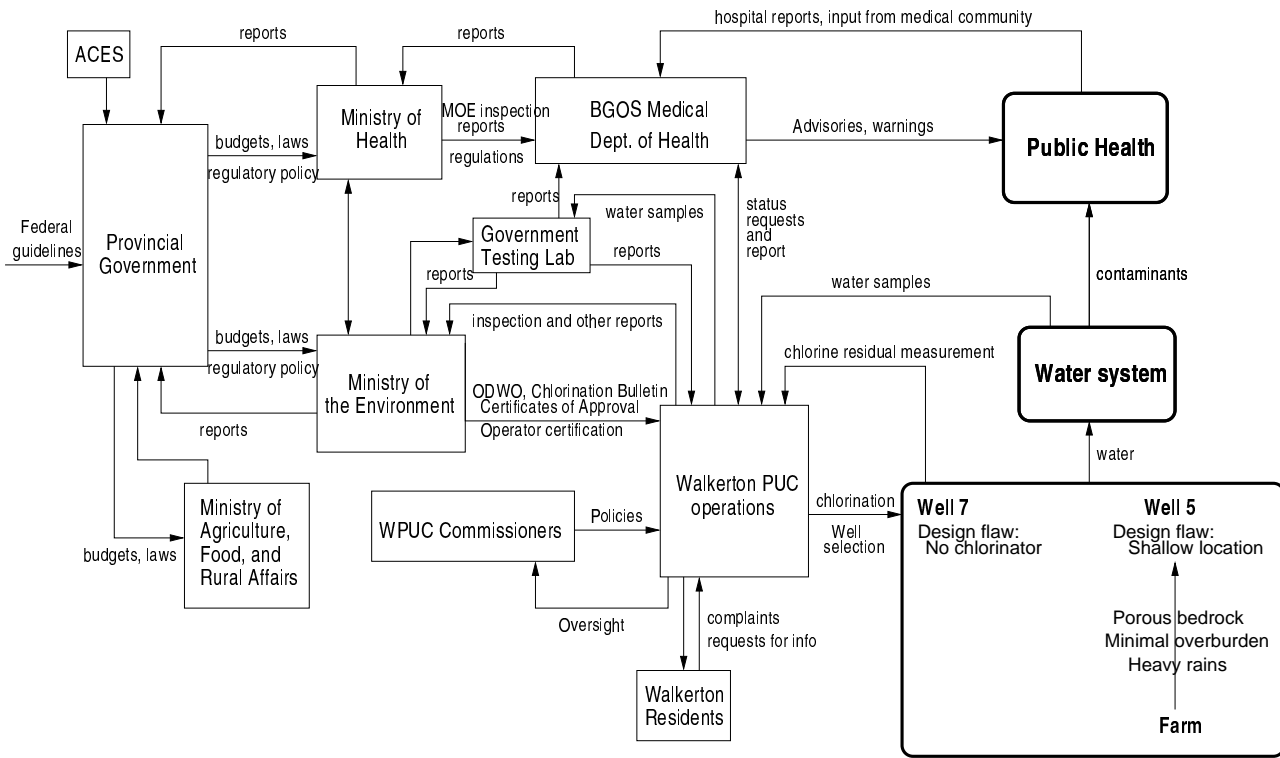


System Hazard: Public is exposed to e. coli or other health-related contaminants through drinking water.

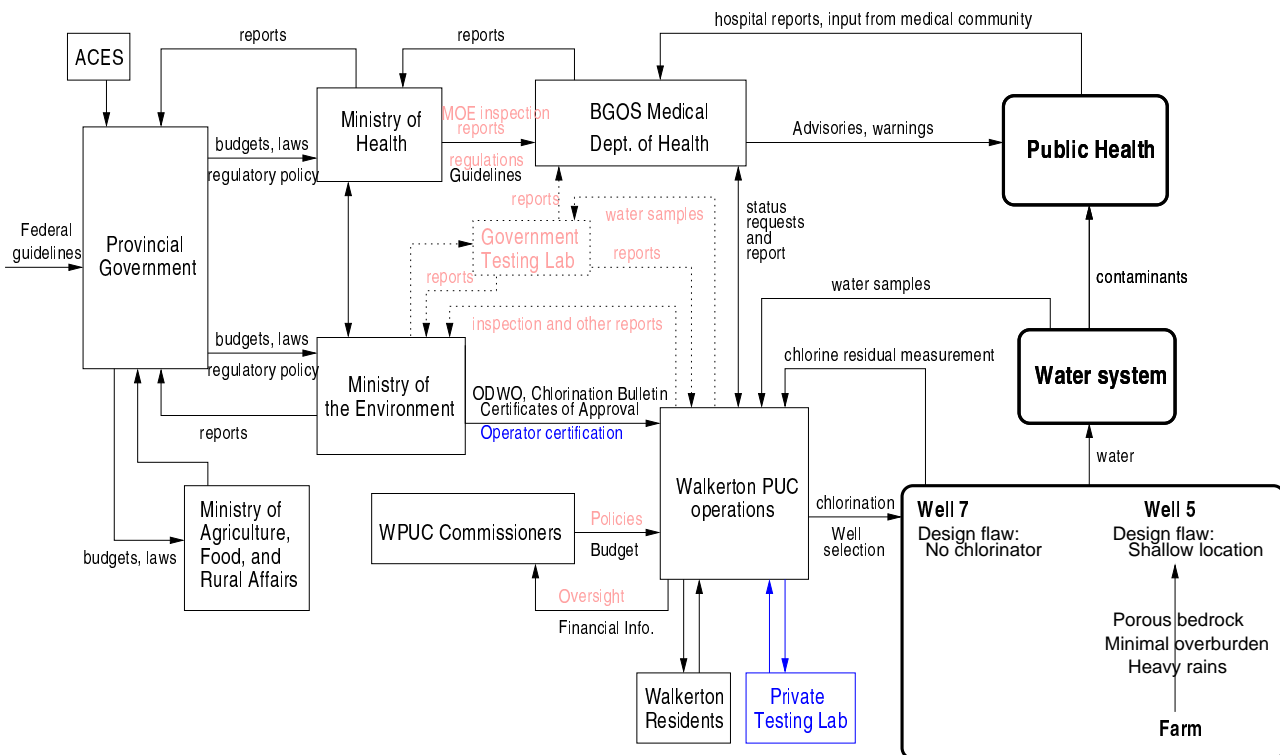
System Safety Constraints: The safety control structure must prevent exposure of the public to contaminated water.

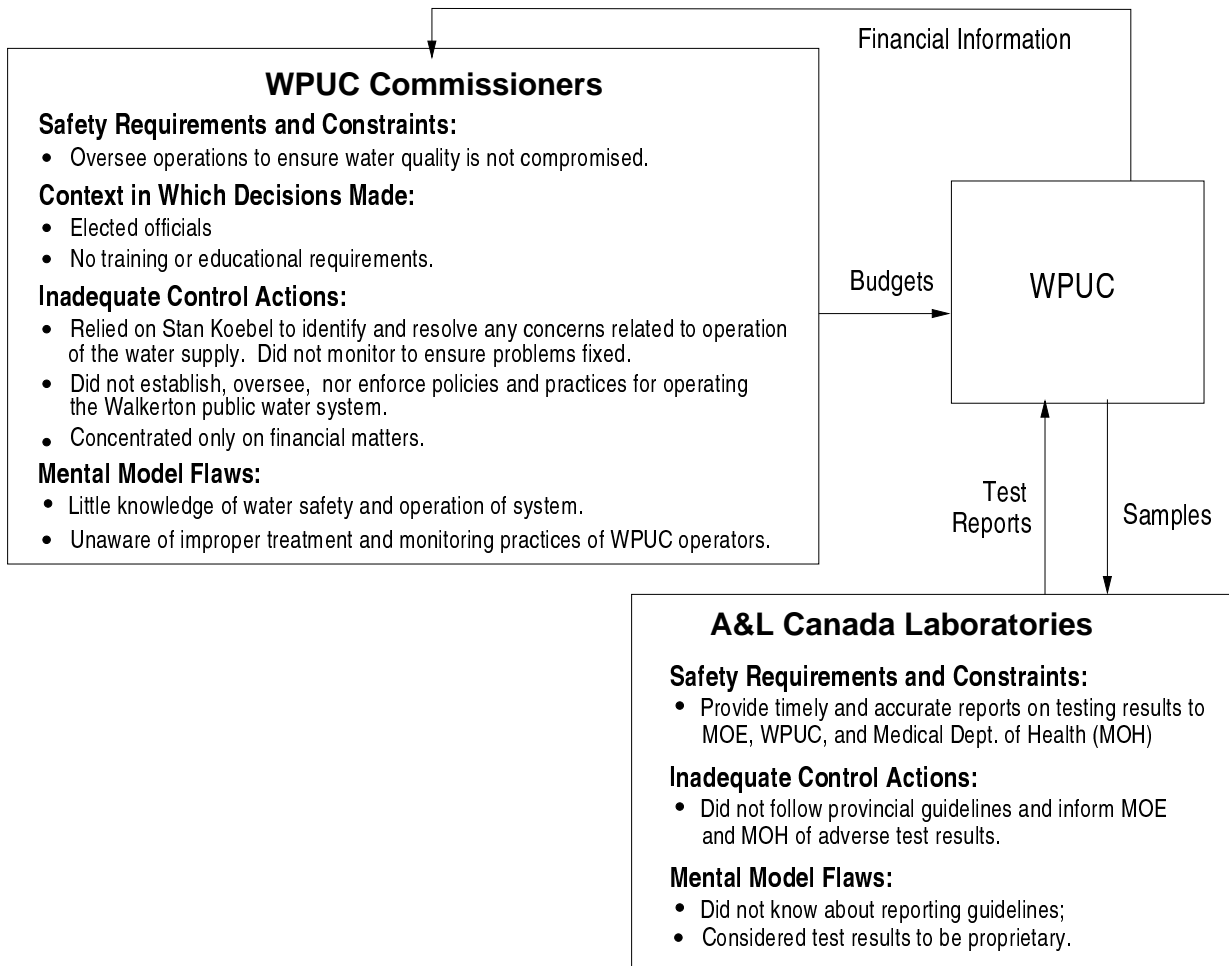
(1) Water quality must not be compromised.

(2) Public health measures must reduce risk of exposure if water quality is compromised (e.g., notification and procedures to follow)

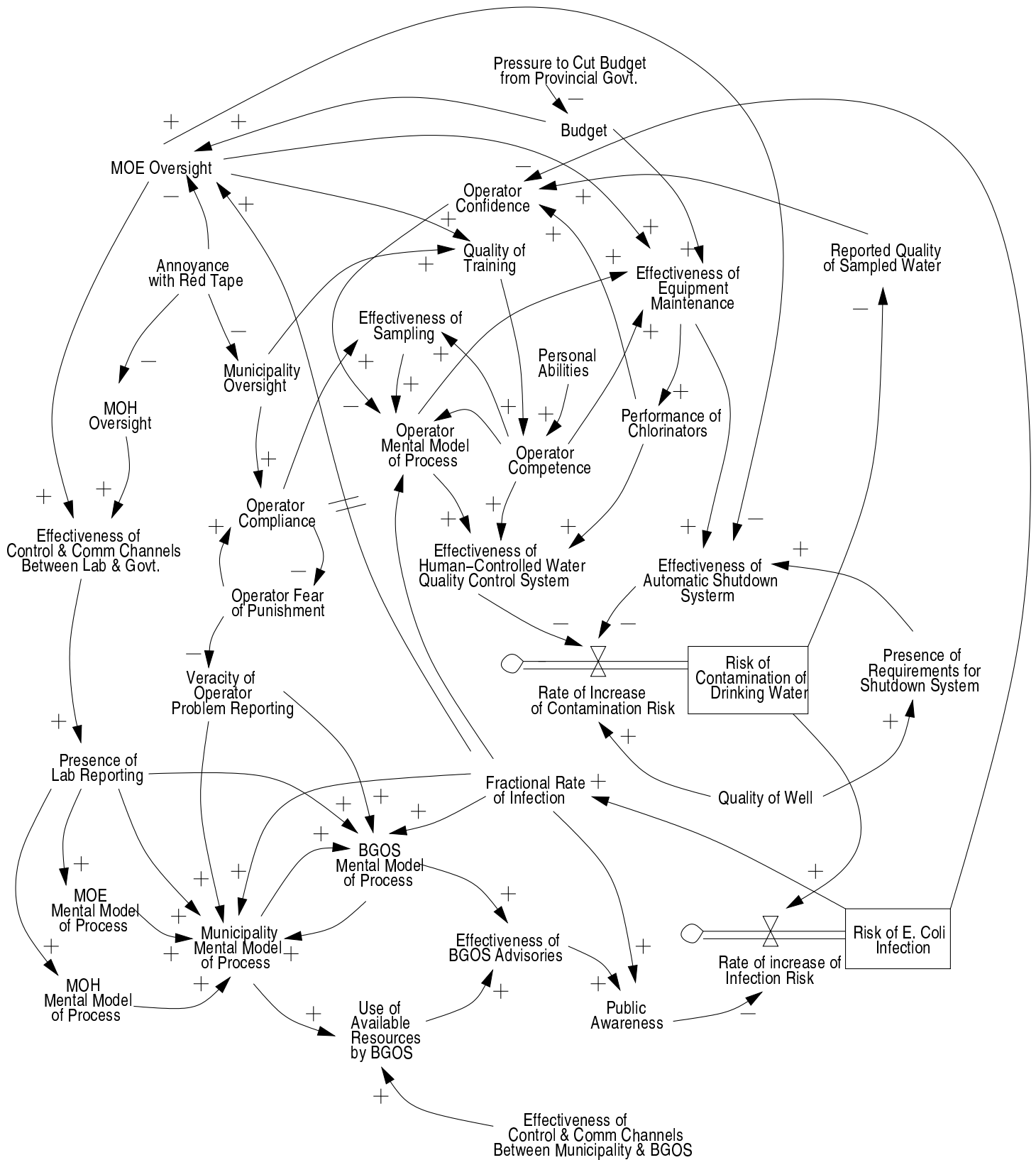


Dynamic Structure



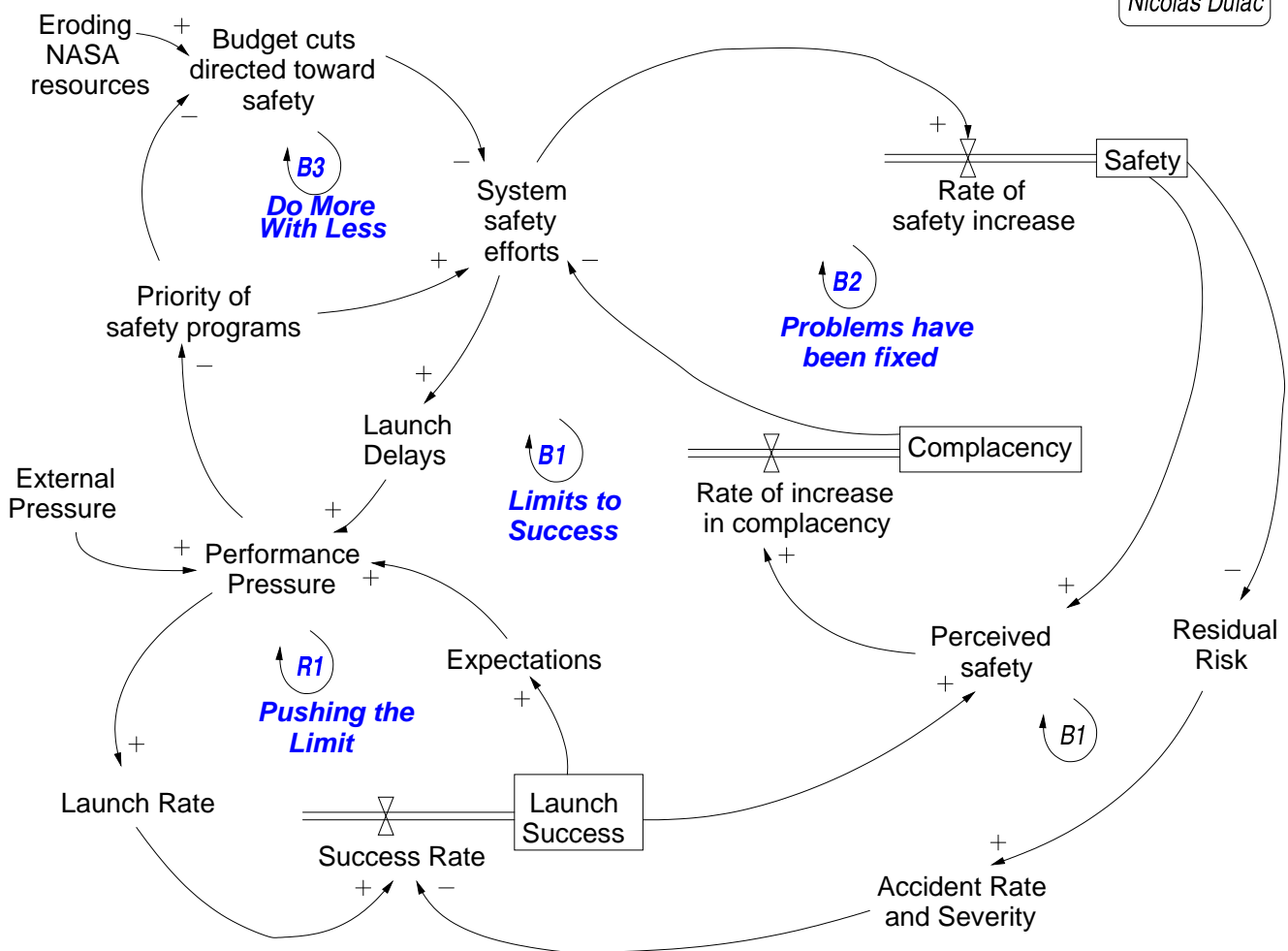


Modeling Behavioral Dynamics

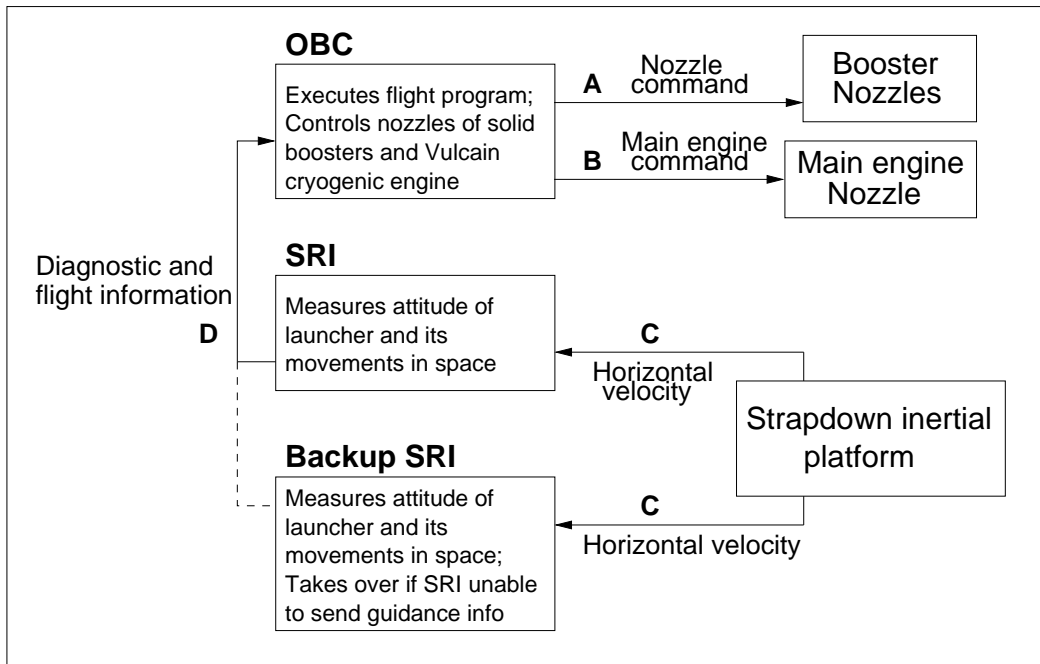


A (Partial) System Dynamics Model of the Columbia Accident

Nicolas Dulac



ARIANE 5 LAUNCHER



Ariane 5: A rapid change in attitude and high aerodynamic loads stemming from a high angle of attack create aerodynamic forces that cause the launcher to disintegrate at 39 seconds after command for main engine ignition (H0).

Nozzles: Full nozzle deflections of solid boosters and main engine lead to angle of attack of more than 20 degrees.

Self-Destruct System: Triggered (as designed) by boosters separating from main stage at altitude of 4 km and 1 km from launch pad.

OBC (On-Board Computer)

OBC Safety Constraint Violated: Commands from the OBC to the nozzles must not result in the launcher operating outside its safe envelope.

Unsafe Behavior: Control command sent to booster nozzles and later to main engine nozzle to make a large correction for an attitude deviation that had not occurred.

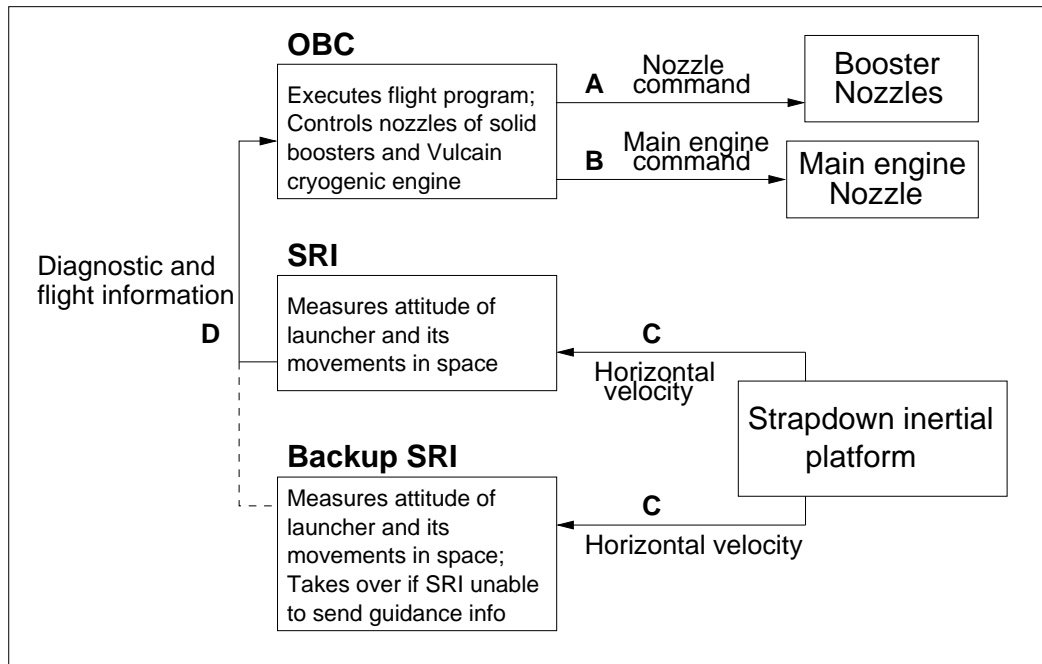
Process Model: Model of the current launch attitude is incorrect, i.e., it contains an attitude deviation that had not occurred. Results in incorrect commands being sent to nozzles.

Feedback: Diagnostic information received from SRI

Interface Model: Incomplete or incorrect (not enough information in accident report to determine which) – does not include the diagnostic information from the SRI that is available on the databus.

Control Algorithm Flaw: Interprets diagnostic information from SRI as flight data and uses it for flight control calculations. With both SRI and backup SRI shut down and therefore no possibility of getting correct guidance and attitude information, loss was inevitable.

ARIANE 5 LAUNCHER



SRI (Inertial Reference System):

SRI Safety Constraint Violated: The SRI must continue to send guidance information as long as it can get the necessary information from the strapdown inertial platform.

Unsafe Behavior: At 36.75 seconds after H0, SRI detects an internal error and turns itself off (as it was designed to do) after putting diagnostic information on the bus (D).

Control Algorithm: Calculates the Horizontal Bias (an internal alignment variable used as an indicator of alignment precision over time) using the horizontal velocity input from the strapdown inertial platform (C). Conversion from a 64-bit floating point value to a 16-bit signed integer leads to an unhandled overflow exception while calculating the horizontal bias. Algorithm reused from Ariane 4 where horizontal bias variable does not get large enough to cause an overflow.

Process Model: Does not match Ariane 5 (based on Ariane 4 trajectory data); Assumes smaller horizontal velocity values than possible on Ariane 5.

Backup SRI (Inertial Reference System):

SRI Safety Constraint Violated: The backup SRI must continue to send guidance information as long as it can get the necessary information from the strapdown inertial platform.

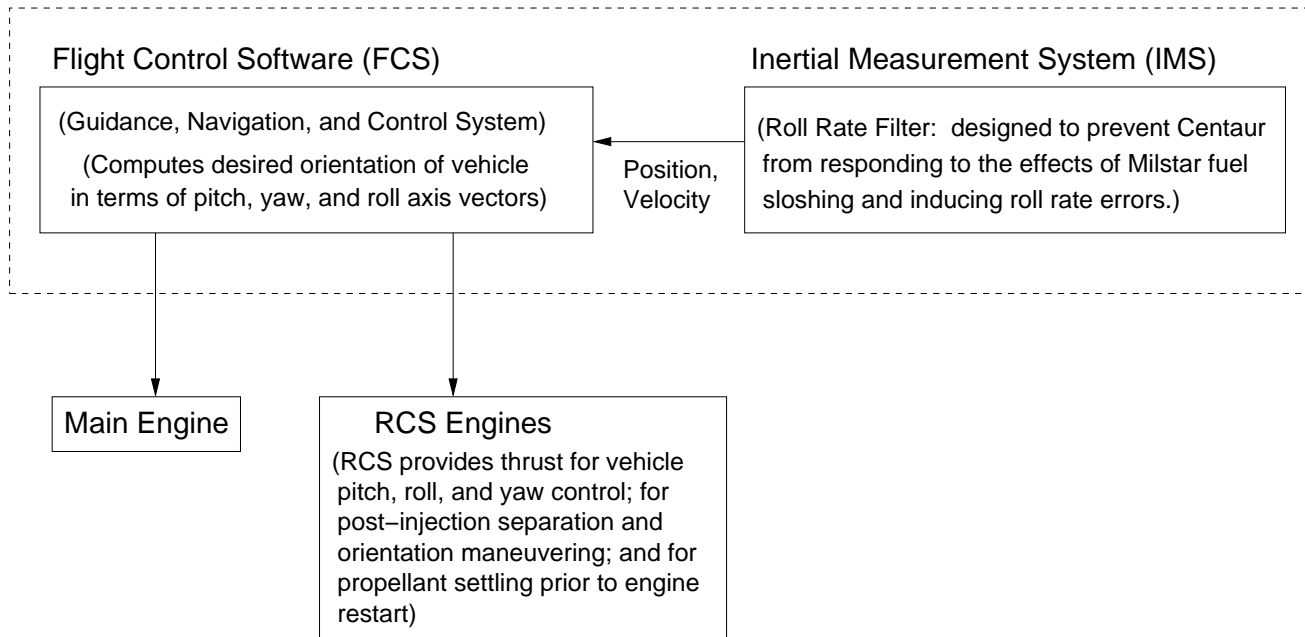
Unsafe Behavior: At 36.75 seconds after H0, backup SRI detects an internal error and turns itself off (as it was designed to do).

Control Algorithm: Calculates the Horizontal Bias (an internal alignment variable used as an indicator of alignment precision over time) using the horizontal velocity input from the strapdown inertial platform (C). Conversion from a 64-bit floating point value to a 16-bit signed integer leads to an unhandled overflow exception while calculating the horizontal bias. Algorithm reused from Ariane 4 where horizontal bias variable does not get large enough to cause an overflow. Because the algorithm was the same in both SRI computers, the overflow results in the same behavior, i.e., shutting itself off.

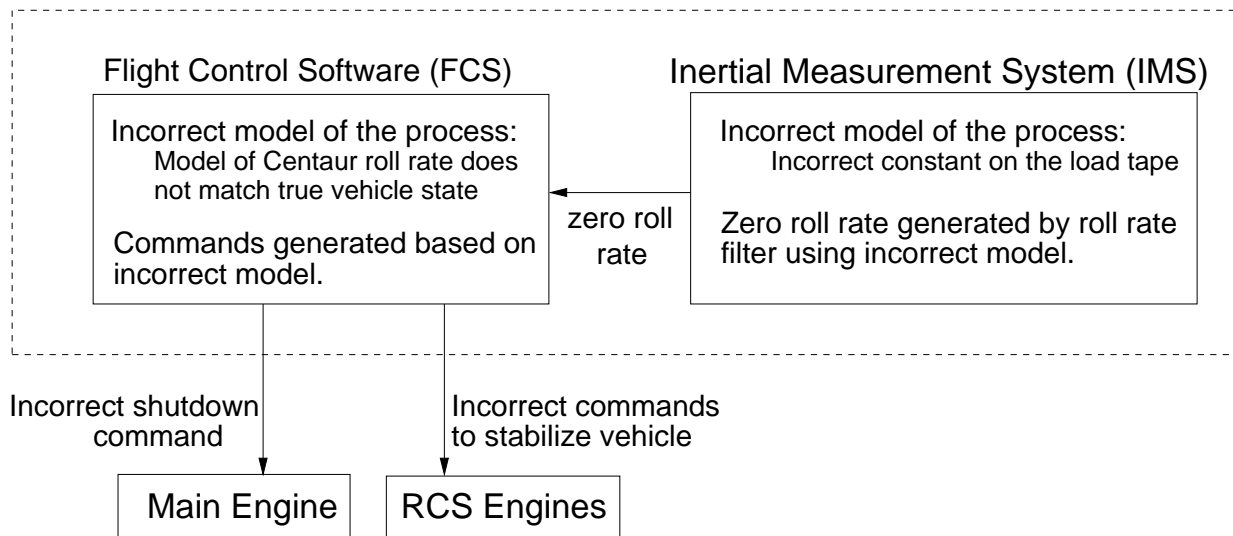
Process Model: Does not match Ariane 5 (based on Ariane 4 trajectory data); Assumes smaller horizontal velocity values than possible on Ariane 5.

Titan/Centaur/Milstar Loss

INU (Inertial Navigation Unit)



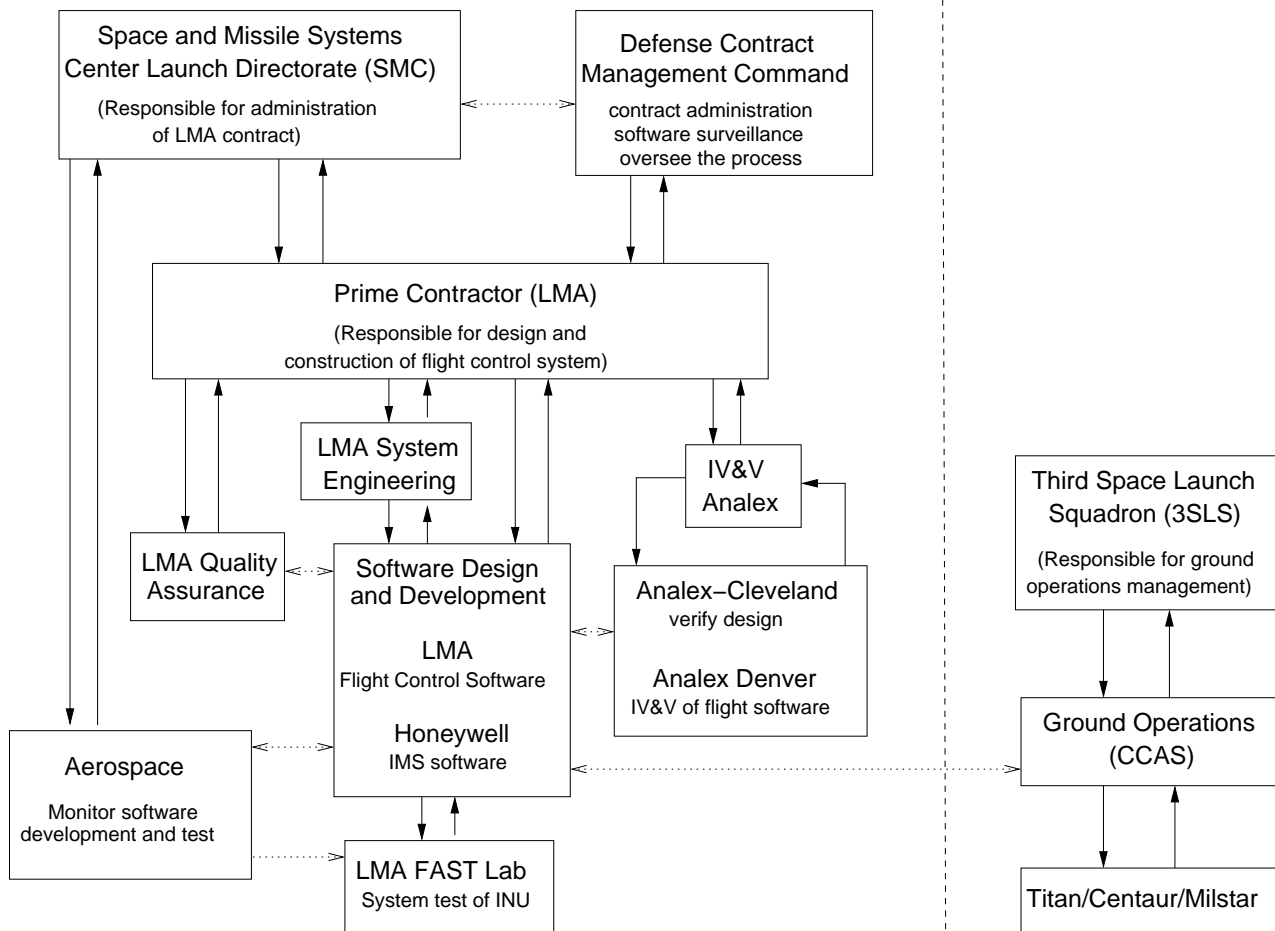
INU (Inertial Navigation Unit)



Titan 4/Centaur/Milstar

DEVELOPMENT

OPERATIONS



Analex IV&V

Safety Constraint:

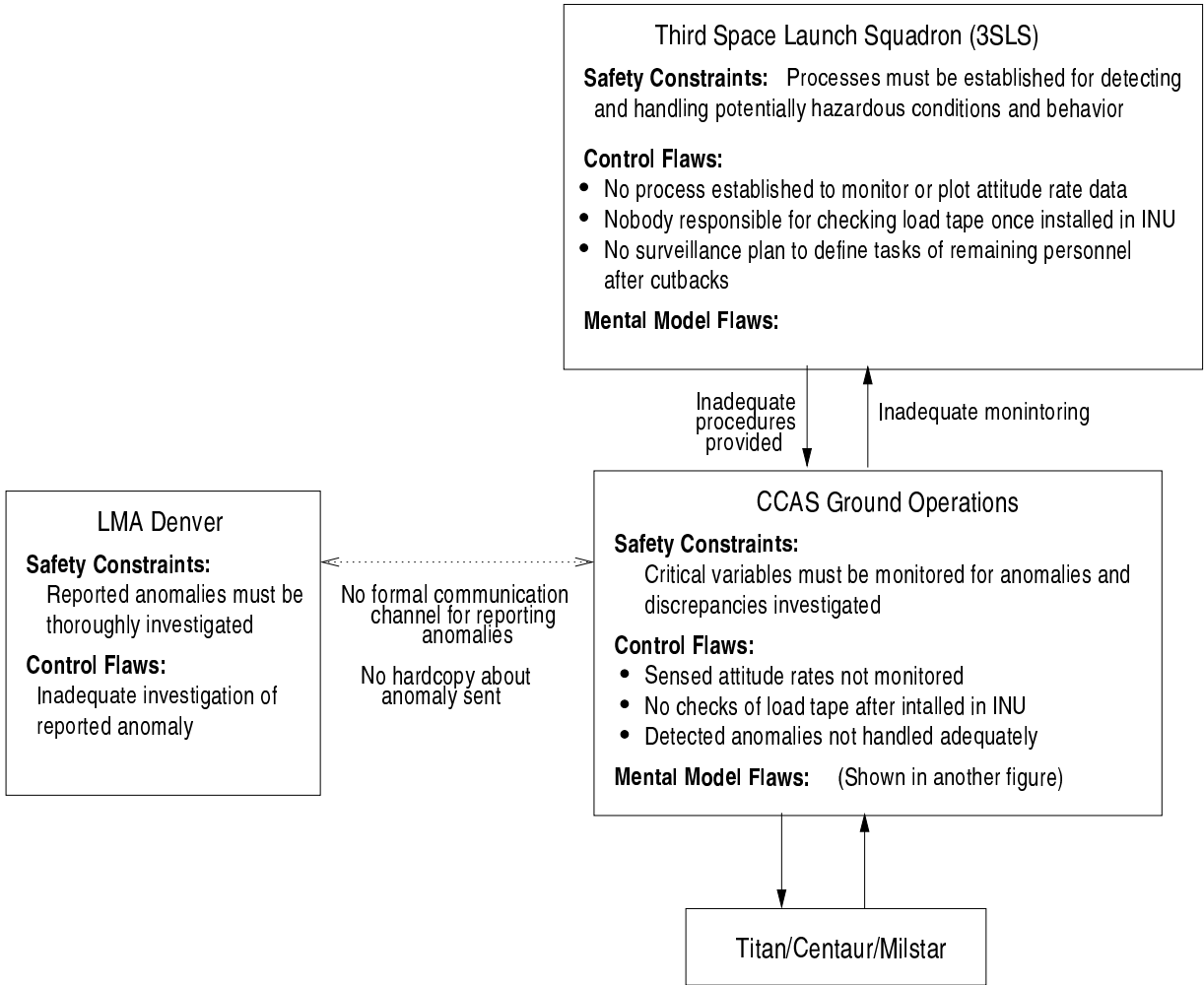
- IV&V must be performed on the as-flown system
- All safety-critical data and software must be included

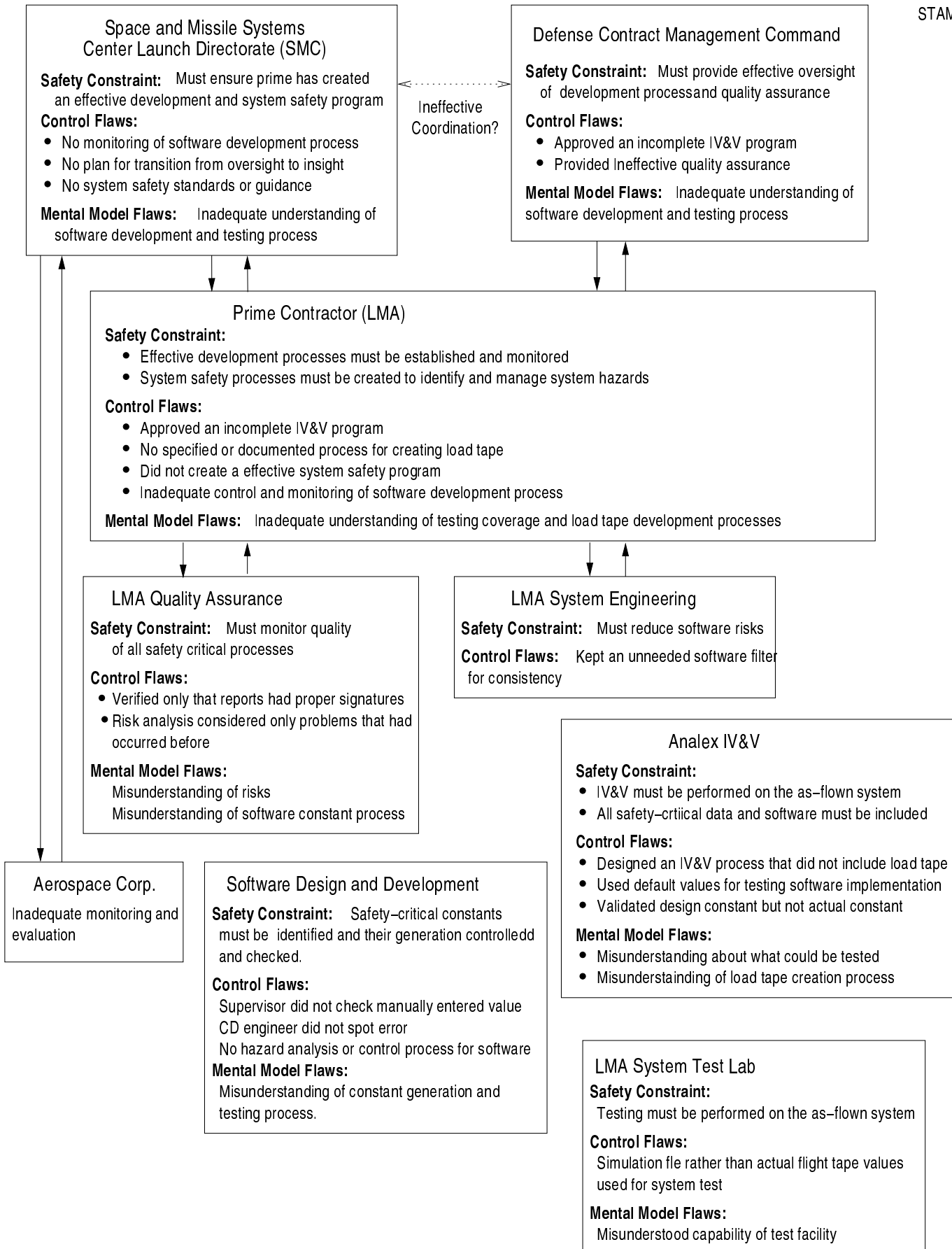
Control Flaws:

- Designed an IV&V process that did not include load tape
- Used default values for testing software implementation
- Validated design constant but not actual constant

Mental Model Flaws:

- Misunderstanding about what could be tested
- Misunderstanding of load tape creation process





STAMP vs. Traditional Accident Models

- Examines interrelationships rather than linear cause–effect chains
- Looks at the processes behind the events
- Includes entire socio–economic system
- Includes behavioral dynamics (changes over time)
 - Want to not just react to accidents and impose controls for a while, but understand why controls drift toward ineffectiveness over time and
 - Change those factors if possible
 - Detect the drift before accidents occur

The System Safety Process

My company has had a safety program for 150 years. The program was instituted as a result of a French law requiring an explosives manufacturer to live on the premises with his family.

Crawford Greenwalt
(former President of Dupont)

System Safety

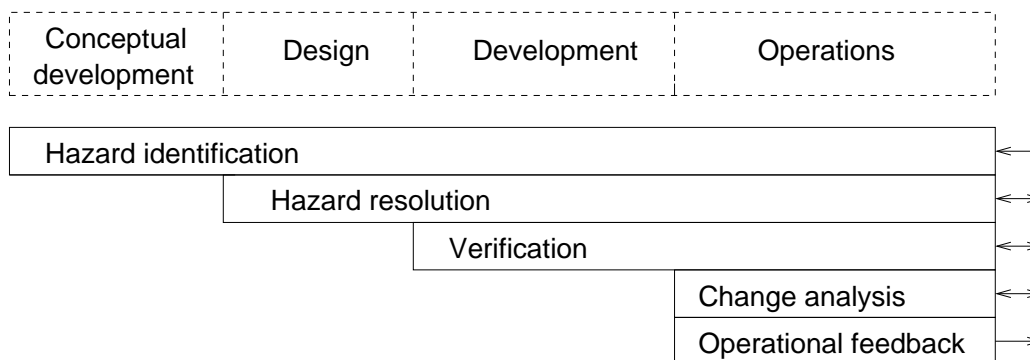
- A planned, disciplined, and systematic approach to preventing or reducing accidents throughout the life cycle of a system.
- “Organized common sense ” (Mueller, 1968)
- Primary concern is the management of hazards:

Hazard
identification
evaluation
elimination
control
through
analysis
design
management

- MIL-STD-882

System Safety (2)

- Analysis: hazard analysis and control is a continuous, iterative process throughout system development and use.



- Design: Hazard resolution precedence:
 1. Eliminate the hazard
 2. Prevent or minimize the occurrence of the hazard
 3. Control the hazard if it occurs.
 4. Minimize damage.
- Management: audit trails, communication channels, etc.

System Safety Process

Safety must be specified and designed into the system and software from the beginning.

- Program/Project Planning
 - Develop policies, procedures, etc.
 - Develop a system safety plan
 - Establish management structure, communication channels, authority, accountability, responsibility
 - Create a hazard tracking system
- Concept Development
 - Identify and prioritize system hazards
 - Generate safety-related system requirements and design constraints

System Safety Process (2)

- System Design
 - Apply hazard analysis to design alternatives
 - Determine if and how system can get into hazardous states
 - Eliminate hazards from system design if possible
 - Control hazards in system design if cannot eliminate
 - Identify and resolve conflicts between design goals
 - Trace hazard causes and controls to components (hardware, software, and human)
 - Generate component safety requirements and design constraints from system safety requirements and constraints

System Safety Process (3)

- System Implementation
 - Design safety into components
 - Verify safety of constructed system
- Configuration Control and Maintenance
 - Evaluate all proposed changes for safety
- Operations
 - Incident and accident analysis
 - Performance monitoring
 - Periodic audits

Software Safety Tasks

- Establish software safety management structure, authority, responsibility, accountability, communication channels, etc.
- Develop a software hazard tracking system and link to system hazard tracking system.
- Trace identified system hazards and system safety design constraints to software interface.
- Translate identified software-related hazards and constraints into requirements and constraints on software behavior.
- Evaluate software requirements with respect to system safety design constraints and other safety-related criteria.

Software Safety Tasks (2)

- Design software and HMI to eliminate or control hazards.
Design safety into the software.
 - Defensive programming
 - Assertions and run-time checks
 - Separation of critical functions
 - Elimination of unnecessary functions
 - Exception handling
 - etc.
- Analyze the behavior of all reused and COTS software for safety (conformance with safety requirements and constraints)
- Trace safety requirements and constraints to the code.
Document safety-related design decisions, design rationale, and other safety-related information.

Software Safety Tasks (3)

- Perform special software safety analyses
e.g.
 - human-computer interaction and interface
 - formal or informal walkthroughs or proofs
 - interface between critical and non-critical software
- Plan and perform software safety testing.
Review test results for safety issues.
Trace identified hazards back to system level.
- Analyze all proposed software changes for their effect on safety.
- Establish feedback sources. Analyze operational software and relate to hazard analysis and documented design assumptions.

Safety Information System

- Studies have ranked this second in importance only to top management concern for safety.
- Contents
 - Updated System Safety Program Plan
 - Status of activities
 - Results of hazard analyses
 - Tracking and status information on all known hazards.
 - Incident and accident information including corrective action.
 - Trend analysis data.
- Information collection
- Information analysis
- Information dissemination

Specification Principles:

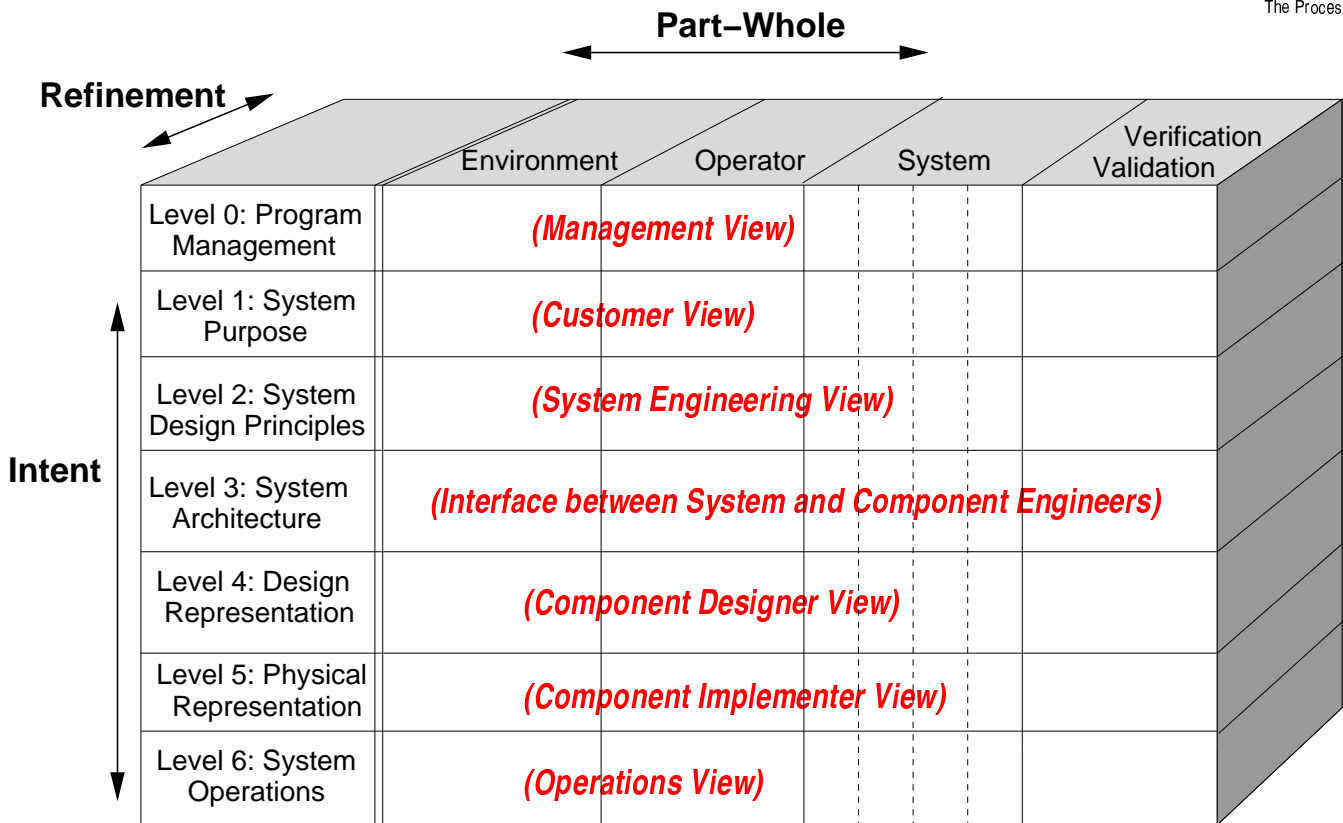
- Hazard information must be integrated into design and decision-making environment.
- Relevant information must be easy to find
- Design rationale must be specified
- Must be able to trace from high-level requirements to system design to component requirements to component design and vice versa
- Must include specification of what NOT to do
- Must be easy to review and find errors

Intent Specifications

- Based on
 - Research in how experts solve problems
 - Basic principles of system engineering
- Goals
 - Enhance communication and review
 - Capture domain knowledge and design rationale
 - Integrate safety information into decision-making environment
 - Provide traceability from requirements to design to code
 - For verification and validation
 - To support change and upgrade process

Intent Specifications (2)

- Differs in structure, not content from other specifications
 - Hierarchy of models
 - Describe system from different viewpoints
 - Traceability between models (levels)
- 7 levels:
 - Represent different views of system (not refinement)
 - From different perspectives
 - To support different types of reasoning



	Environment	Operator	System and components	V&V
Level 0 Prog. Mgmt.	Project management plans, status information, safety plan, etc.			
Level 1 System Purpose	Assumptions Constraints	Responsibilities Requirements I/F requirements	System goals, high-level requirements, design constraints, limitations	Preliminary Hazard Analysis, Reviews
Level 2 System Principles	External interfaces	Task analyses Task allocation Controls, displays	Logic principles, control laws, functional decomposition and allocation	Validation plan and results, System Hazard Analysis
Level 3 System Architecture	Environment models	Operator Task models HCI models	Blackbox functional models Interface specifications	Analysis plans and results, Subsystem Hazard Analysis
Level 4 Design Rep.		HCI design	Software and hardware design specs	Test plans and results
Level 5 Physical Rep.		GUI design, physical controls design	Software code, hardware assembly instructions	Test plans and results
Level 6 Operations	Audit procedures	Operator manuals Maintenance Training materials	Error reports, change requests, etc.	Performance monitoring and audits

Level 1: System Purpose

- Introduction
- Historical Perspective
- Environment Description
- Environment Assumptions
 - Altitude information is available from intruders with a minimum precision of 100 feet.
 - All aircraft have legal identification numbers.
- Environment Constraints
 - The behavior or interaction of non-TCAS equipment with TCAS must not degrade the performance of the TCAS equipment.

- System Functional Goals
 - Provide affordable and compatible collision avoidance system options for a broad spectrum of National Airspace System users.
- High-Level Requirements
 - [1.2] TCAS shall provide collision avoidance protection for any two aircraft closing horizontally at any rate up to 1200 knots and vertically up to 10,000 feet per minute.
 - Assumption: Commercial aircraft can operate up to 600 knots and 5000 fpm during vertical climb or controlled descent (and therefore the planes can close horizontally up to 1200 knots and vertically up to 10,000 fpm.

Hazard List for TCAS

- H1: Near midair collision (NMAC): An encounter for which, at the closest point of approach, the vertical separation is less than 100 feet and the horizontal separation is less than 500 feet.

- H2: TCAS causes controlled maneuver into ground
e.g. descend command near terrain

- H3: TCAS causes pilot to lose control of the aircraft.

- H4: TCAS interferes with other safety-related systems
e.g. interferes with ground proximity warning

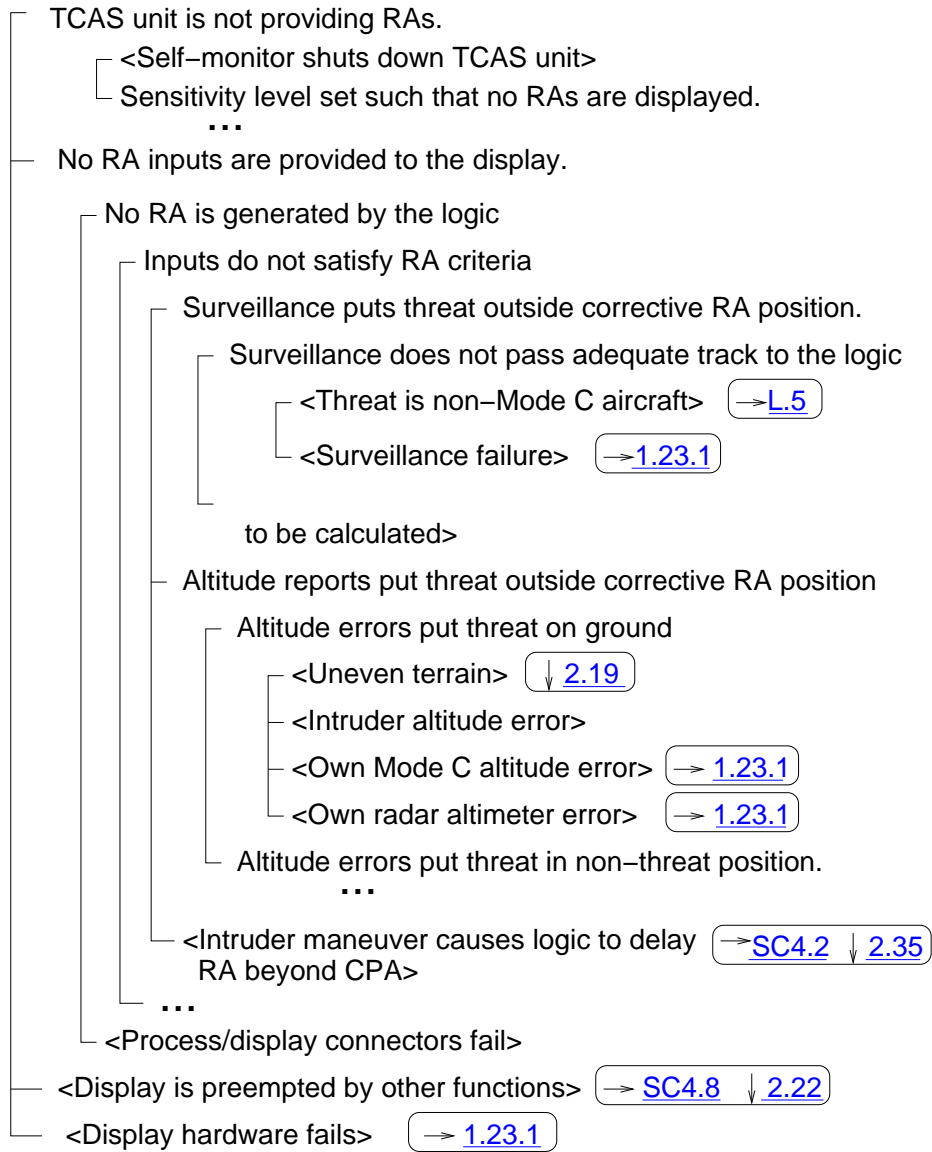
Level-1 Safety Constraints and Requirements

SC-5: The system must not disrupt the pilot and ATC operations during critical phases of flight nor disrupt aircraft operation.
[\[H3\]](#) [\[2.2.3\]](#), [2.19](#), [2.42.2](#), [2.37\]](#)

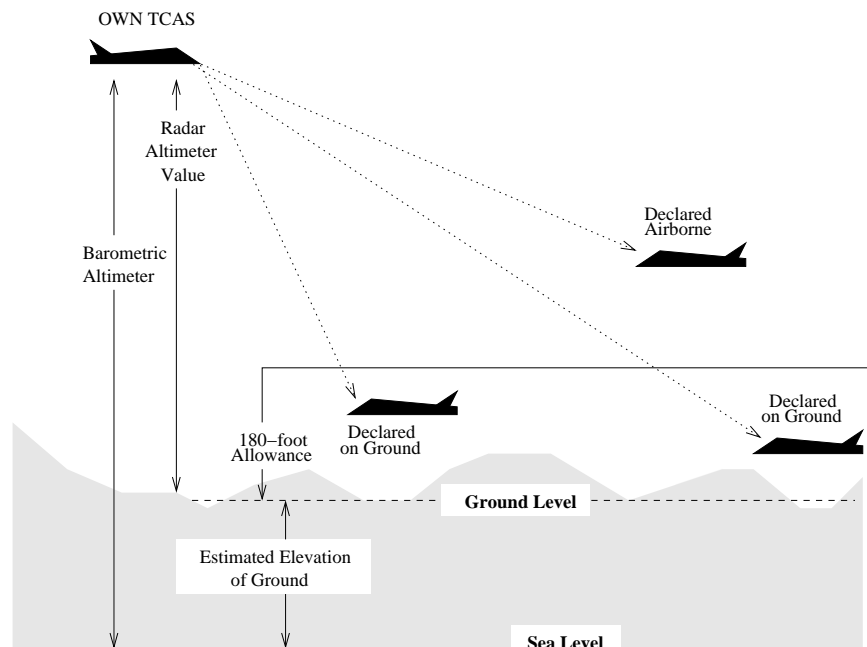
SC-5.1: The pilot of a TCAS-equipped aircraft must have the option to switch to the Traffic-Advisory mode where traffic advisories are displayed but display of resolution advisories is prohibited [\[2.37\]](#)

Assumption: This feature will be used only during final approach to parallel runways when two aircraft are projected to come close to each other and TCAS would call for an evasive maneuver [\[6.17\]](#)

TCAS does not display a resolution advisory



2.19 When below 1700 feet AGL, the CAS logic uses the difference between its own aircraft pressure altitude and radar altitude to determine the approximate elevation of the ground above sea level (see Figure 2.5). It then subtracts the latter value from the pressure altitude value received from the target to determine the approximate altitude of the target above the ground (barometric altitude - radar altitude + 180 feet). If this altitude is less than 180 feet, TCAS considers the target to be on the ground [SC-4.9]. Traffic and resolution advisories are inhibited for any intruder whose tracked altitude is below this estimate. Hysteresis is provided to reduce vacillations in the display of traffic advisories that might result from hilly terrain [FTA-320]. All RAs are inhibited when own TCAS is within 500 feet of the ground.



TCAS displays a resolution advisory that the pilot does not follow.

Pilot does not execute RA at all.

Crew does not perceive RA alarm.

<Inadequate alarm design> → 1.6,1.7,1.8 ↓ 2.74, 2.76
<Crew is preoccupied>

<Crew does not believe RA is correct.> → OP.1

...

Pilot executes the RA but inadequately

<Pilot stops before RA is removed> → OP.10

<Pilot continues beyond point RA is removed> → OP.4

<Pilot delays execution beyond time allowed> → OP.10

- Operator Requirements

OP. 4 After the threat is resolved the pilot shall return promptly and smoothly to his/her previously assigned flight path.

[FTA-560, 3.3, 6.49.7]

- Human-Machine Interface Requirements

1.8 A red visual alert shall be provided in the primary field of view for each pilot for resolution advisories.

[FTA-515, 2.84]

- System Limitations

L.5 TCAS provides no protection against aircraft with nonoperational or non-Mode C transponders.

[FTA-370]

SC-7: TCAS must not create near misses (result in a hazardous level of vertical separation that would not have occurred had the aircraft not carried TCAS) [\[H1\]](#)

SC-7.1: Crossing maneuvers must be avoided if possible.
[\[2.36, 2.38, 2.48, 2.49.2\]](#)

SC-7.2: The reversal of a displayed advisory must be extremely rare [\[2.51, 2.56.3, 2.65.3, 2.66\]](#)

SC-7.3: TCAS must not reverse an advisory if the pilot will have insufficient time to respond to the RA before the closest point of approach (four seconds or less) or if own and intruder aircraft are separated by less than 200 feet vertically when ten seconds or less remain to closest point of approach [\[2.52\]](#)

Example Level-2 System Design for TCAS

SENSE REVERSALS [↓ Reversal-Provides-More-Separation](#)

2.51 In most encounter situations, the resolution advisory sense will be maintained for the duration of an encounter with a threat aircraft.
[\[SC-7.2 \]](#)

However, under certain circumstances, it may be necessary for that sense to be reversed. For example, a conflict between two TCAS-equipped aircraft will, with very high probability, result in selection of complementary advisory senses because of the coordination protocol between the two aircraft. However, if coordination communications between the two aircraft are disrupted at a critical time of sense selection, both aircraft may choose their advisories independently.

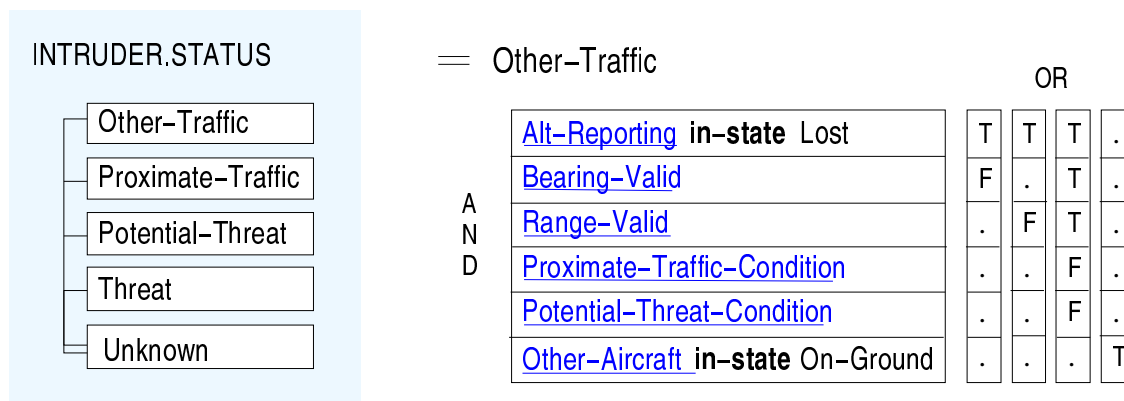
[\[FTA-1300 \]](#)

This could possibly result in selection of incompatible senses.

[\[FTA-395 \]](#)

2.51.1 [Information about how incompatibilities are handled]

Example from Level 3 Model of Collision Avoidance Logic



Description: A threat is reclassified as other traffic if its altitude reporting has been lost ([↑2.13](#)) and either the bearing or range inputs are invalid; if its altitude reporting has been lost and both the range and bearing are valid but neither the proximate nor potential threat classification criteria are satisfied; or the aircraft is on the ground ([↑2.12](#)).

Mapping to Level 2: [↑2.23](#), [↑2.29](#)

Mapping to Level 4: [↓4.7.1](#), [Traffic_advisory_detection](#)

Example MITRE Pseudo-Code from Level 4

```

PROCESS Traffic_advisory_detection;
  CLEAR PROX_TEST;
  IF (ITF.TACODE EQ $RA)
    THEN ITF.TATIME = P.MINATIME;
  ELSEIF (ITF.IOGROUND EQ $TRUE)
    THEN ITF.TACODE = $NOTAPA;
    ITF.TATIME = 0;
  OTHERWISE PERFORM Traffic_parameters;
  PERFORM Traffic_range_test;
  IF (RHITA EQ $TRUE)
    THEN PERFORM Range_hit_processing;
  ELSEIF (ITF.TATIME NE 0)
    THEN ITF.TATIME = ITF.TATIME - 1;
    IF (ITF.MODC EQ $FALSE)
      THEN ITF.TACODE = $TANMC;
      ELSE ITF.TACODE = $TAMC;
    ELSE SET PROX_TEST;
  IF (PROX_TEST EQ $TRUE)
    THEN PERFORM Proximity_test;
    IF (PRXHITA EQ $TRUE)
      THEN ITF.TACODE = $PENDPA;
      ELSE ITF.TACODE = $NOTAPA;
  END Traffic_advisory_detection;

```

Hazard Analysis for Software-Intensive Systems

Terminology

Accident: An undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss.

Incident: An event that involves no loss (or only minor loss) but with the potential for loss under different circumstances.

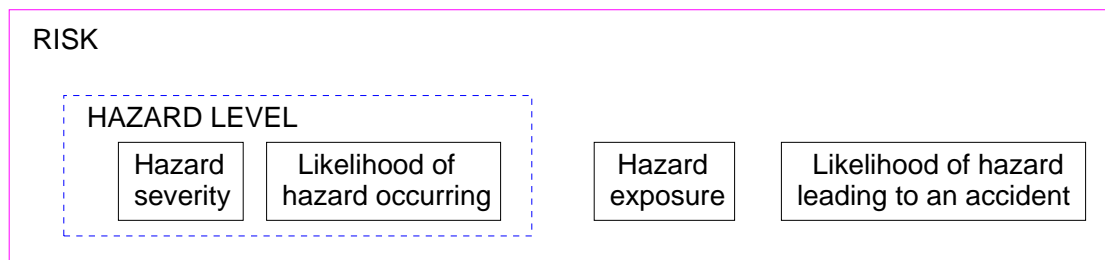
Hazard: A state or set of conditions that, together with other conditions in the environment, will lead to an accident (loss event).

Note that a hazard is NOT equal to a failure.

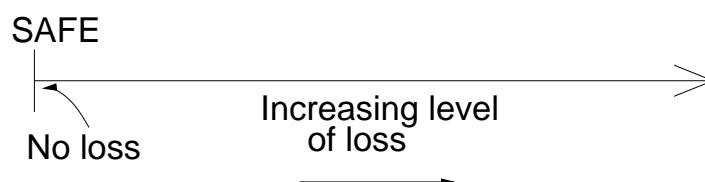
“Distinguishing hazards from failures is implicit in understanding the difference between safety and reliability engineering.
C.O Miller”

Hazard Level: A combination of severity (worst potential damage in case of an accident) and likelihood of occurrence of the hazard.

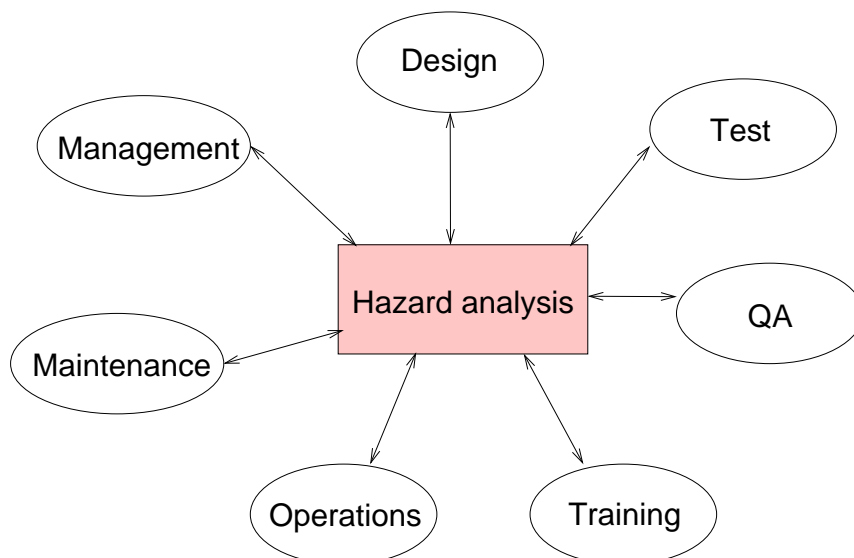
Risk: The hazard level combined with the likelihood of the hazard leading to an accident plus exposure (or duration) of the hazard.



Safety: Freedom from accidents or losses.



Hazard analysis affects, and in turn, is affected by all aspects of the development process.



Hazard Analysis

Hazard analysis is the heart of any system safety program.

Used for:

- Developing requirements and design constraints
- Validating requirements and design for safety
- Preparing operational procedures and instructions
- Test planning
- Management planning

Serves as:

- A framework for ensuing steps
- A checklist to ensure management and technical responsibilities for safety are accomplished.

"Types" (Stages) of Hazard Analysis

- Preliminary Hazard Analysis (PHA)
 - Identify, assess, and prioritize hazards
 - Identify high-level safety design constraints
- System Hazard Analysis (SHA)
 - Examine subsystem interfaces to evaluate safety of system working as a whole
 - Refine design constraints and trace to individual components (including operators)

"Types" (Stages) of Hazard Analysis (2)

- Subsystem Hazard Analysis (SSHA)
 - Determine how subsystem design and behavior can contribute to system hazards.
 - Evaluate subsystem design for compliance with safety constraints.
- Change and Operations Analysis
 - Evaluate all changes for potential to contribute to hazards
 - Analyze operational experience

Preliminary Hazard Analysis

1. Identify system hazards
2. Translate system hazards into high–level system safety design constraints.
3. Assess hazards if required to do so.
4. Establish the hazard log.

System Hazards for Automated Train Doors

- Train starts with door open.
- Door opens while train is in motion.
- Door opens while improperly aligned with station platform.
- Door closes while someone is in doorway
- Door that closes on an obstruction does not reopen or reopened door does not reclose.
- Doors cannot be opened for emergency evacuation.

System Hazards for Air Traffic Control

- Controlled aircraft violate minimum separation standards (NMAC).
- Airborne controlled aircraft enters an unsafe atmospheric region.
- Controlled airborne aircraft enters restricted airspace without authorization.
- Controlled airborne aircraft gets too close to a fixed obstacle other than a safe point of touchdown on assigned runway (CFIT)
- Controlled airborne aircraft and an intruder in controlled airspace violate minimum separation.
- Controlled aircraft operates outside its performance envelope.
- Aircraft on ground comes too close to moving objects or collides with stationary objects or leaves the paved area.
- Aircraft enters a runway for which it does not have clearance.
- Controlled aircraft executes an extreme maneuver within its performance envelope.
- Loss of aircraft control.

Exercise: Identify the system hazards for this cruise–control system

The cruise control system operates only when the engine is running. When the driver turns the system on, the speed at which the car is traveling at that instant is maintained. The system monitors the car's speed by sensing the rate at which the wheels are turning, and it maintains desired speed by controlling the throttle position. After the system has been turned on, the driver may tell it to start increasing speed, wait a period of time, and then tell it to stop increasing speed. Throughout the time period, the system will increase the speed at a fixed rate, and then will maintain the final speed reached.

The driver may turn off the system at any time. The system will turn off if it senses that the accelerator has been depressed far enough to override the throttle control. If the system is on and senses that the brake has been depressed, it will cease maintaining speed but will not turn off. The driver may tell the system to resume speed, whereupon it will return to the speed it was maintaining before braking and resume maintenance of that speed.

Hazard Identification

- Use historical safety experience, lessons learned, trouble reports, hazard analyses, and accident and incident files.
- Look at published lists, checklists, standards, and codes of practice.
- Examine basic energy sources, flows, high–energy items, hazardous materials (fuels, propellants, lasers, explosives, toxic substances, and pressure systems).
- Look at potential interface problems such as material incompatibilities, possibilities for inadvertent activation, contamination, and adverse environmental scenarios.
- Review mission and basic performance requirements including environments in which operations will take place. Look at all possible system uses, all modes of operation, all possible environments, and all times during operation.

Hazard Identification (2)

- Examine human–machine interface.
- Look at transition phases, nonroutine operating modes, system changes, changes in technical and social environment, and changes between modes of operation.
- Use scientific investigation of physical, chemical, and other properties of system.
- Think through entire process, step by step, anticipating what might go wrong, how to prepare for it, and what to do if the worst happens.

Specifying Safety Constraints

- Most software requirements only specify nominal behavior
 - Need to specify off–nominal behavior
 - Need to specify what software must NOT do
- What must not do is not inverse of what must do
- Derive from system hazard analysis

HAZARD	DESIGN CONSTRAINT
Train starts with door open.	Train must not be capable of moving with any door open.
Door opens while train is in motion.	Doors must remain closed while train is in motion.
Door opens while improperly aligned with station platform.	Door must be capable of opening only after train is stopped and properly aligned with platform unless emergency exists (see below).
Door closes while someone is in doorway.	Door areas must be clear before door closing begins.
Door that closes on an obstruction does not reopen or reopened door does not reclose.	An obstructed door must reopen to permit removal of obstruction and then automatically reclose.
Doors cannot be opened for emergency evacuation.	Means must be provided to open doors anywhere when the train is stopped for emergency evacuation.

Example ATC Approach Control

HAZARDS	REQUIREMENTS/CONSTRAINTS
1. A pair of controlled aircraft violate minimum separation standards.	1a. ATC shall provide advisories that maintain safe separation between aircraft. 1b. ATC shall provide conflict alerts.
2. A controlled aircraft enters an unsafe atmospheric region. (icing conditions, windshear areas, thunderstorm cells)	2a. ATC must not issue advisories that direct aircraft into areas with unsafe atmospheric conditions. 2b. ATC shall provide weather advisories and alerts to flight crews. 2c. ATC shall warn aircraft that enter an unsafe atmospheric region.
3. A controlled aircraft enters restricted airspace without authorization.	3a. ATC must not issue advisories that direct an aircraft into restricted airspace unless avoiding a greater hazard. 3b. ATC shall provide timely warnings to aircraft to prevent their incursion into restricted airspace.
4. A controlled aircraft gets too close to a fixed obstacle or terrain other than a safe point of touchdown on assigned runway.	4. ATC shall provide advisories that maintain safe separation between aircraft and terrain or physical obstacles.
5. A controlled aircraft and an intruder in controlled airspace violate minimum separation standards.	5. ATC shall provide alerts and advisories to avoid intruders if at all possible.
6. Loss of controlled flight or loss of airframe integrity.	6a. ATC must not issue advisories outside the safe performance envelope of the aircraft. 6b. ATC advisories must not distract or disrupt the crew from maintaining safety of flight. 6c. ATC must not issue advisories that the pilot or aircraft cannot fly or that degrade the continued safe flight of the aircraft. 6d. ATC must not provide advisories that cause an aircraft to fall below the standard glidepath or intersect it at the wrong place.

Classic Hazard Level Matrix

		SEVERITY			
		I Catastrophic	II Critical	III Marginal	IV Negligible
LIKELIHOOD	A Frequent	I-A	II-A	III-A	IV-A
	B Moderate	I-B	II-B	III-B	IV-B
	C Occasional	I-C	II-C	III-C	IV-C
	D Remote	I-D	II-D	III-D	IV-D
	E Unlikely	I-E	II-E	III-E	IV-E
	F Impossible	I-F	II-F	III-F	IV-F

Another Example Hazard Level Matrix

	A Frequent	B Probable	C Occasional	D Remote	E Improbable	F Impossible
Catastrophic I	Design action required to eliminate or control hazard 1	Design action required to eliminate or control hazard 2	Design action required to eliminate or control hazard 3	Hazard must be controlled or hazard probability reduced 4	▲ 9	▲ 12
Critical II	Design action required to eliminate or control hazard 3	Design action required to eliminate or control hazard 4	Hazard must be controlled or hazard probability reduced 6	Hazard control desirable if cost effective 7	Assume will not occur 12	Impossible occurrence 12
Marginal III	Design action required to eliminate or control hazard 5	Hazard must be controlled or hazard probability reduced 6	Hazard control desirable if cost effective 8	Normally not cost effective 10	12	12
Negligible IV	10	11	12	12	▼ 12	▼ 12

Negligible hazard

Risk Assessment

Hazard Level Assessment:

- Not feasible for complex human/computer–controlled systems
 - No way to determine likelihood, even qualitatively
 - Almost always involves new designs and new technology
- Severity is usually adequate (and that can be determined) to determine effort to spend on eliminating or mitigating hazard.

System Risk Assessment:

- Again, not feasible.
- May be possible to establish qualitative criteria to evaluate potential risk to make deployment or technology decisions, but will depend on system.

Example Qualitative Hazard Level Assessment

AATT Safety Criterion:

The introduction of AATT tools will not degrade safety from the current level.

Risk assessment for each tool based on:

- Severity of worst possible loss associated with tool
- Likelihood that introduction of tool will reduce current safety level of ATC system.

Example Likelihood Level (2)

- Safety margins
 - Low: Insignificant or no change
 - Medium: Minor change
 - High: Significant change
- Potential for reducing situation awareness
 - Low: Insignificant or no change
 - Medium: Minor change
 - High: Significant change
- Skills currently used and those necessary to backup and monitor new decision support tools
 - Low: Insignificant or no change
 - Medium: Minor change
 - High: Significant change
- Introduction of new failure modes and hazard causes
 - Low: New tools have same function and failure modes as system components they are replacing
 - Medium: Introduced but well understood and effective mitigation measures can be designed
 - High: Introduced and cannot be classified under medium
- Effect of software on current system hazard mitigation measures
 - Low: Cannot render ineffective
 - High: Can render ineffective
- Need for new system hazard mitigation measures
 - Low: Potential software errors will not require
 - High: Potential software errors could require

Hazard Log Information

- System, subsystem, unit
- Description
- Cause(s)
- Possible effects, effect on system
- Category (hazard level)
- Safety requirements and design constraints
- Corrective or preventative measures, possible safeguards, recommended action
- Operational phase when hazardous
- Responsible group or person for ensuring safeguards provided.
- Tests (verification) to be undertaken to demonstrate safety.
- Other proposed and necessary actions
- Status of hazard resolution process.

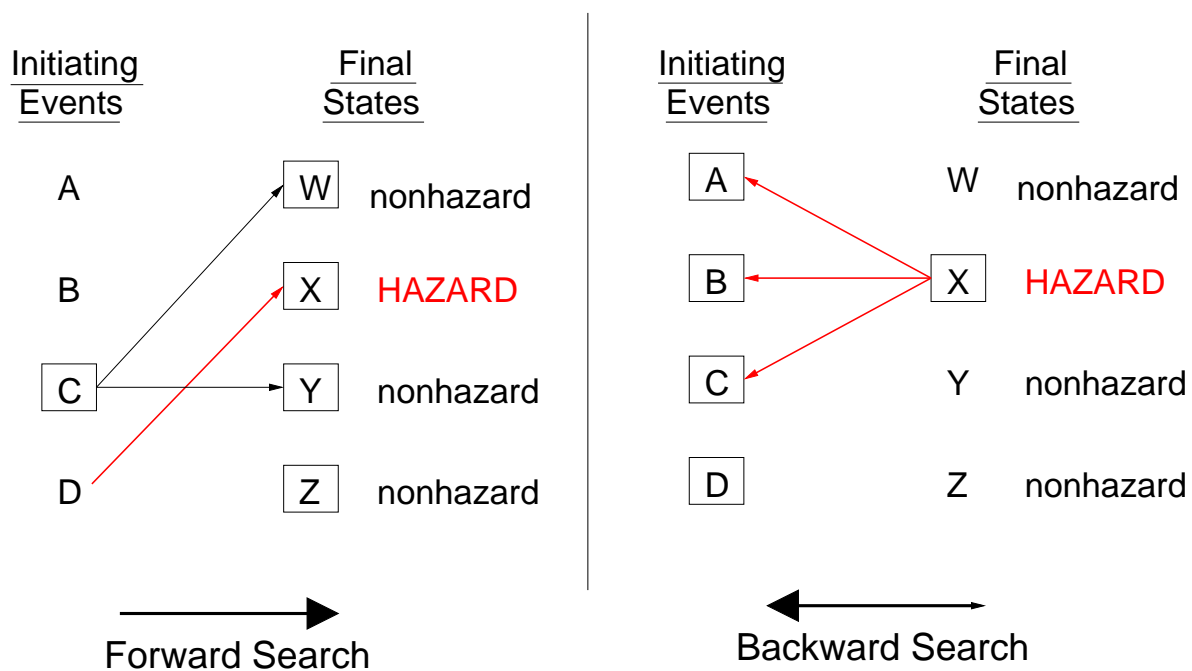
System Hazard Analysis

- Builds on PHA as a foundation (expands PHA)
- Considers system as a whole and identifies how
 - system operation
 - interfaces and interactions between subsystems
 - interface and interactions between system and operators
 - component failures and normal (correct) behaviorcould contribute to system hazards.
- Refines high-level safety design constraints
- Validates conformance of system design to design constraints
- Traces safety design constraints to individual components.
(based on functional decomposition and allocation)

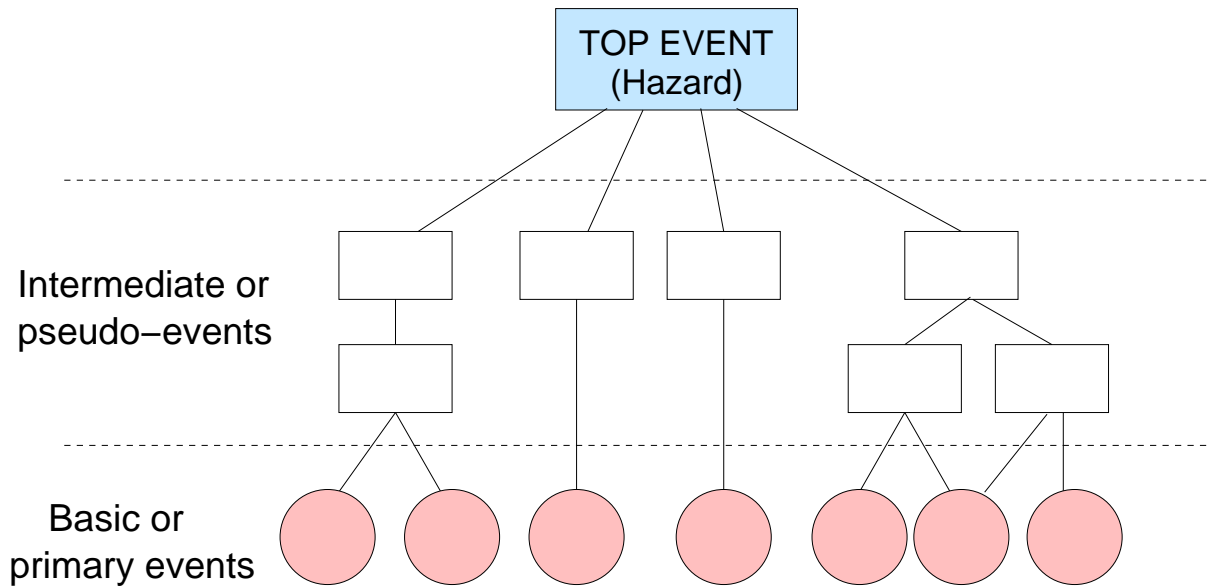
Hazard (Causal) Analysis

- "Investigating an accident before it happens"
- Requires
 - An accident model
 - A system design model (even if only in head of analyst)
- Almost always involves some type of search through the system design (model) for states or conditions that could lead to system hazards.
 - Forward
 - Backward
 - Top-down
 - Bottom-up
- Can be used to refine high-level safety constraints into more detailed constraints.

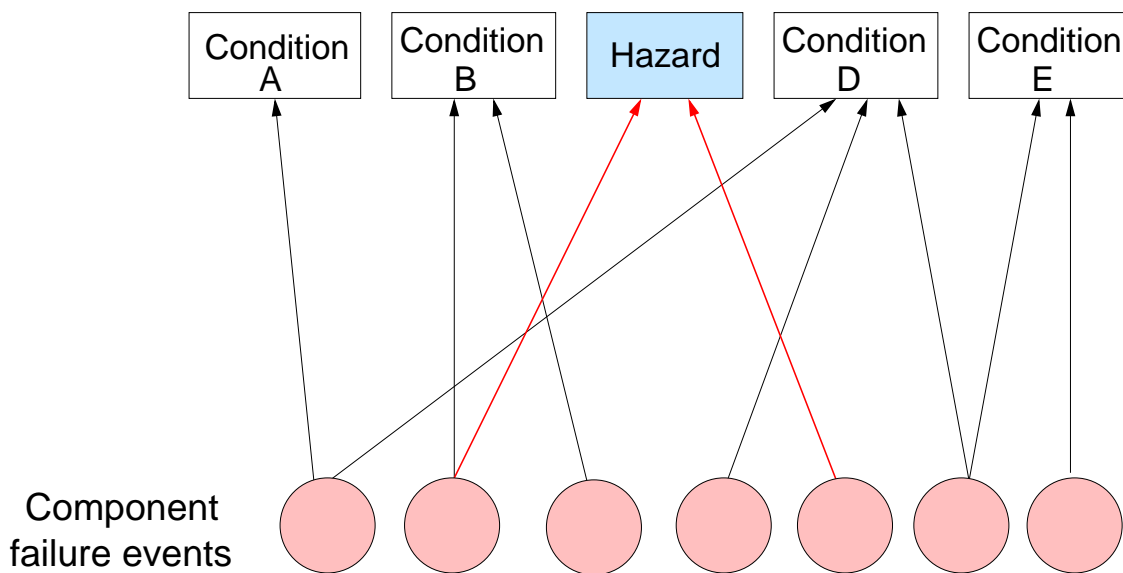
Forward vs. Backward Search



Top-Down



Bottom-Up



Fault Tree Analysis

- Developed originally in 1961 for Minuteman.
- Top-down search method.
- Based on converging chains-of-events accident model.
- Tree is simply a record of results; analysis done in head.
- FT can be written as Boolean expression and simplified to show specific combinations of identified basic events sufficient to cause the undesired top event (hazard).
- If want quantified analysis and individual probabilities for all basic events are known, frequency of top event can be calculated.

Example:

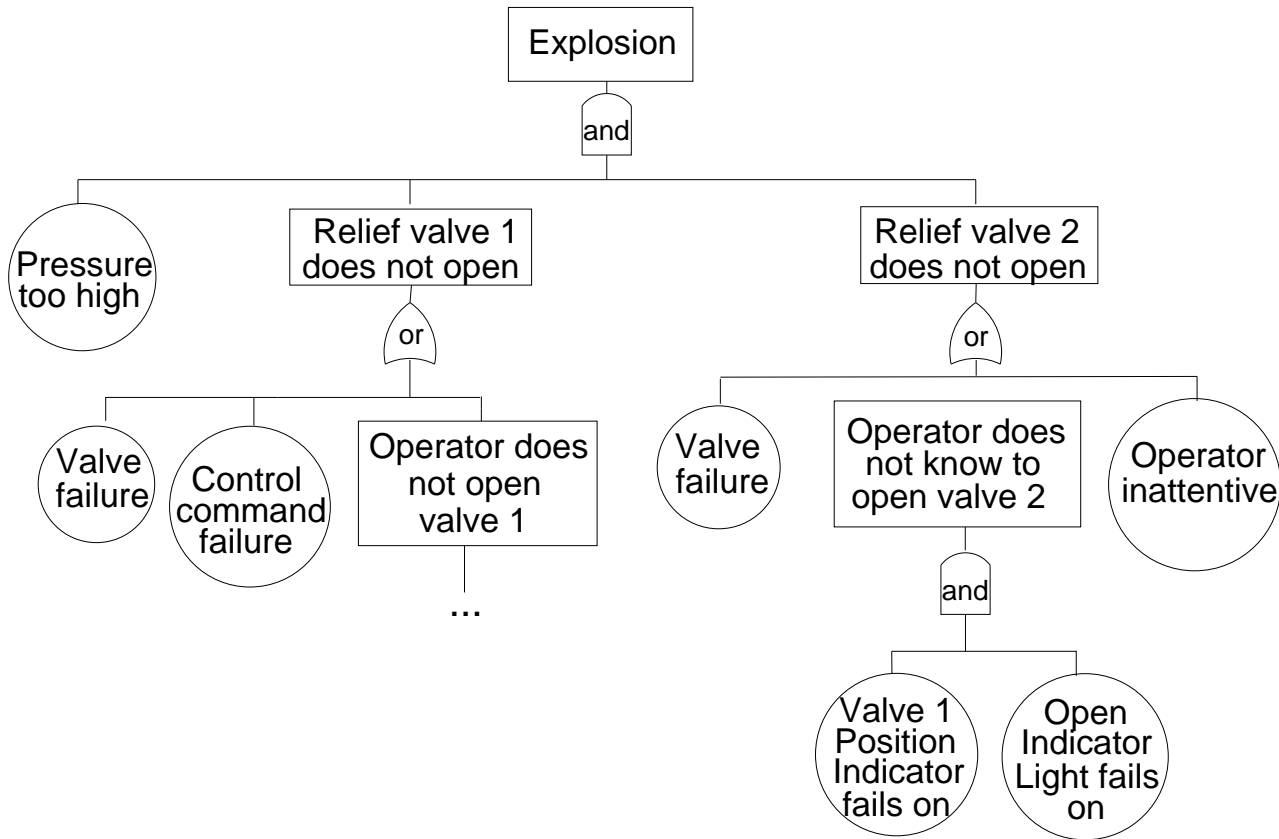
Hazard: Explosion

Design:

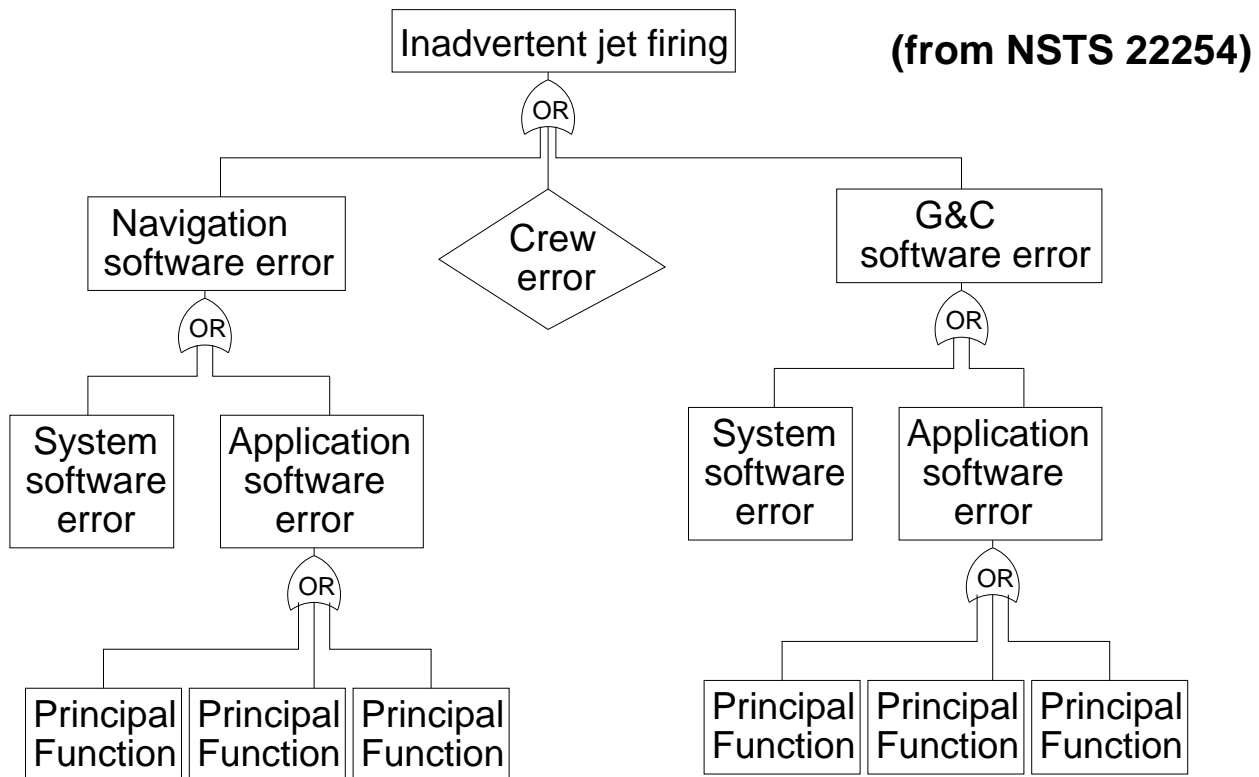
System includes a relief valve opened by an operator to protect against overpressurization. A secondary valve is installed as backup in case the primary valve failed. The operator must know if the first valve does not open so the second valve can be activated.

Operator console contains both a primary valve position indicator light and a primary valve open indicator light.

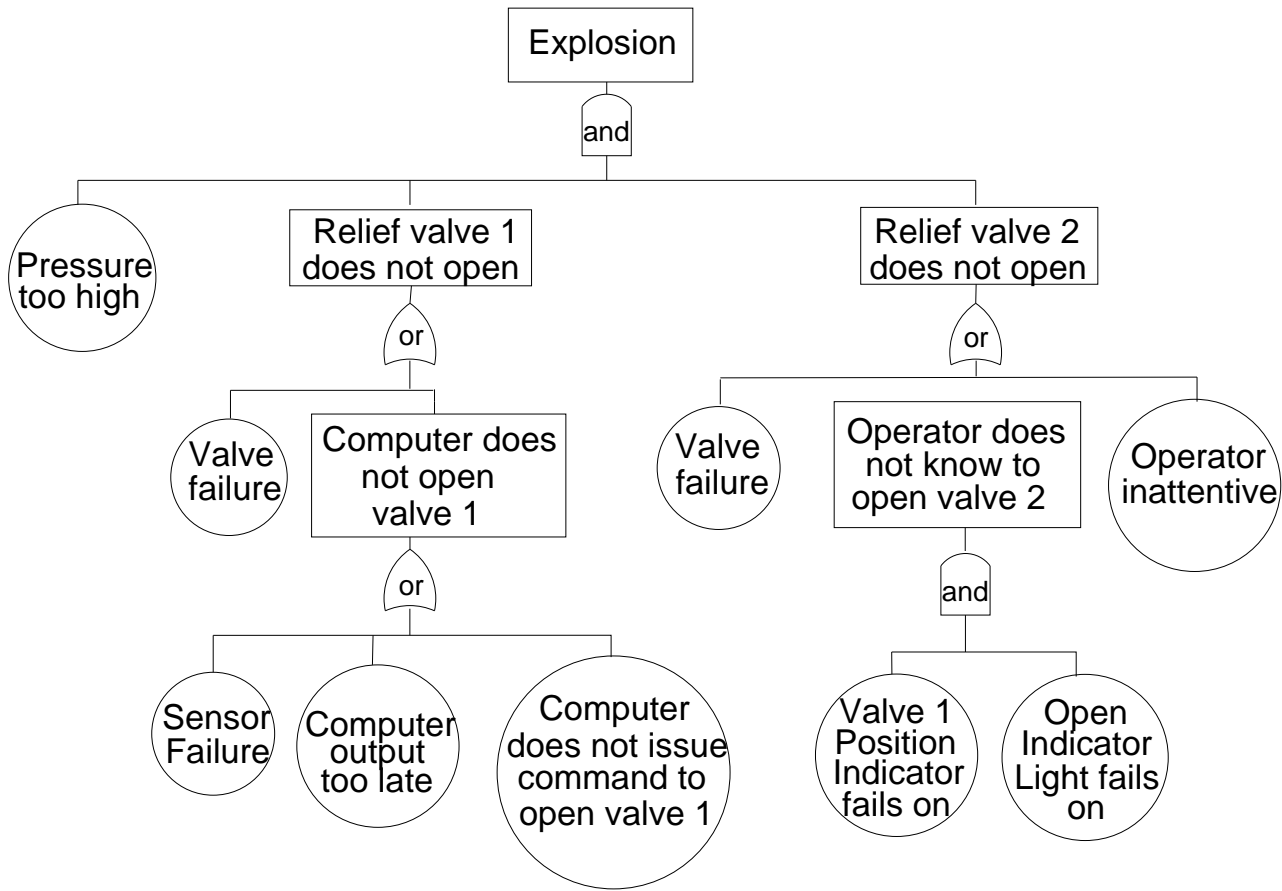
Fault Tree Example



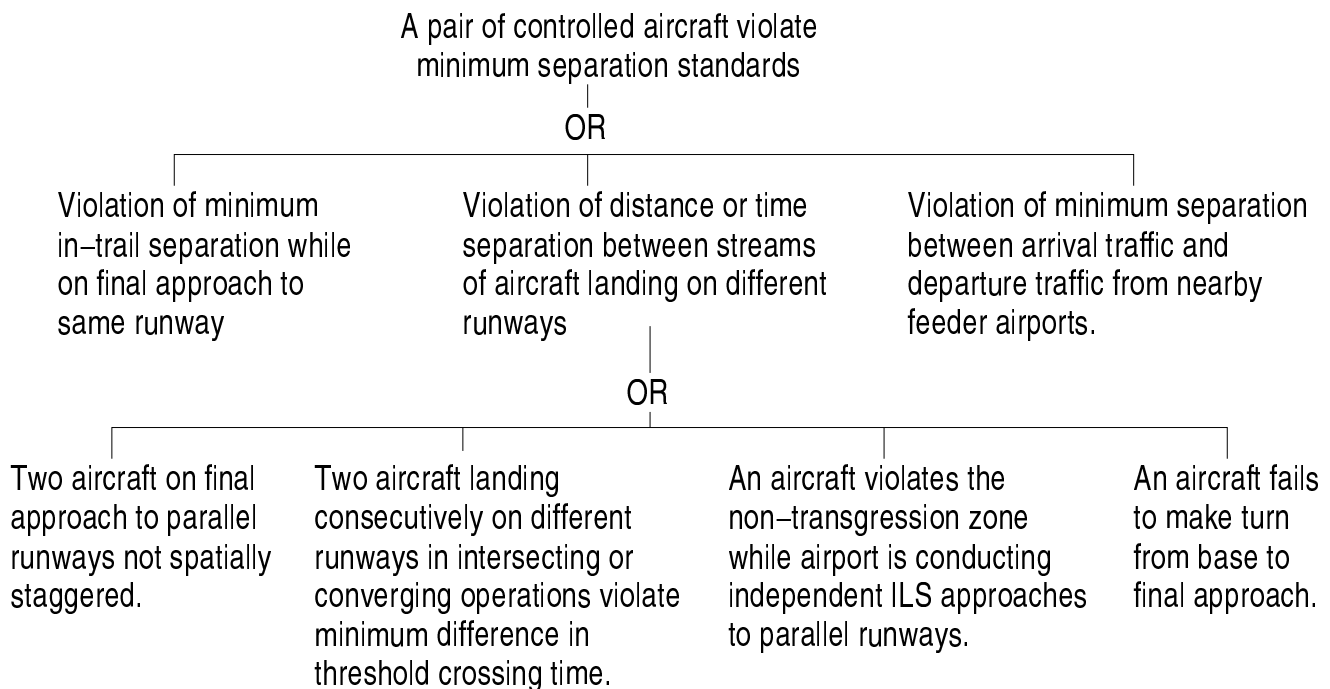
THE WRONG WAY TO DO IT!!



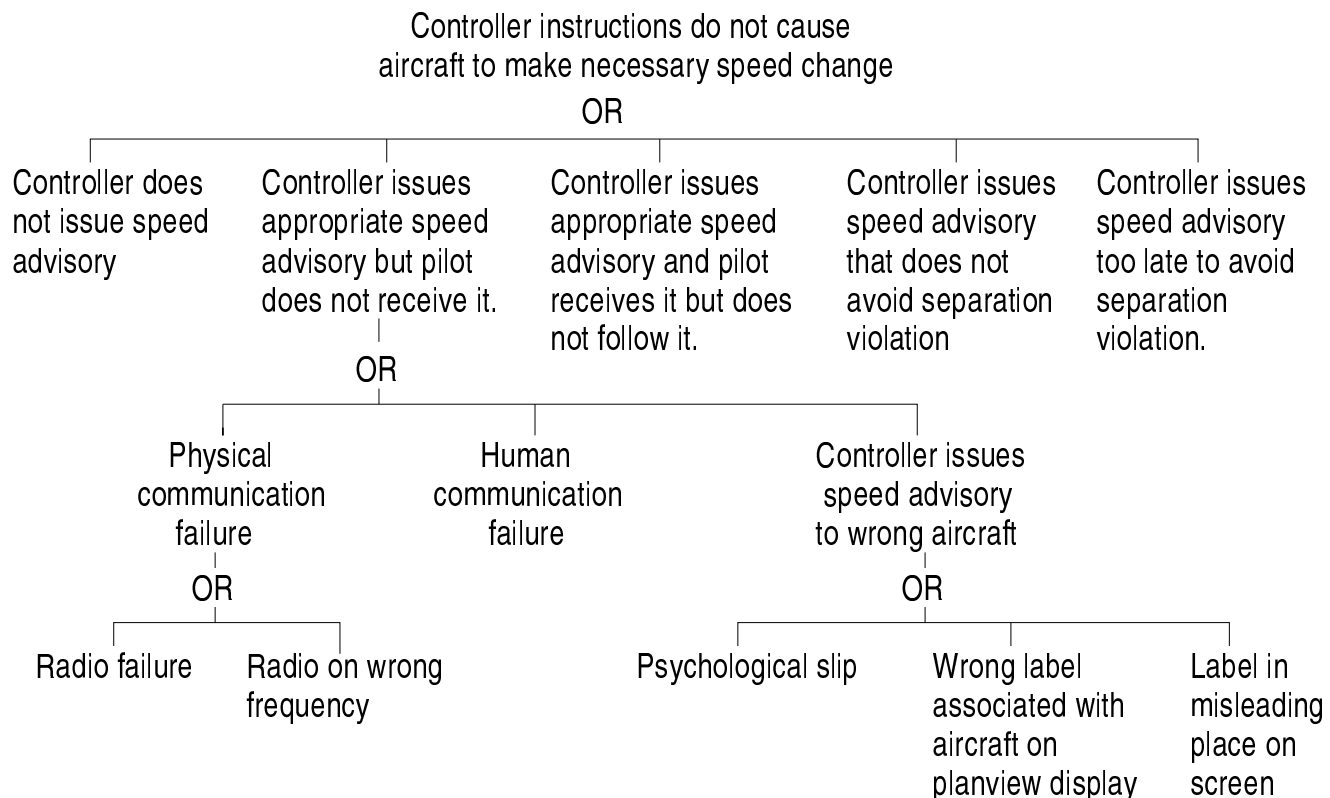
Fault Tree Example



Example Fault Tree for ATC Arrival Traffic



Example Fault Tree for ATC Arrival Traffic (2)



FTA Evaluation

- Graphical format helps in understanding system and relationship between events.
- Can be useful in tracing hazards to software interface and identifying potentially hazardous software behavior.
- Little guidance on deciding what to include
- Tends to concentrate on failures, but does not have to do so
- Quantitative evaluation may be misleading and may lead to accidents.

"On U.S. space programs where FTA (and FMEA) were used, 35% of actual in-flight malfunctions were not identified or were not identified as credible."

(list of aircraft accidents with risk of 10^{-9} or greater)

Example of unrealistic risk assessment contributing to an accident

System design:

Previous overpressurization example

Events:

The open position indicator light and open indicator light both illuminated. However, the primary valve was NOT open, and the system exploded.

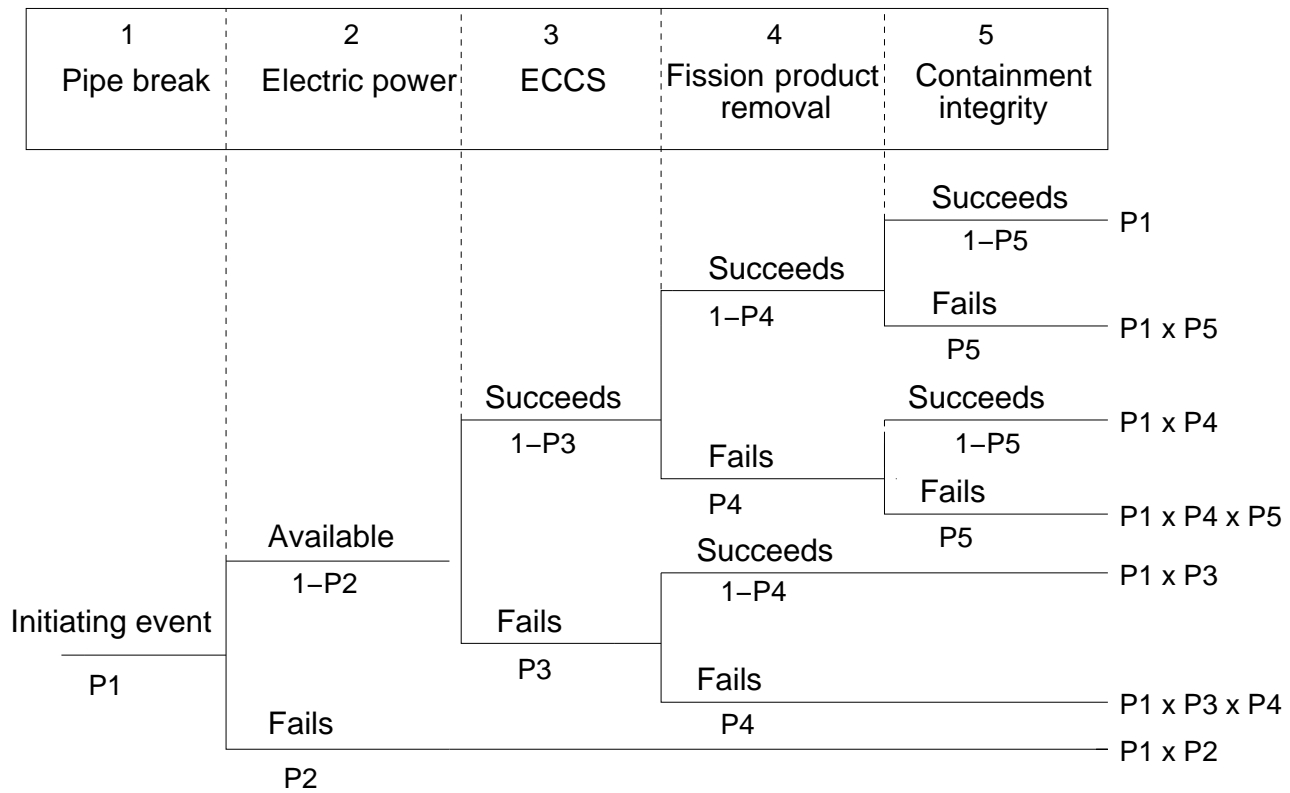
Causal Factors:

Post-accident examination discovered the indicator light circuit was wired to indicate presence of power at the valve, but it did not indicate valve position. Thus, the indicator showed only that the activation button had been pushed, not that the valve had opened. An extensive quantitative safety analysis of this design had assumed a low probability of simultaneous failure for the two relief valves, but ignored the possibility of design error in the electrical wiring; the probability of design error was not quantifiable. No safety evaluation of the electrical wiring was made; instead confidence was established on the basis of the low probability of coincident failure of the two relief valves.

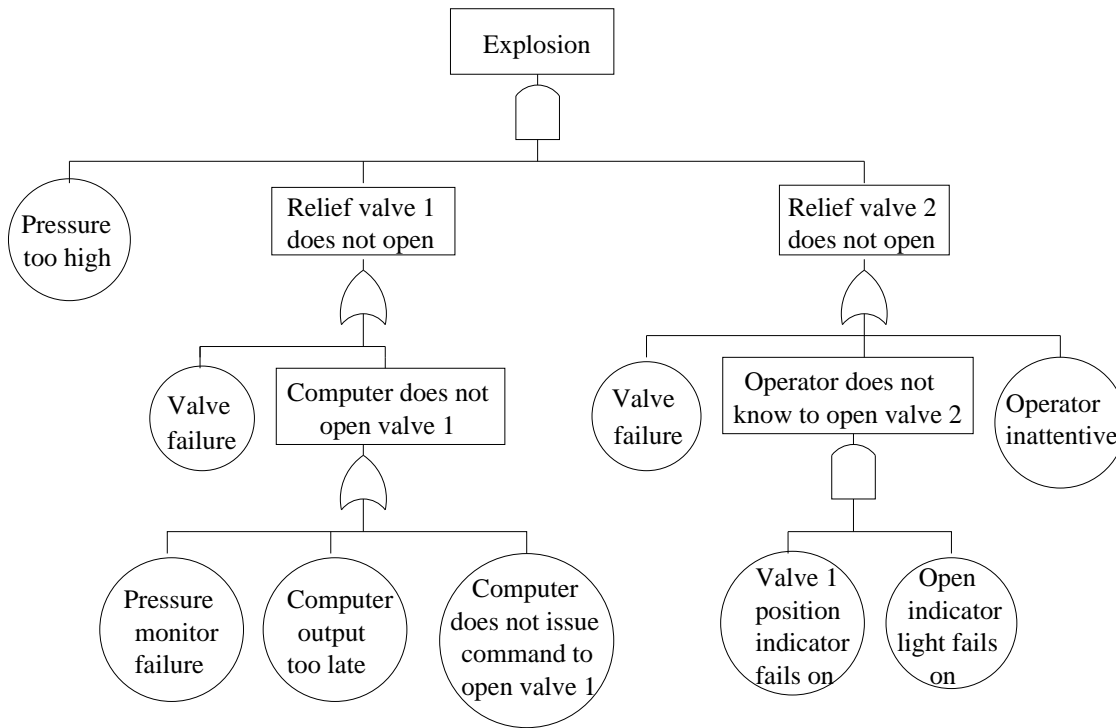
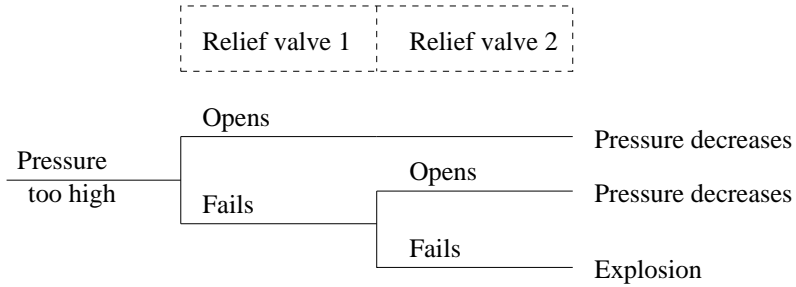
Event Tree Analysis

- Developed for and used primarily for nuclear power.
- Underlying single chain of events model of accidents.
- Forward search
- Simply another form of decision tree.
- Problems with dependent events.

Event Tree Example



Event Trees vs. Fault Trees



ETA Evaluation

- Events trees are better at handling ordering of events but fault trees better at identifying and simplifying event scenarios.
- Practical only when events can be ordered in time (chronology of events is stable) and events are independent of each other.
- Most useful when have a protection system.
- Can become exceedingly complex and require simplification.

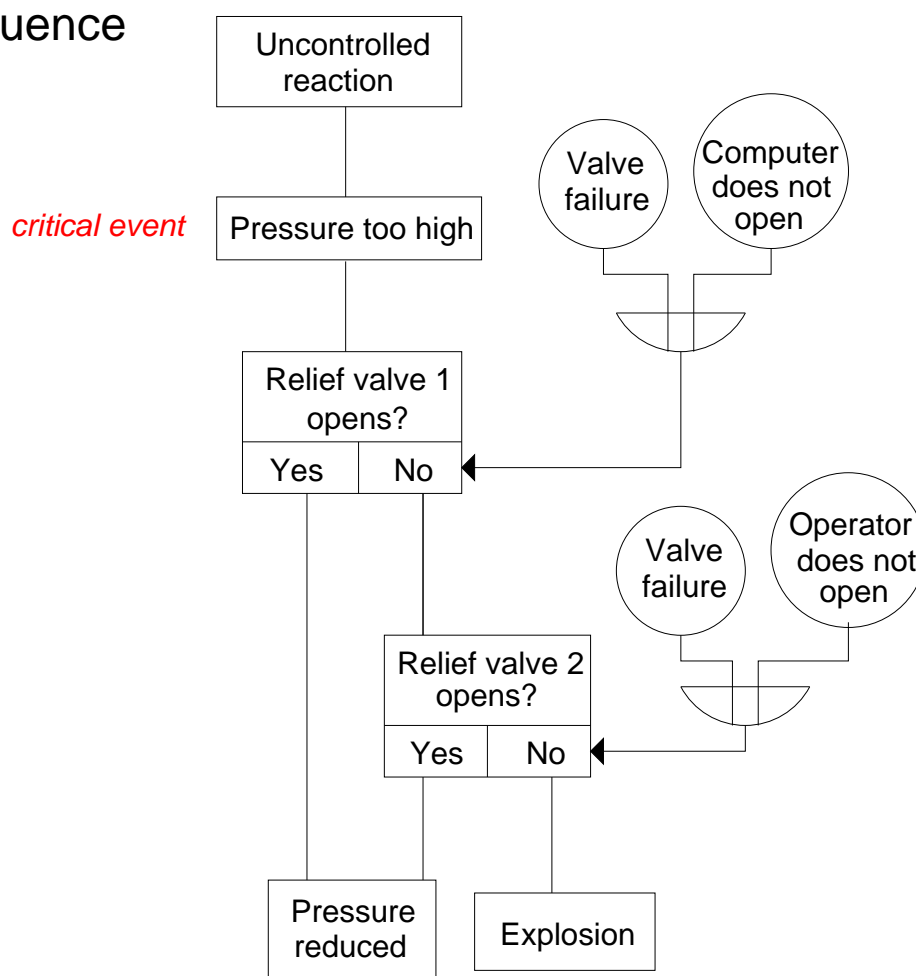
ETA Evaluation (2)

- Separate tree required for each initiating event.
 - Difficult to represent interactions between events
 - Difficult to consider effects of multiple initiating events.
- Defining functions across top of event tree and their order is difficult.
- Depends on being able to define set of initiating events that will produce all important accident sequences.
 - Probably most useful in nuclear power plants where
 - all risk associated with one hazard (overheating of fuel)
 - designs are fairly standard
 - large reliance on protection systems and shutdown systems.

Cause-Consequence Analysis

- Used primarily in Europe.
- A combination of forward and top-down search.
 - Basically a fault tree and event tree attached to each other
- Again based on converging chain-of-events.
- Diagrams can become unwieldy.
- Separate diagrams required for each initiating event.

Cause-Consequence Diagram



FMEA or FMECA

Failure Modes and Effects (Criticality) Analysis

- Developed to predict equipment reliability.
- Forward search based on underlying single chain-of-events and failure models (like event trees).
- Initiating events are failures of individual components.

HAZOP: Hazard and Operability Analysis

- Based on model of accidents that assumes they are caused by deviations from design or operating intentions.
- Purpose is to identify all possible deviations from the design's expected operation and all hazards associated with these deviations.
- Unlike other techniques, works on a concrete model of plant (e.g., piping and wiring diagram).
- Applies a set of guidewords to the plant diagram.

HAZOP Guidewords

<i>Guideword</i>	<i>Meaning</i>
NO, NOT, NONE	The intended result is not achieved, but nothing else happens (such as no forward flow when there should be)
MORE	More of any relevant physical property than there should be (such as higher pressure, higher temperature, higher flow, or higher viscosity).
LESS	Less of a relevant physical property than there should be.
AS WELL AS	An activity occurs in addition to what was intended, or more components are present in the system than there should be (such as extra vapors or solids or impurities, including air, water, acids, corrosive products).
PART OF	Only some of the design intentions are achieved (such as only one of two components in a mixture).
REVERSE	The logical opposite of what was intended occurs (such as backflow instead of forward flow).
OTHER THAN	No part of the intended result is achieved, and something completely different happens (such as the flow of the wrong material).

Example Entry in a HAZOP report

<i>Guide Word</i>	<i>Deviation</i>	<i>Possible Causes</i>	<i>Possible Consequences</i>
NONE	No flow	<ol style="list-style-type: none"> 1. Pump failure 2. Pump suction filter blocked 3. Pump isolation valve closed. 	<ol style="list-style-type: none"> 1. Overheating in heat exchanger. 2. Loss of feed to reactor.

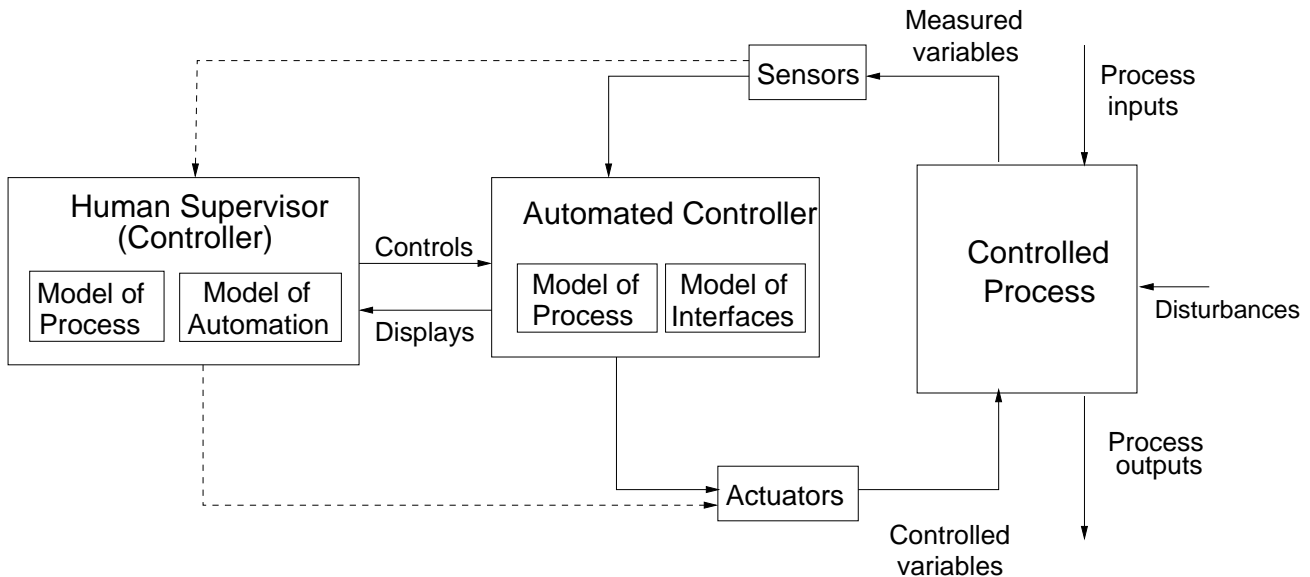
Task and Human Error Analyses

- Qualitative Techniques
 - Break down tasks into a sequence of steps.
 - Investigate potential deviations and their consequences.
 - Quantitative Techniques
 - Assign probabilities for various types of human error.
 - Most effective in simple systems where tasks routine.
 - Not effective for cognitively complex tasks operators often asked to perform today.
-

STAMP–Based Hazard Analysis (STPA)

- Supports a safety–driven design process where
 - Hazard analysis influences and shapes early design decisions
 - HA iterated and refined as design evolves
- Goals (same as any hazard analysis)
 - Identification of system hazards and related safety constraints necessary to ensure acceptable risk
 - Accumulation of information about how hazards can be violated, which is used to eliminate, reduce, and control hazards in system design, development, manufacturing, and operations.
- Process
 - Starts with identifying system requirements and design constraints necessary to maintain safety
 - Then assists in top–down refinement into requirements and constraints on individual components.

Basic Control Loop



Process models must contain:
Required relationship among process variables
Current state (values of process variables)
The ways the process can change state

• Inadequate Control Actions (enforcement of constraints)

- Design of control algorithm (process) does not enforce constraints
- Process models inconsistent, incomplete, or incorrect (lack of linkup)
 - o Flaw(s) in creation or updating process
 - o Inadequate or missing feedback
 - Not provided in system design
 - Communication flaw
 - Inadequate sensor operation (incorrect or no information provided)
 - o Time lags and measurement inaccuracies not accounted for
- Inadequate coordination among controllers and decision-makers (boundary and overlap areas)

• Inadequate Execution of Control Action

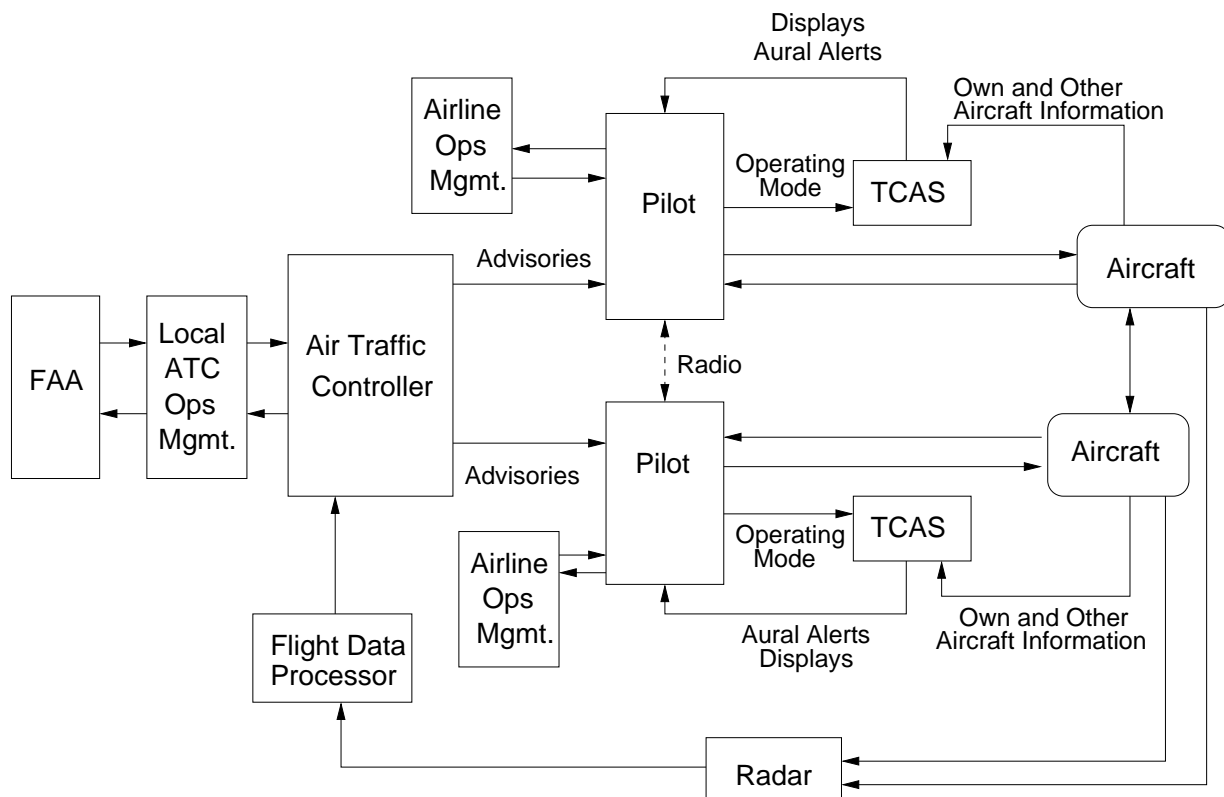
- Communication flaw
- Inadequate "actuator" operation
- Time lag

STPA – Step1: Identify hazards and translate into high-level requirements and constraints on behavior

TCAS Hazards

1. A near mid-air collision (NMAC)
(a pair of controlled aircraft violate minimum separation standards)
2. A controlled maneuver into the ground
3. Loss of control of aircraft
4. Interference with other safety-related aircraft systems
5. Interference with ground-based ATC system
6. Interference with ATC safety-related advisory

STPA – Step 2: Define basic control structure



Aircraft Components (e.g., transponders, antennas):

Execute control maneuvers
Receive and send messages to/from other aircraft
etc.

TCAS:

Receive and update information about its own and other aircraft
Analyze information received and provide pilot with

1. Information about where other aircraft in the vicinity are located
2. An escape maneuver to avoid potential NMAC threats

Pilot:

Maintain separation between own and other aircraft using visual scanning
Monitor TCAS displays and implement TCAS escape maneuvers
Follow ATC advisories

Airline Operations Management:

Provide procedures for using TCAS and following TCAS advisories
Train pilots
Audit pilot performance

Air Traffic Controller:

Maintain separation between aircraft in the controlled airspace by providing advisories (control actions) for the pilot to follow

ATC Operations Management:

Provide procedures, train controllers, audit performance of controllers and of the overall collision avoidance system.

STPA – Step 3: Identify potential inadequate control actions that could lead to hazardous process state

In general:

1. A required control action is not provided
2. An incorrect or unsafe control action is provided.
3. A potentially correct or inadequate control action is provided too late (at the wrong time)
4. A correct control action is stopped too soon

For the NMAC hazard:

TCAS:

1. The aircraft are on a near collision course and TCAS does not provide an RA
2. The aircraft are in close proximity and TCAS provides an RA that degrades vertical separation
3. The aircraft are on a near collision course and TCAS provides an RA too late to avoid an NMAC
4. TCAS removes an RA too soon.

Pilot:

1. The pilot does not follow the resolution advisory provided by TCAS (does not respond to the RA)
2. The pilot incorrectly executes the TCAS resolution advisory.
3. The pilot applies the RA but too late to avoid the NMAC
4. The pilot stops the RA maneuver too soon.

STPA – Step 4: Determine how potentially hazardous control actions could occur.

Eliminate from design or control or mitigate in design or operations

- Can use a concrete model in SpecTRM–RL
 - Assists with communication and completeness of analysis
 - Provides a continuous simulation and analysis environment to evaluate impact of faults and effectiveness of mitigation features.

TCAS does not provide an RA when required to avoid an NMAC

Unit is not operational

Pilot did not turn on

Self-monitor turns off TCAS unit

Component failure

TCAS does not perceive conflict

Current location of other aircraft is incorrect

TCAS thinks other aircraft on the ground

Incorrect altitude provided to TCAS

Uneven terrain

TCAS puts other aircraft outside of protected volume

Location of own aircraft incorrect

Altimeter error

Delay in receipt of information about altitude change

Trajectory of other aircraft computed incorrectly

Trajectory of own aircraft computed incorrectly

Other aircraft does not have an operating transponder

...

Step 4a: Augment control structure with process models for each control component

Step 4b: For each of inadequate control actions, examine parts of control loop to see if could cause it.

Guided by set of generic control loop flaws

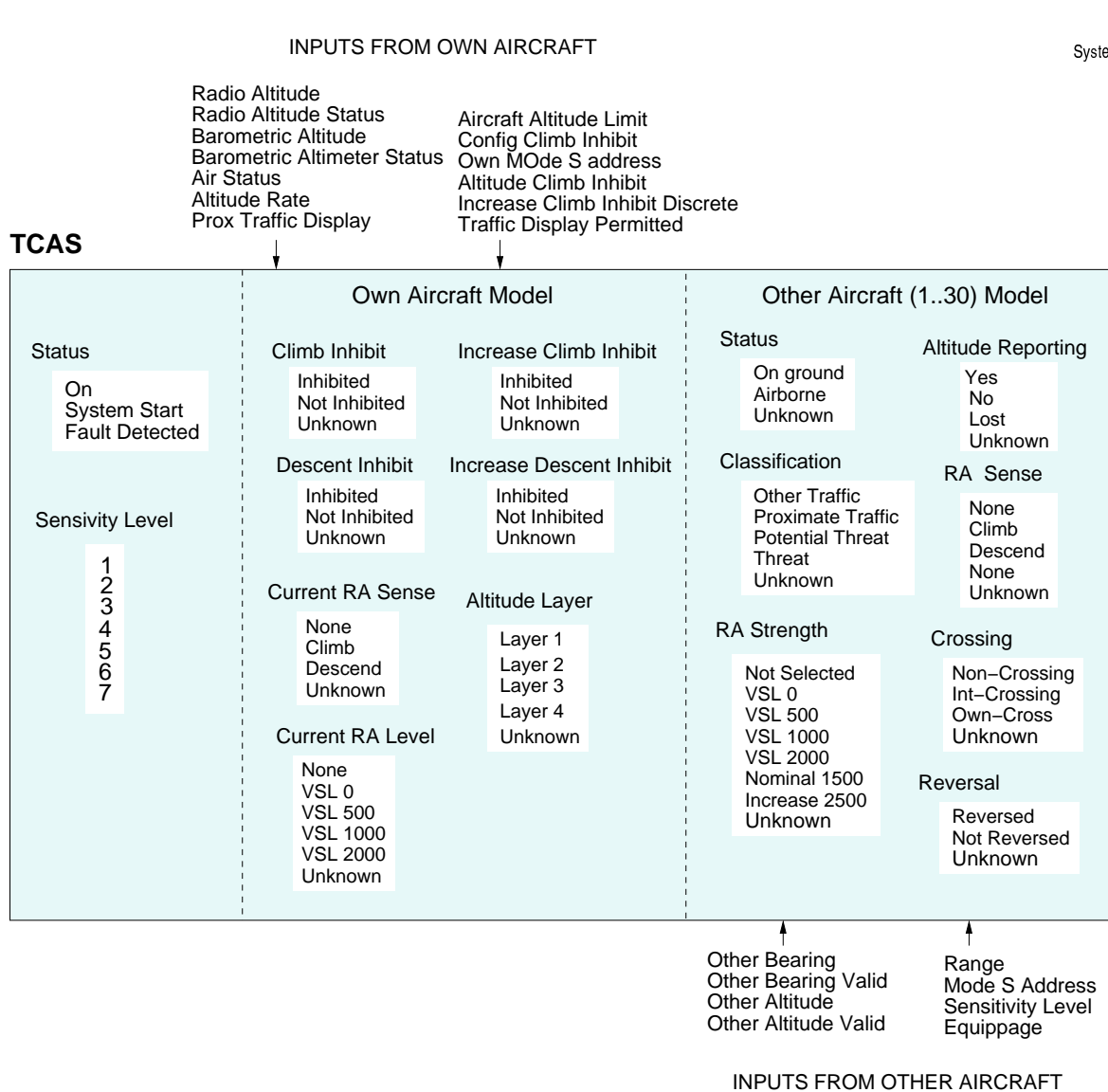
Where human or organization involved must evaluate:

Context in which decisions made

Behavior-shaping mechanisms (influences)

in order to understand types of reasons for potentially unsafe decisions and to design controls and mitigation measures

Step 4c: Consider how designed controls could degrade over time



Comparisons with Traditional HA Techniques

- Top-down (vs. bottom-up like FMECA)
- Considers more than just component failures and failure events
- Guidance in doing analysis (vs. FTA)
- Handles dysfunctional interactions, software, management, etc.
- Concrete model (not just in head)
 - Not physical structure (HAZOP) but control (functional) structure
 - General model of inadequate control
 - HAZOP guidewords based on model of accidents being caused by deviations in system variables
 - Includes HAZOP model but more general
- Compared with TCAS II Fault Tree (MITRE)
STPA results more comprehensive

Requirements Analysis

Subsystem Hazard Analysis (SSHA)

- Examine subsystems to determine how their
 - Normal performance
 - Operational degradation
 - Functional failure
 - Unintended function
 - Inadvertent function (proper function but at wrong time or in wrong order)could contribute to system hazards.
- Determine how to satisfy design constraints in subsystem design.
- Validate the subsystem design satisfies safety design constraints and does not introduce previously unidentified hazardous system behavior.

Software Hazard Analysis

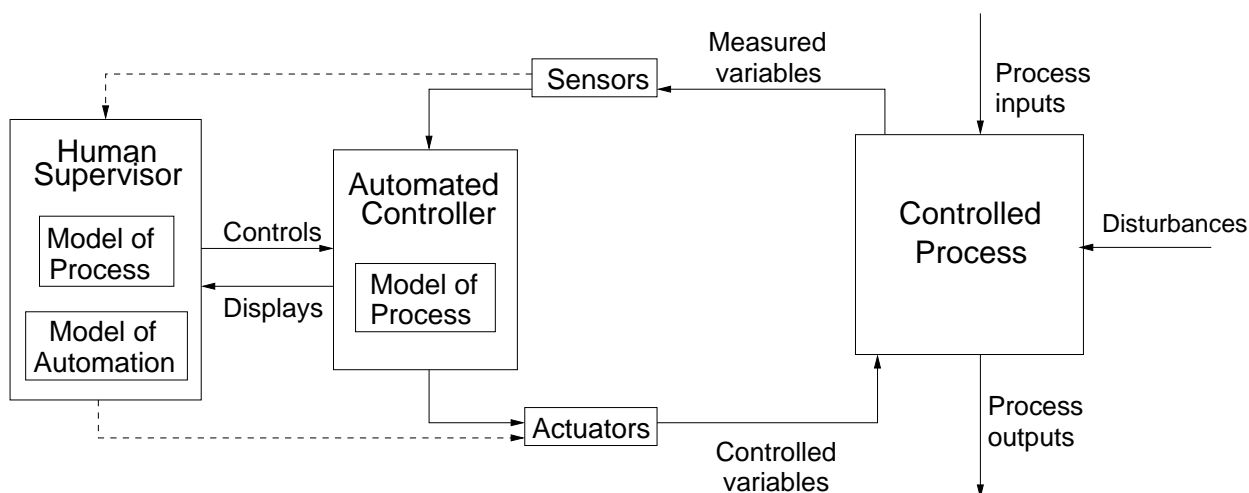
- A form of subsystem hazard analysis.
- Validate that specified software blackbox behavior (requirements) satisfies the system safety design constraints.
- Check specified software behavior satisfies general software system safety design criteria.
- Trace requirements into code.
- Must perform on ALL software, including COTS.

Requirements Validation

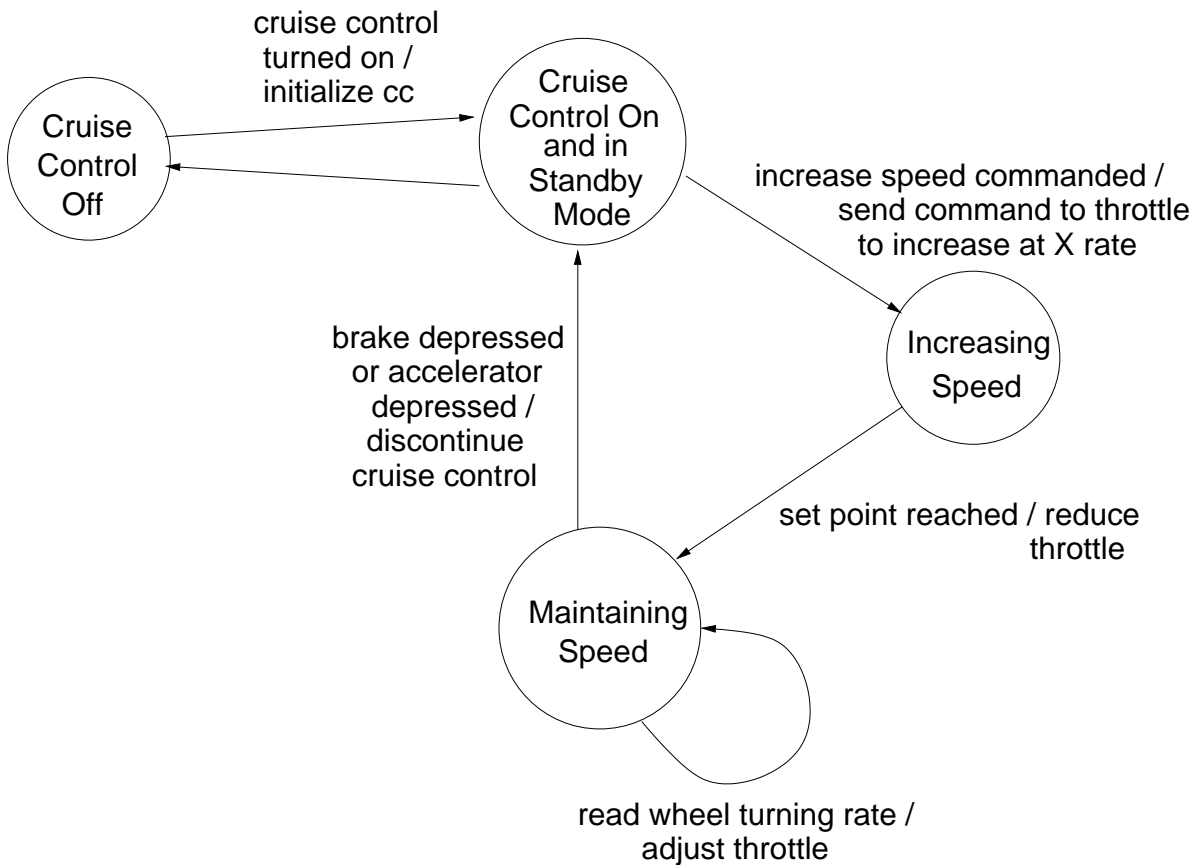
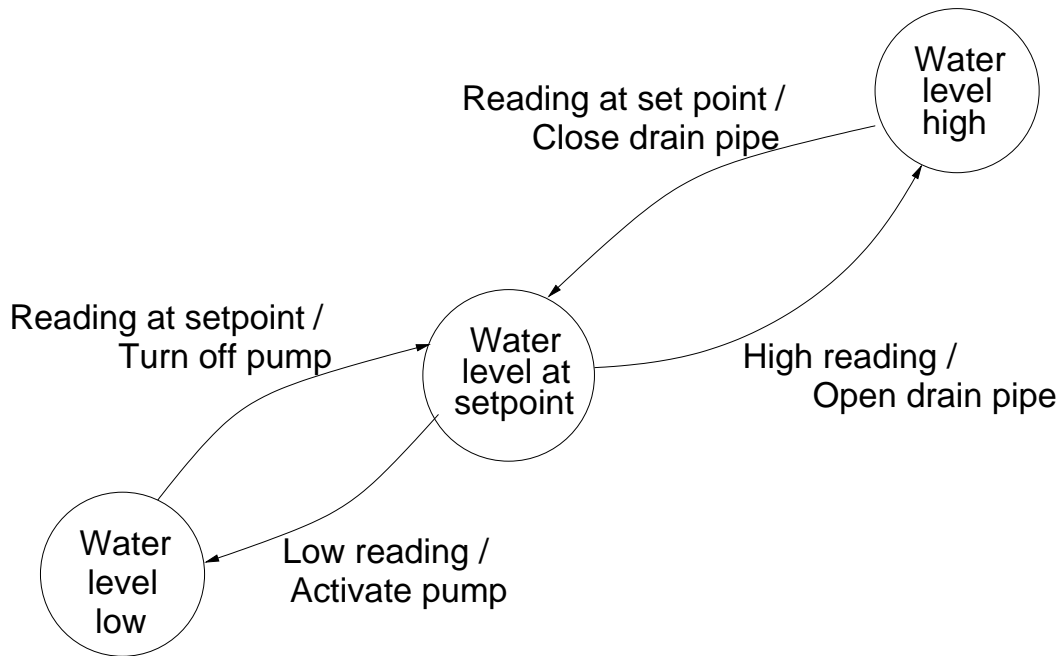
- Requirements are source of most operational errors and almost all the software contributions to accidents.
- Much of software safety effort therefore should focus on requirements.
- Problem is dealing with complexity
 - 1) Use blackbox models to separate external behavior from complexity of internal design to accomplish the behavior.
 - 2) Use abstraction and metamodels to handle large number of discrete states required to describe software behavior.
 - Do not have continuous math to assist us
 - But new types of state machine modeling languages drastically reduce number of states and transitions modeler needs to describe.

Requirements as State Machine Models

Define required blackbox behavior of software in terms of a state machine model of the process (plant).



Example of a State Machine Model



Requirements Completeness

- Most software–related accidents involve software requirements deficiencies.
- Accidents often result from unhandled and unspecified cases.
- We have defined a set of criteria to determine whether a requirements specification is complete.
- Derived from accidents and basic engineering principles.
- Validated (at JPL) and used on industrial projects.

Completeness: Requirements are sufficient to distinguish the desired behavior of the software from that of any other undesired program that might be designed.

Blackbox specifications

Start from a "blackbox" statement of software behavior:

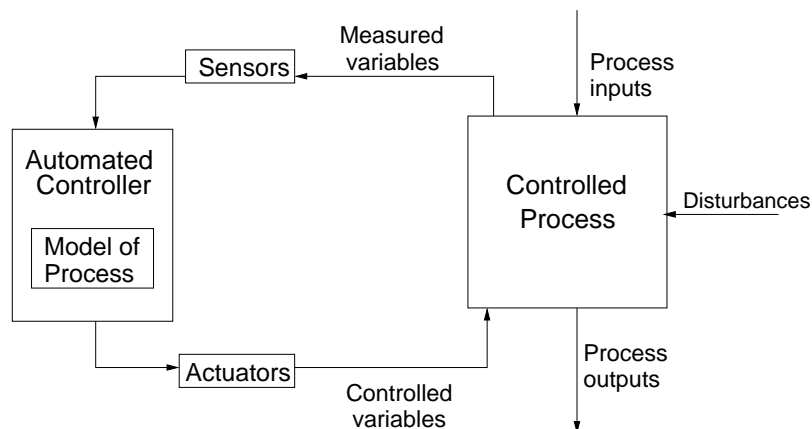
- Includes externally visible behavior only
 - Internal design decisions are not included.
 - Simplifies the specification and review by system experts
- In essence, the specification is the input to output function computed by the component, i.e., the transfer function.
- Specify:
 - Outputs
 - Triggers for outputs (externally observable conditions or events)

(not just inputs, e.g., may want to trigger an output on the passage of time)

Requirements Completeness Criteria

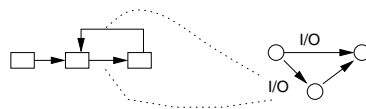
Completeness criteria define what needs to be specified about triggers, outputs, and the relationship between them.

- Defined in terms of a state machine model of the process (plant) model.
- Required blackbox behavior specified using the process model.



Requirements Completeness Criteria (2)

- How were criteria derived?
 - Mapped the parts of a control loop to a state machine



- Defined completeness for each part of state machine
 - States, inputs, outputs, transitions
 - Mathematical completeness
- Added basic engineering principles (e.g., feedback)
- Added what have learned from accidents

Requirements Completeness Criteria (3)

About 60 criteria in all including human–computer interaction.

(won't go through them all— they are in the book)

Startup, shutdown	Robustness
Mode transitions	Data age
Inputs and outputs	Latency
Value and timing	Feedback
Load and capacity	Reversibility
Environment capacity	Preemption
Failure states and transitions	Path Robustness
Human–computer interface	

Startup and State Completeness

Many accidents involve off–nominal processing modes, including startup and shutdown and handling unexpected inputs.

Examples of completeness criteria in this category:

- The internal software model of the process must be updated to reflect the actual process state at initial startup and after temporary shutdown.
- The maximum time the computer waits before the first input must be specified.
- There must be a response specified for the arrival of an input in any state, including indeterminate states.

Input and Output Variable Completeness

At blackbox interface, only time and value observable to software.

So triggers and outputs must be defined only as constants or as the value of observable events or conditions.

Criteria:

- All information from the sensors should be used somewhere in the specification.
- Legal output values that are never produced should be checked for potential specification incompleteness.

Trigger Event Completeness

- Behavior of computer defined with respect to assumptions about the behavior of the other parts of the system.
- A robust system will detect and respond appropriately to violations of these assumptions (such as unexpected inputs).
- Therefore, robustness of software built from specification will depend on completeness of specification of environmental assumptions.
 - There should be no observable events that leave the program's behavior indeterminate.
- Why need to document and check all assumptions?

Formal Robustness Criteria

- To be robust, the events that trigger state changes must satisfy the following:
 1. Every state must have a behavior (transition) defined for possible input.
 2. The logical OR of the conditions on every transition out of every state must form a tautology.
$$x < 5$$
$$x \geq 5$$
 3. Every state must have a software behavior (transition) defined in case there is no input for a given period of time (a timeout).
- Together these criteria guarantee handling input that are within range, out of range, and missing.

Nondeterminism Criterion

- The behavior of the requirements should be deterministic (only one possible transition out of a state is applicable at any time).
$$X > 0$$
$$X < 2$$
- We (and others) have tools to check specifications based on state machines for robustness, consistency, and nondeterminism.

NOTE: This type of mathematical completeness is NOT enough.

e.g., “*true*” is a mathematically complete, consistent, and deterministic specification.

Failure States and Transition Criteria

Need to completely specify:

- Off-nominal states and transitions
- Performance degradation
- Communication with operator about fail-safe behavior
- Partial shutdown and restart
- Hysteresis in transitions between off-nominal and nominal

Most accidents occur while in off-nominal processing modes.

Value and Timing Assumptions

Examples:

- All inputs should be checked and a response specified in the event of an out-of-range or unexpected value.
- All inputs must be fully bounded in time and the proper behavior specified in case the limits are violated.
- Minimum and maximum load assumptions ...
- A minimum-arrival-rate check should be required for each physically distinct communication path.

Software should have the capability to query its environment with respect to inactivity over a given communication path.

- Response to excessive inputs (violations of load assumptions) must be specified.

Environment Capacity Constraints

Examples:

- For the largest interval in which both input and output loads are assumed and specified, the absorption rate of the output environment must equal or exceed the input arrival rate.
- Contingency action must be specified when the output absorption rate limit will be exceeded.

Human-Computer Interface Criteria

- For every data item displayable to a human, must specify:
 1. What events cause this item to be displayed?
 2. What events cause item to be updated?
If so, what events should cause the update?
 3. What events should cause the display to disappear?
- For queues need to specify:
 1. Events to be queued
 2. Type and number of queues to be provided (alert and routine)
 3. Ordering scheme within queue (priority vs. time of arrival)
 4. Operator notification mechanism for items inserted in the queue.
 5. Operator review and disposal commands for queue entries.
 6. Queue entry deletion.

Data Age Criteria

- All inputs used in specifying output events must be properly limited in the time they can be used.
- Output commands that may not be able to be executed immediately must be limited in the time they are valid.
- Incomplete hazardous action sequences (transactions) should have a finite time specified after which the software should be required to cancel the sequence automatically and inform the operator.
- Revocation of partially completed transactions may require:
 1. Specification of multiple times and conditions under which varying automatic cancellation or postponement actions are taken without operator confirmation.
 2. Specification of operator warnings to be issued in case of such revocation.

Latency Criteria

- Latency is the time interval during which receipt of new information cannot change an output even though it arrives prior to output.
 - Influenced by hardware and software design (e.g., interrupt vs. polling)
 - Cannot be eliminated completely.
 - Acceptable length determined by controlled process.
- Subtle problems when considering latency of HCI data.
(see book for criteria)

Feedback Criteria

Basic feedback loops, as defined by the process control function, must be included in the requirements along with appropriate checks to detect internal or external failures or errors.

Examples:

- There should be an input that the software can use to detect the effect of any output on the process.
- Every output to which a detectable input is expected must have associated with it:
 1. A requirement to handle the normal response
 2. Requirements to handle a response that is missing, too late, too early, or has an unexpected value.

Path Criteria

- Paths between states are uniquely defined by the sequence of trigger events along the path.
- Transitions between modes are especially hazardous and susceptible to incomplete specification.

REACHABILITY

- Required states must be reachable from initial state.
- Hazardous states must not be reachable.
- Complete reachability analysis often impractical, but may be able to reduce search by focusing on a few properties or using backward search.
- Sometimes what is not practical in general case is practical in specific cases.

Path Criteria (2)

RECURRENT BEHAVIOR

- Most process control software is cyclic. May have some non-cyclic states (mode change, shutdown)
- Required sequences of events must be specified in and limited by transitions in a cycle.
- *Inhibiting state*: State from which output cannot be generated.
- There should be no states that inhibit later required outputs.

REVERSIBILITY

PREEMPTION

Path Criteria (3)

PATH ROBUSTNESS

Soft failure mode: The loss of ability to receive input X could inhibit the production of output Y

Hard failure mode: The loss of ability to receive input X will inhibit the production of output Y

- Soft and hard failure modes should be eliminated for all hazard reducing outputs.
- Hazard increasing outputs should have both soft and hard failure modes.
- Multiple paths should be provided for state changes that maintain safety.
- Multiple inputs or triggers should be required for paths from safe to hazardous states.

CONSTRAINT ANALYSIS

SpecTRM

Specification Tools and Requirements Methodology

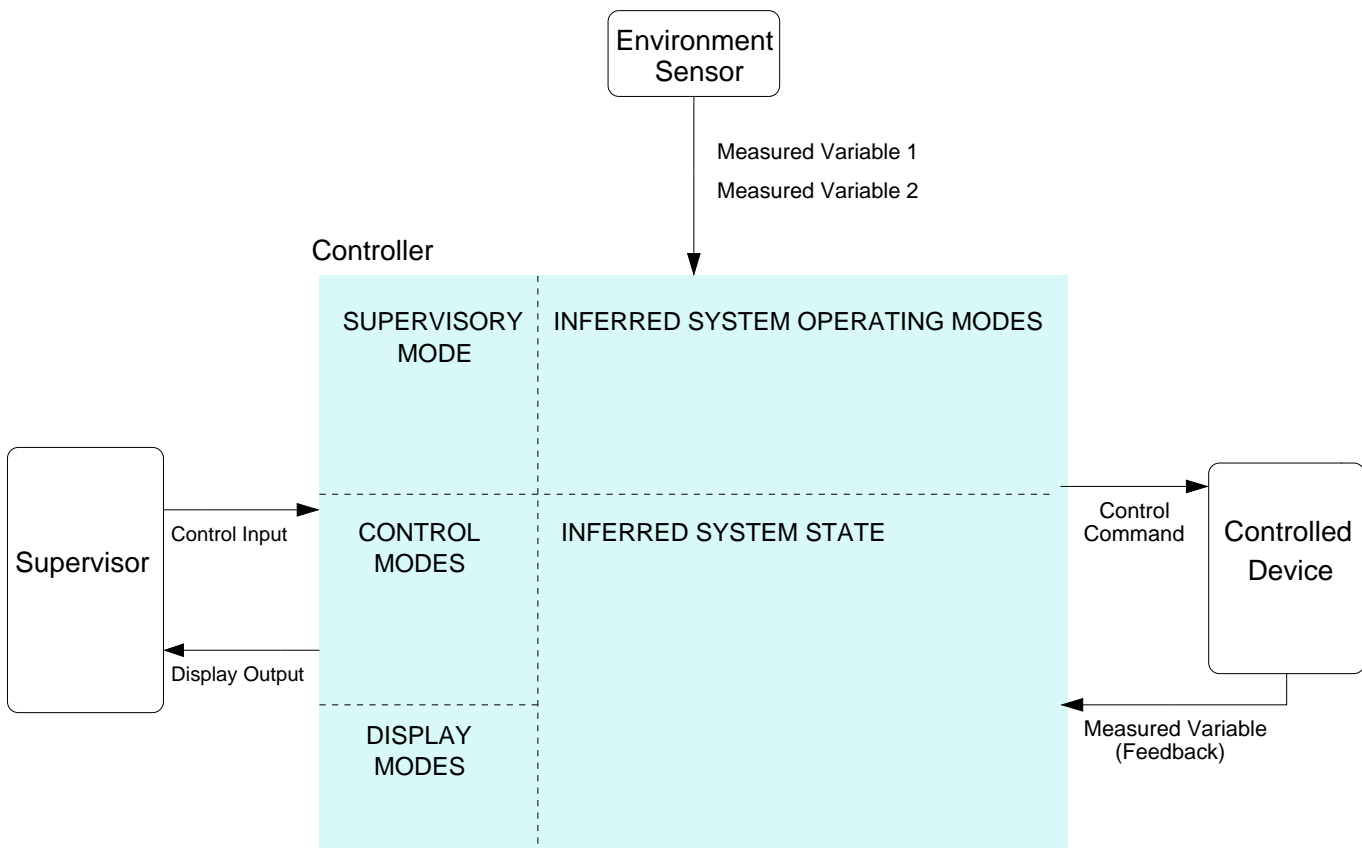
A set of integrated tools to assist in
building complex control systems.

Level 3 Specification (modeling) language goals

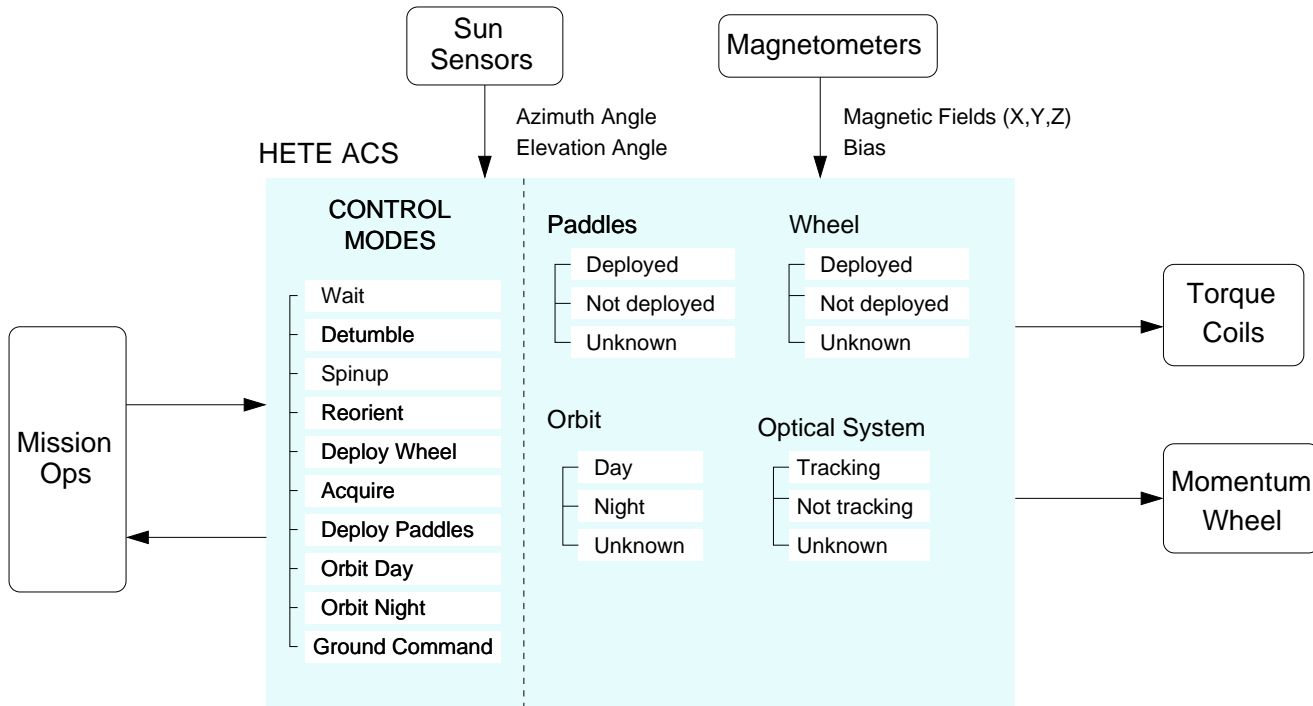
- Readable and reviewable
 - Minimize semantic distance
 - Minimal (blackbox)
 - Easy to learn
 - Unambiguous and simple semantics
- Complete
 - Can specify everything need to specify
- Analyzable
 - Executable
 - Formal (mathematical) foundation
 - Includes human actions
 - Assists in finding incompleteness

SpecTRM-RL

- Combined requirements specification and modeling language
- A state machine with a domain-specific notation on top of it.
 - Includes a task modeling language
 - Can add other notations and visualizations of state machine
- Enforces or includes most of completeness criteria
- Supports specifying systems in terms of modes
 - Control modes
 - Operational modes
 - Supervisory modes
 - Display modes



SpecTRM Model of HETE Attitude Control System



Control Mode

ACS Mode (2)

= Detumble (Mode 1)

The purpose of detumble mode is to minimize the magnitude of body momentum vector in the X-Z plane. As soon as the magnitude falls below a threshold, software should transition to spinup mode. The mode delay provides hysteresis in the mode transitions to prevent the software from jumping between modes too rapidly.

In detumble mode, the wheel actuator shall be controlled such that the wheel maintains the velocity it had upon entering the mode, and the magnetic moment along the Y axis shall be controlled to minimize the angular velocity about the X and Z axes.

		OR				
Control Mode	Wait	T				
	Detumble		T	T		
	Spinup				T	T
	Ground Control					T
State Values	Time since entered wait >= 10 sec	T				
	Time since entered detumble < 100 sec		T	F		
	xz momentum error > xz momentum error threshold			T	T	T
	Time since entered spinup >= 100 sec				T	T
	Paddles in-state deployed				F	
	Optical system in-state tracking					F
	Time since entered ground control >= 10 sec					T

Requirements Analysis

- Model Execution, Animation, and Visualization
- Completeness
- State Machine Hazard Analysis (backwards reachability)
- Human Task Analysis
- Test Coverage Analysis and Test Case Generation
- Automatic code generation

Completeness Analysis

- Automated consistency checker
- Automated completeness (robustness) checker
- Components of modeling language to assist in checking additional criteria
- API to allow checking by other formal analysis tools

Output Command

Name

Destination:

Acceptable Values:

Units:

Granularity:

Exception Handling:

Hazardous Values:

Timing Behavior:

Initiation Delay:

Completion Deadline:

Output Capacity Assumptions:

Load:

Min time between outputs:

Max time between outputs:

Hazardous timing behavior:

Exception-Handling:

Feedback Information:

Variables:

Values:

Relationship:

Min. time (latency):

Max. time:

Exception Handling:

Reversed By:

Comments:

References: ↑ ↓

DEFINITION

= ...

Input Value

DA1 Alt Signal

Source: Digital Altimeter 1

Type: integer

Possible Values (Expected Range): -20..2500

Exception-Handling: Values below -20 are treated as -20 and values above 2500 as 2500

Units: feet AGL

Granularity: 1 foot

Arrival Rate (Load): one per second average

Min-Time-Between-Inputs: 100 milliseconds

Max-Time-Between-Inputs: none

Obsolescence: 2 seconds

Exception-Handling: Assumes value Obsolete

Description:

Comments:

Appears in: Altitude

DEFINITION

= New Data for DA1 Alt Signal

DA1 Alt Signal was Received	T
-----------------------------	---

= Previous Value of DA1 Alt Signal

DA1 Alt Signal was Received	F
Time Since DA1 Alt Signal was last received > 2 seconds	F

= Obsolete

DA1 Alt Signal was Received	F	*	*
Time Since DA1 Alt Signal was last received > 2 seconds	T	*	*
System Start	*	T	*
DA1 Alt Signal was Never Received	*	*	T

Model Execution and Animation

- SpecTRM–RL models are executable.
- Model execution is animated
- Results of execution could be input into a graphical visualization
- Inputs can come from another model or simulator and output can go into another model or simulator.

Executable Specifications as Prototypes

- Easily changed
- At end, have specification to use
- Can be reused (product families)
- Can be more easily reviewed
- If formal, can be analyzed
- Can be used in hardware–in–the–loop or operator–in–the–loop simulations

Operator Task Models

- To ensure safe and efficient operations, must look at the interaction between the human controllers and the computer.
- Use same underlying formal modeling language.
- Designed a visual representation more appropriate for the task modeling.
- Can be executed and analyzed along with other parts of the model.

Interactive Visualizations

- Pictures or diagrams allowing direct manipulation to learn more about the thing represented
- Useful to understand specification or results of analysis or simulation
- Part of a potential "CATIA" for the logical parts of systems
- Goal is to enhance intellectual manageability of complex system design, review, maintenance, and operation.
 To extend human cognitive limits
- Experimental results show extremely useful in reading and reviewing specifications of complex systems.
- We are trying to provide a theoretical foundation for designing interactive visualizations.

Summary

- Integrate design rationale and safety information into specification and its structure
- Capture domain knowledge (reusable architectures)
- Provide traceability from high-level requirements to detailed design and code.
- Blackbox models at Level 3
 - Executable and analyzable
e.g., completeness, robustness, mode confusion, hazard analysis, test case generation, code generation
 - Specification acts as an executable prototype
Can interface with system simulation
 - Visualization tools
 - Interface to contractors

Design for Safety

Unfortunately, everyone had forgotten why the branch came off the top of the main and nobody realized that this was important.

Trevor Kletz
What Went Wrong?

Before a wise man ventures into a pit, he lowers a ladder — so he can climb out.

Rabbi Samuel Ha-Levi Ben Joseph Ibm Nagrela

Design for Safety

- Software design must enforce safety constraints
- Should be able to trace from requirements to code (vice versa)
- Design should incorporate basic safety design principles

Passive vs. Active Protection

- Passive safeguards:
 - Maintain safety by their presence
 - Fail into safe states
- Active safeguards:
 - Require hazard or condition to be detected and corrected

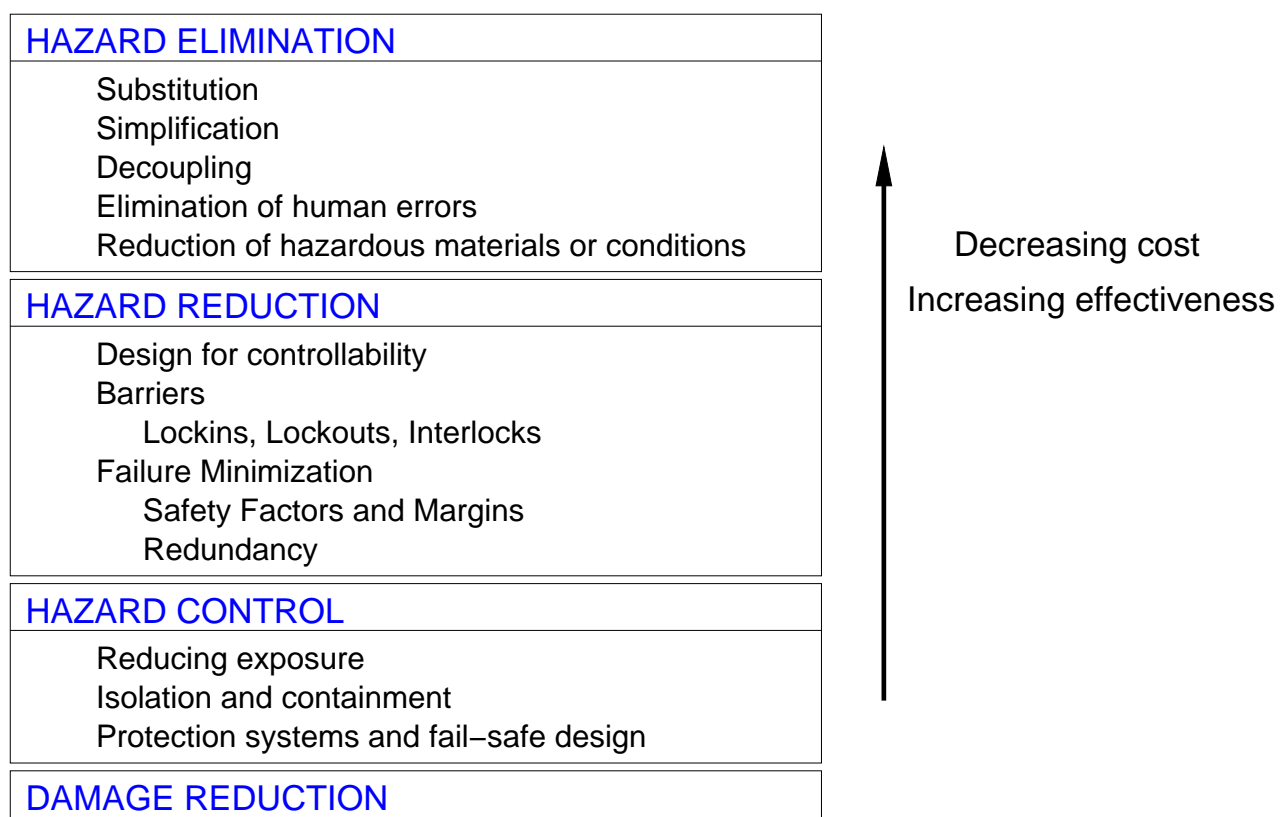
Tradeoffs:

- Passive rely on physical principles
- Active depend on less reliable detection and recovery mechanisms.

BUT

- Passive tend to be more restrictive in terms of design freedom and not always feasible to implement.

Safe Design Precedence



Hazard Elimination

▫ SUBSTITUTION

- Use safe or safer materials.
- Simple hardware devices may be safer than using a computer.
- No technological imperative that says we **MUST** use computers to control dangerous devices.
- Introducing new technology introduces unknowns and even unk-unks.

▫ SIMPLIFICATION

- ? A simple design minimizes:
 - Number of parts
 - Functional Modes
 - Interfaces
- A simple system has a small number of unknowns in the interactions within the system and with its environment.
- A system is intellectually unmanageable when the level of interactions reaches point where they cannot be thoroughly
 - planned
 - understood
 - anticipated
 - guarded against
- System accidents occur when systems become intellectually unmanageable.

□ SIMPLIFICATION

Criteria for a simple software design:

1. Testable: Number of states limited
 - determinism vs. nondeterminism
 - single tasking vs. multitasking
 - polling over interrupts
2. Easily understood and readable
3. Interactions between components are limited and straightforward.
4. Code includes only minimum features and capability required by system.
 - Should not contain unnecessary or undocumented features or unused executable code.
5. Worst case timing is determinable by looking at code.

□ SIMPLIFICATION (con't)

- Reducing and simplifying interfaces will eliminate errors and make designs more testable.
- Easy to add functions to software, hard to practice restraint.
- Constructing a simple design requires discipline, creativity, restraint, and time.
- Design so that structural decomposition matches functional decomposition.

□ DECOUPLING

- Tightly coupled system is one that is highly interdependent:
 - Each part linked to many other parts.
Failure or unplanned behavior in one can rapidly affect status of others.
 - Processes are time-dependent and cannot wait.
Little slack in system
 - Sequences are invariant.
 - Only one way to reach a goal.
- System accidents caused by unplanned interactions.
- Coupling creates increased number of interfaces and potential interactions.

□ DECOUPLING (con't)

- Computers tend to increase system coupling unless very careful.
- Applying principles of decoupling to software design:
 - Modularization: How split up is crucial to determining effects.
 - Firewalls
 - Read-only or restricted write memories
 - Eliminating hazardous effects of common hardware failures
 - Functional cohesion

□ ELIMINATION OF HUMAN ERRORS

- Design so few opportunities for errors.
 - Make impossible or possible to detect immediately.
- Lots of ways to increase safety of human–machine interaction.
 - Making status of component clear.
 - Designing software to be error tolerant
 - etc. (will cover separately)
- Programming language design:
 - Not only simple itself (masterable), but should encourage the production of simple and understandable programs.
 - Some language features have been found to be particularly error prone.

□ REDUCTION OF HAZARDOUS MATERIALS OR CONDITIONS

- Software should contain only code that is absolutely necessary to achieve required functionality.
 - Implications for COTS
 - Extra code may lead to hazards and may make software analysis more difficult.
- Memory not used should be initialized to a pattern that will revert to a safe state.

Turbine–Generator Example

Safety requirements:

1. Must always be able to close steam valves within a few hundred milliseconds.
2. Under no circumstances can steam valves open spuriously, whatever the nature of internal or external fault.

Divided into two parts (decoupled) on separate processors:

1. Non–critical functions: loss cannot endanger turbine nor cause it to shutdown.
 - less important governing functions
 - supervisory, coordination, and management functions
2. Small number of critical functions.

Turbine–Generator Example (2)

- Uses polling : No interrupts except for fatal store fault (nonmaskable)
 - Timing and sequencing thus defined
 - More rigorous and exhaustive testing possible.
- All messages unidirectional
 - No recovery or contention protocols required
 - Higher level of predictability
- Self–checks of
 - Sensibility of incoming signals
 - Whether processor functioning correctly
- Failure of self–check leads to reversion to safe state through fail–safe hardware.
- State table defines:
 - Scheduling of tasks
 - Self–check criteria appropriate under particular conditions

Design for Controllability

Make system easier to control, both for humans and computers.

- Use incremental control:
 - Perform critical steps incrementally rather than in one step.
 - Provide feedback
 - To test validity of assumptions and models upon which decisions made
 - To allow taking corrective action before significant damage done.
 - Provide various types of fallback or intermediate states
- Lower time pressures
- Provide decision aids
- Use monitoring

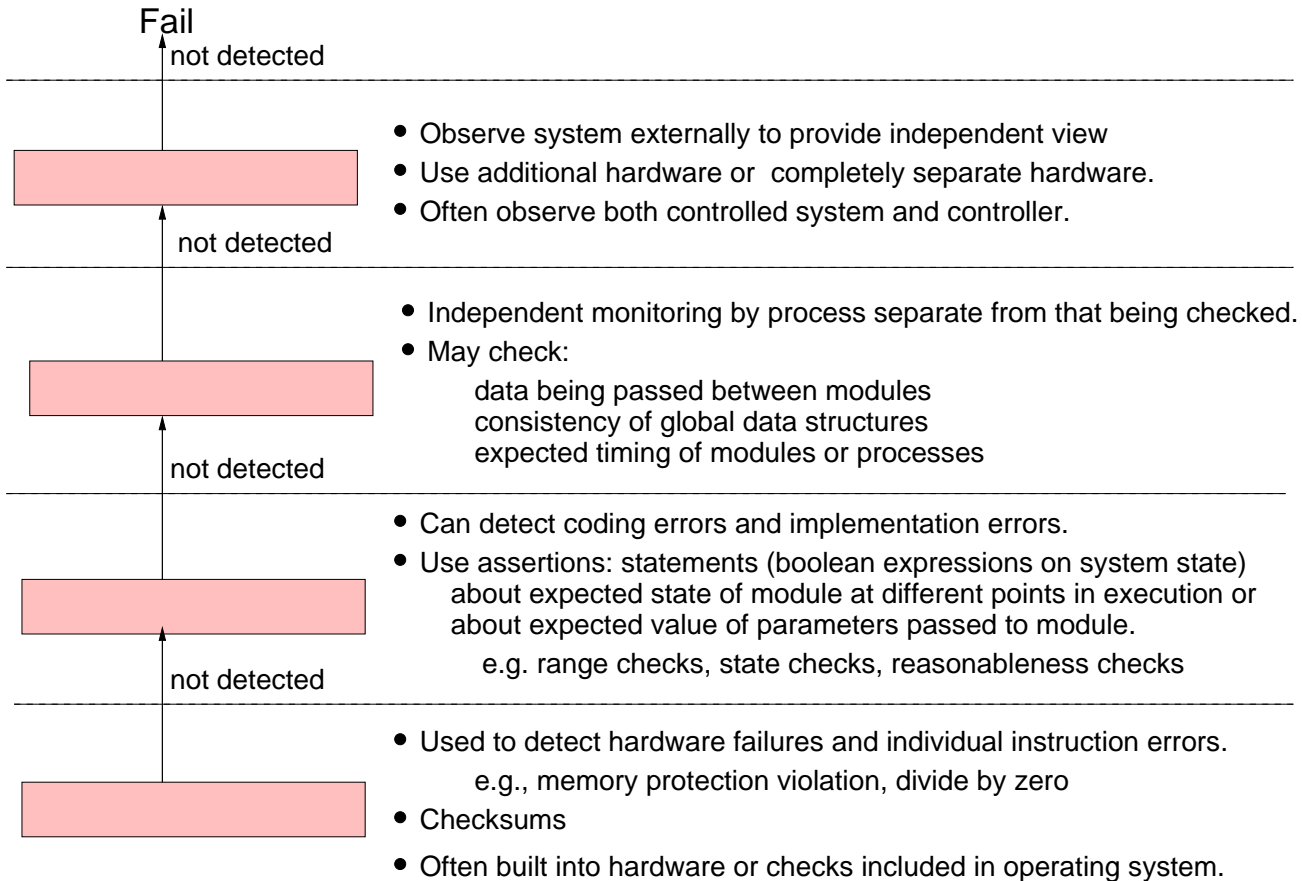
Monitoring

Difficult to make monitors independent:

- Checks require access to information being monitored but usually involves possibility of corrupting that information.
- Depends on assumptions about structure of system and about errors that may or may not occur
 - May be incorrect under certain conditions
 - Common incorrect assumptions may be reflected both in design of monitor and devices being monitored.

A Hierarchy of Software Checking

© Leveson - 253
Design



Software Monitoring (Checking)

© Leveson - 254
Design

- In general, farther down the hierarchy check can be made, the better:
 - Detect the error closer to the time it occurred and before erroneous data used.
 - Easier to isolate and diagnose the problem
 - More likely to be able to fix erroneous state rather than recover to safe state.
- Writing effective self-checks very hard and number usually limited by time and memory.
 - Limit to safety-critical states
 - Use hazard analysis to determine check contents and location
- Added monitoring and checks can cause failures themselves.

Barriers

- LOCKOUTS
 - Make access to dangerous state difficult or impossible.
 - Implications for software:
 - Avoiding EMI
 - Authority limiting
 - Controlling access to and modification of critical variables
- Can adapt some security techniques

- LOCKIN
 - Make it difficult or impossible to leave a safe state.
 - Need to protect software against environmental conditions.
 - e.g., operator errors
 - data arriving in wrong order or at unexpected speed
 - Completeness criteria ensure specified behavior robust against mistaken environmental conditions.

▫ INTERLOCK

- Used to enforce a sequence of actions or events.
 1. Event A does not occur inadvertently
 2. Event A does not occur while condition C exists
 3. Event A occurs before event D.
- Examples:
 - Batons
 - Critical sections
 - Synchronization mechanisms

Remember, the more complex the design, the more likely errors will be introduced by the protection facilities themselves.

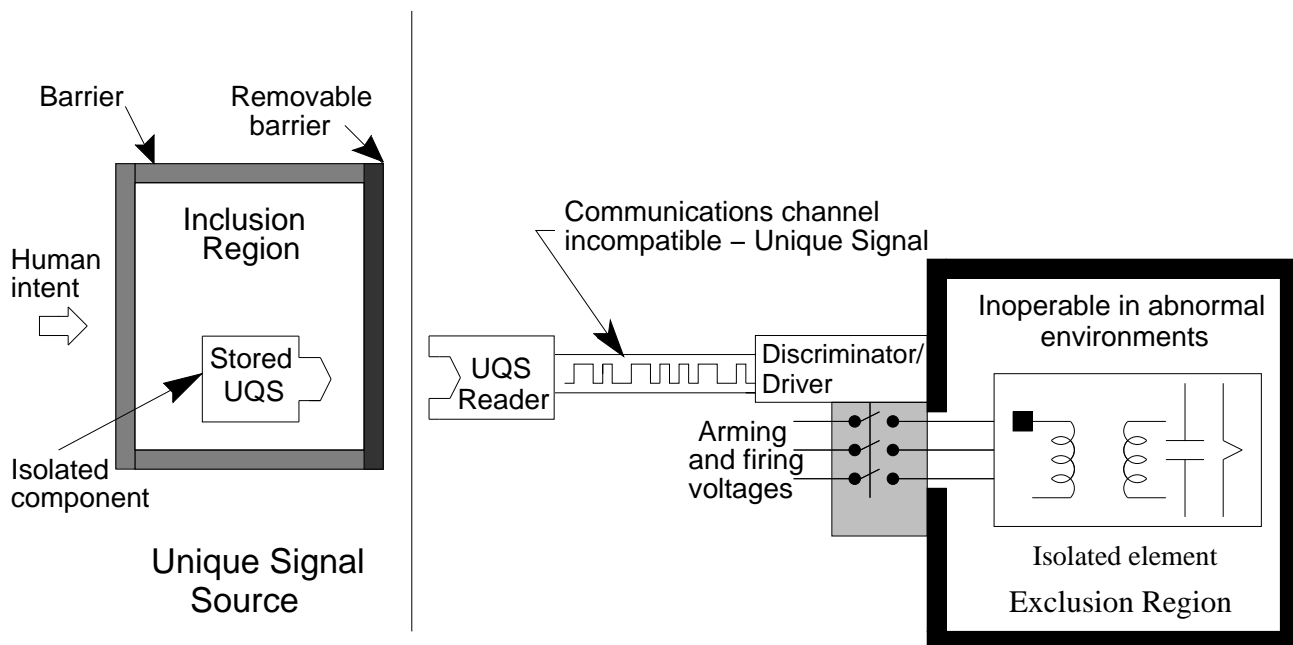
Example: Nuclear Detonation

- Safety depends on NOT working
- Three basic techniques (called “positive measures”)
 1. Isolation
 - Separate critical elements (barriers)
 2. Inoperability
 - Keep in inoperable state, e.g., remove ignition device or arming pin
 3. Incompatibility
 - Detonation requires an unambiguous indication of human intent be communicated to weapon.
 - Protecting entire communication system against all credible abnormal environments (including sabotage) not practical.
 - Instead, use unique signal of sufficient information complexity that unlikely to be generated by an abnormal environment.

Example: Nuclear Detonation (2)

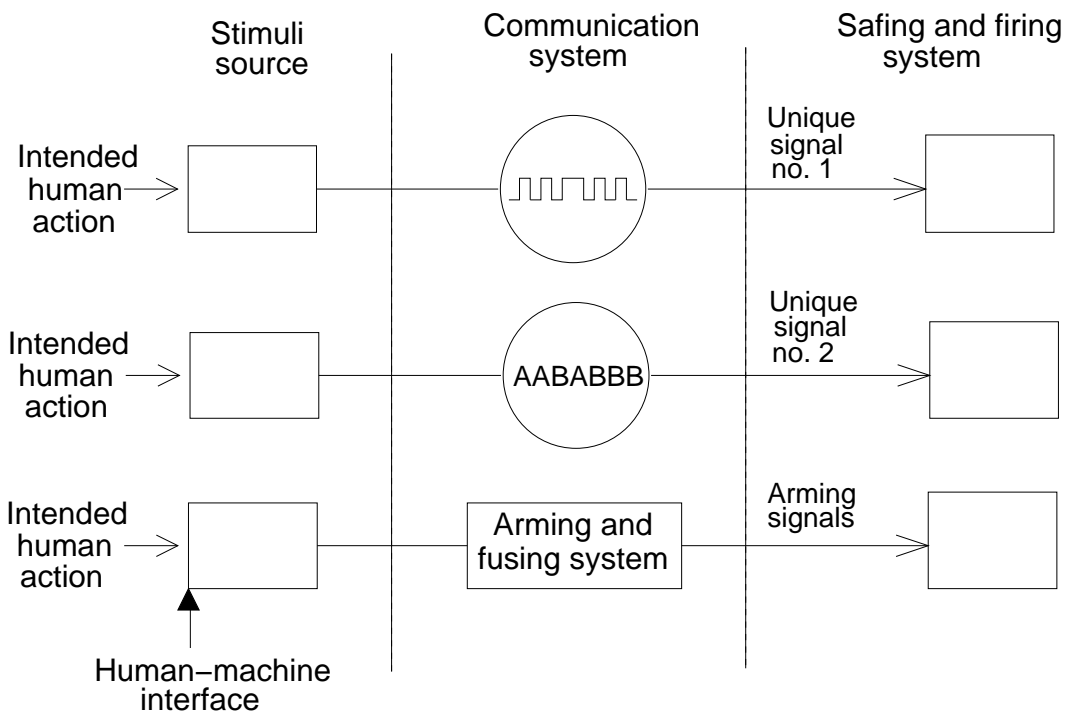
- Unique signal discriminators must:
 1. Accept proper unique signal while rejecting spurious inputs
 2. Have rejection logic that is highly immune to abnormal environments
 3. Provide predictably safe response to abnormal environments
 4. Be analyzable and testable
- Protect unique signal sources by barriers.
- Removable barrier between these sources and communication channels.

Example: Nuclear Detonation (3)



Example: Nuclear Detonation (4)

May require multiple unique signals from different individuals along various communication channels, using different types of signals (energy and information) to ensure proper intent.



Failure Minimization

▫ SAFETY FACTORS AND SAFETY MARGINS

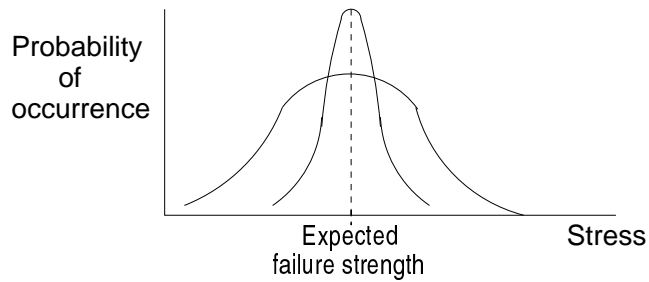
Used to cope with uncertainties in engineering:

- Inaccurate calculations or models
- Limitations in knowledge
- Variation in strength of a specific material due to differences in composition, manufacturing, assembly, handling, environment, or usage.

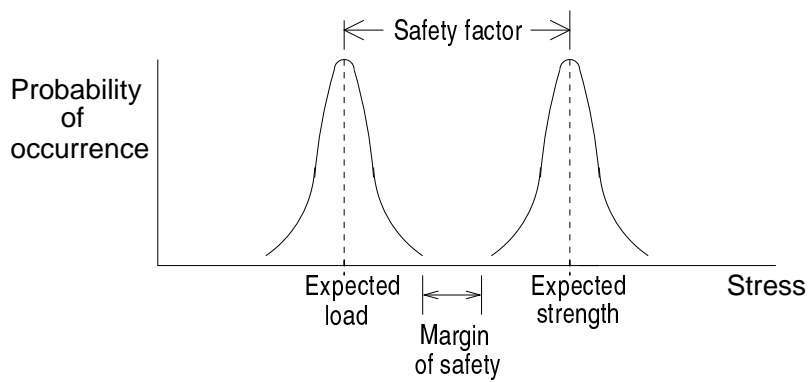
Some ways to minimize problem, but cannot eliminate it.

Appropriate for continuous and non-action systems.

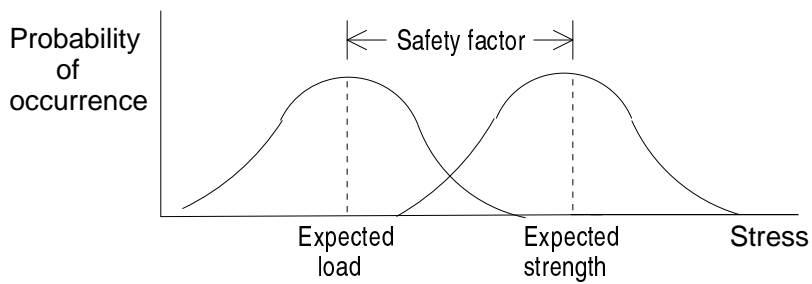
Safety Margins and Safety Factors



(a) Probability density function of failure for two parts with same expected failure strength.



(b) A relatively safe case.



(c) A dangerous overlap but the safety factor is the same as in (b)

▫ REDUNDANCY

Goal is to increase reliability and reduce failures.

- Common-cause and common-mode failures
- May add so much complexity that causes failures.
- More likely to operate spuriously.
- May lead to false confidence (Challenger)

Useful to reduce hardware failures. But what about software?

- Design redundancy vs. design diversity
- Bottom Line: claims that multiple version software will achieve ultra-high reliability levels are not supported by empirical data or theoretical models.

▫ REDUNDANCY (con't.)

- Standby spares vs. concurrent use of multiple devices (with voting)
- Identical designs or intentionally different ones (diversity).
- Diversity must be carefully planned to reduce dependencies.
 - Can also introduce dependencies in maintenance, testing, repair
- Redundancy most effective against random failures not design errors.

▫ REDUNDANCY (con't.)

- Software errors are design errors.

Data redundancy: extra data for detecting errors

e.g. parity bit and other codes

checksums

message sequence numbers

duplicate pointers and other structural information

Algorithmic redundancy:

1. Acceptance tests (hard to write)
2. Multiple versions with voting on results

Multi (or N) Version Programming

- Assumptions:
 - Probability of correlated failures is very low for independently developed software.
 - Software errors occur at random and are unrelated.
- Even small probabilities of correlated failures cause a substantial reduction in expected reliability gains.
- Conducted a series of experiments with John Knight
 - Failure independence in N-version programming
 - Embedded assertions vs. N-version programming
 - Fault Tolerance vs. Fault Elimination

Failure Independence

- Experimental Design:
 - 27 programs, one requirements specification
 - Graduate students and seniors from two universities
 - Simulation of a production environment: 1,000,000 input cases
 - Individual programs were high quality
- Results:
 - Rejected independence hypothesis: Analysis of reliability gains must include effect of dependent errors.
 - Statistically correlated failures result from:
 - Nature of application
 - "Hard" cases in input space
 - Programs with correlated failures were structurally and algorithmically very different.
 - Using different programming languages and compilers won't help

Conclusion: Correlations due to fact that working on same problem, not due to tools used or languages used or even algorithms used.

Consistent Comparison Problem

- Arises from use of finite-precision real numbers (rounding errors)
- Correct versions may arrive a completely different correct outputs and thus be unable to reach a consensus even when none of components "fail".
- May cause failures that would not have occurred with single versions.
- No general practical solution to the problem .

Self-Checking Software

Experimental Design:

- Launch Interceptor Programs (LIP) from previous study.
- 24 graduate students from UCI and UVA employed to instrument 8 programs (chosen randomly from subset of 27 in which we had found errors).
- Provided with identical training materials.
- Checks written using specifications only at first and then participants were given a program to instrument.
- Allowed to make any number or type of check.
- Students treated this as a competition among themselves.

Fault Tolerance vs. Fault Elimination

Techniques compared:

- Run-time assertions (self-checks)
- Multi-version voting
- Functional testing augmented with structural testing
- Code reading by stepwise abstraction
- Static data-flow analysis

Experimental Design:

- Combat Simulation Problem (from TRW)
- Programmers separate from fault detectors
- Eight version produced with 2 person teams
 - Number of modules from 28 to 75
 - Executable lines of code from 1200 to 2400
- Attempted to hold resources constant for each technique.

Self-Checking Software (2)

	Already Known Errors			Other Errors			Added Errors	
	#	SP	CR	CD	SP	CR		CD
3a 3b 3c	4		1					
6a 6b 6c	3		2			1	1 1	
8a 8b 8c	2			2			1 3	
12a 12b 12c	2	1				1	2 2	
14a 14b 14c	2		1			1	2 2	
14a 14b 14c	2						4	
20a 20b 20c	2		1		1		1 2	
20a 20b 20c	2		1		1		2 2	
23a 23b 23c	2	2					4	
23a 23b 23c	2						4	
25a 25b 25c	3		2			1	1 1	
25a 25b 25c	3			1			1 1	
Total	60	3	8	3	1	0	5	22

Spec Read Chks Spec Read Chks
KNOWN NEWLY FOUND ADDED

Fault Tolerance vs. Fault Elimination (2)

Results:

- Multi-version programming is not a substitute for testing.
 - Did not tolerate most of faults detected by fault-elimination techniques.
 - Unreliable in tolerating the faults it was capable of tolerating.
- Testing failed to detect errors causing coincident failures.
- Cast doubt on effectiveness of voting as a test oracle.
 - Instrumenting the code to examine internal states was much more effective.
- Intersection of sets of faults found by each method was relatively small.

N-Version Programming (Summary)

Doesn't mean shouldn't use, but should have realistic expectations of benefits to be gained and costs involved:

- Costs very high (more than N times)
- In practice, end up with lots of similarity in designs (more than in our experiments)
 - Overspecification
 - Cross Checks

So safety of system dependent on quality that has been systematically eliminated.

And no way to tell how different 2 software designs are in their failure behavior.

- Requirements flaws not handled, which is where most safety problems arise anyway.

Recovery

- Backward
 - Assume can detect error before does any damage.
 - Assume alternative will be more effective.
- Forward
 - Robust data structures.
 - Dynamically altering flow of control.
 - Ignoring single cycle errors.
- But real problem is detecting erroneous states.

Hazard Control

- LIMITING EXPOSURE
 - Start out in safe state and require deliberate change to unsafe state.
 - Set critical flags and conditions as close to code they protect as possible.
 - Critical conditions should not be complementary, e.g., absence of an arm condition should not be used to indicate system is unarmed.
- ISOLATION AND CONTAINMENT
- PROTECTION SYSTEMS AND FAIL-SAFE DESIGN

Protection Systems and Fail-Safe Design

- Depends upon existence of a safe state and availability of adequate warning time.
- May have multiple safe states, depending upon process conditions.
- General rule is hazardous states should be hard to get into and safe states should be easy.
- Panic button
- Watchdog timer: Software it is protecting should not be responsible for setting it.
- Sanity checks (I'm alive signals)
- Protection system should provide information about its control actions and status to operators or bystanders.
- The easier and faster is return of system to operational state, the less likely protection system is to be purposely bypassed or turned off.

Damage Reduction

- May need to determine a “point of no return” where recovery no longer possible or likely and should just try to minimize damage.

Design Modification and Maintenance

- Need to reanalyze for every proposed/implemented change
- Recording design rationale from beginning and traceability will help.

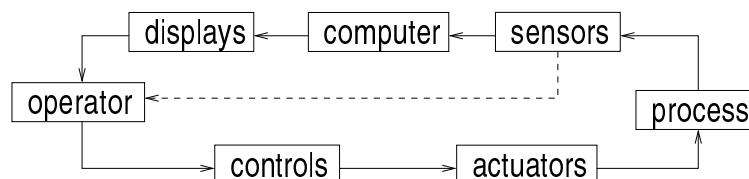
Human-Computer Interaction

[The designers] had no intention of ignoring the human factor ... But the technological questions became so overwhelming that they commanded the most attention.

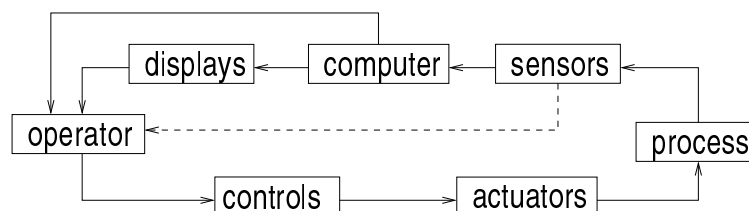
John Fuller
Death by Robot

Possible Roles for Computers in Control Loops

- Computer reads and interprets sensor data for operator

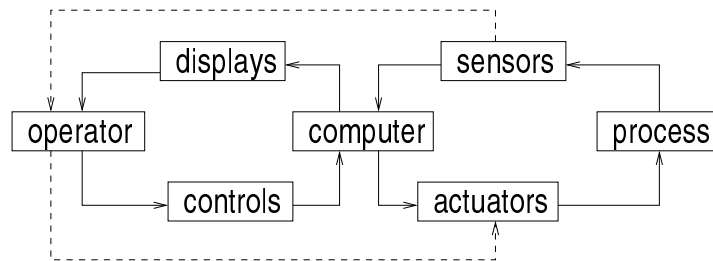


- Computer provides information and advice to operator



More Roles for Computers in Control Loops

- Computer interprets and displays data for operator and issues commands; operator makes varying levels of decisions.



-
- Computer assumes complete control with operator providing advice or high-level supervision or simply monitoring.

Role of Humans in Automated Systems

- The Human as Monitor
 - Task may be impossible
 - Dependent on information provided
 - State of information more indirect
 - Failures may be silent or masked
 - Little active behavior can lead to lower alertness and vigilance, complacency, and overreliance.

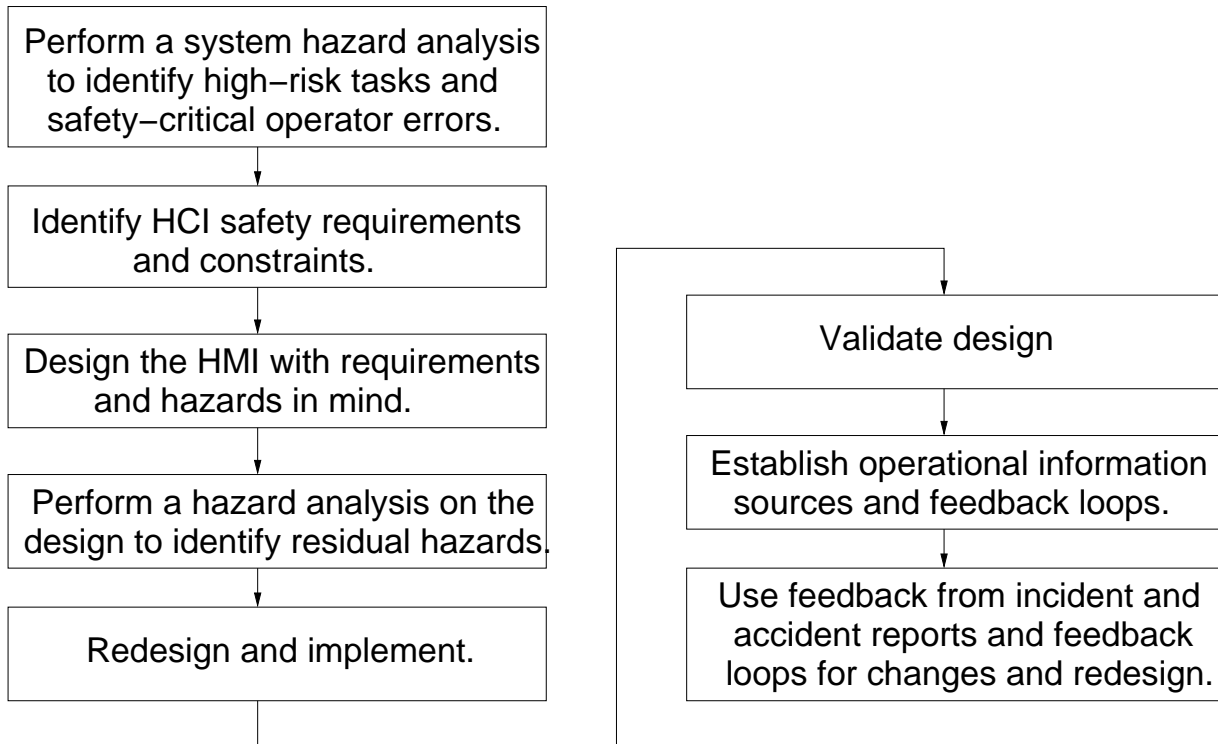
Role of Humans in Automated Systems (con't.)

- The Human as Backup
 - May lead to lowered proficiency and increased reluctance to intervene
 - Fault intolerance may lead to even larger errors
 - May make crisis handling more difficult
- The Human as Partner
 - May be left with miscellaneous tasks
 - Tasks may be more complex and new tasks added
 - By taking away easy parts, may make difficult parts harder

HMI Design

- Simple solution is to automate as much as possible, but is this the best solution?
- Different is not necessarily better.
- Need to consider conflicts between HMI design qualities.
- Norman: Appropriate design should:
 - Assume the existence of error.
 - Continually provide feedback.
 - Continually interact with operators in an effective manner.
 - Allow for the worst situation possible.

HMI Design Process



Matching Tasks to Human Characteristics

- Tailor systems to human requirements instead of vice versa.
- Design to withstand normal, expected human behavior.
- Design to combat lack of alertness.
- Design for error tolerance:
 - Help operators monitor themselves and recover from errors.
 - Provide feedback about actions operators took and their effects.
 - Allow for recovery from erroneous actions.

Approach	
Seat Belt	ON
Cont Ignition	ON
Anti-skid	ON
Altimeters	4 ON
Fuel boost pumps	
Air speed bugs	
Pressurization	
Approach checklist	COMPLETE

Allocating Tasks

- Design considerations.
- Failure detection.
- Making allocation decisions.
- Emergency shutdown.

Human Error vs. Computer Error

- Automation does not eliminate human error or remove humans from systems.
- It simply moves them to other functions
 - Design and programming
 - High-level supervisory control and decision making
 - Maintenance

where increased system complexity and reliance on indirect information makes decision-making process more difficult.

Designers Make Mistakes Too

- Many of same limitations of human operators are characteristic of designers:
 - Difficulty in assessing probabilities of rare events.
 - Bias against considering side effects.
 - Tendency to overlook contingencies.
 - Limited capacity to comprehend complex relationships.
 - Propensity to control complexity by concentrating only on a few aspects of the system.

Mixing Humans and Computers

- Automated systems on aircraft have eliminated some types of human error and created some new ones.
- Human skill levels and required knowledge may go up.
- Correct partnership and allocation of tasks is difficult

Who has the final authority?

Why Not Simply Replace Humans with Computers?

Computers do not produce new sorts of errors. They merely provide new and easier opportunities for making the old errors.

Trevor Kletz, "Wise After the Event"

- Not all conditions (or the correct way to deal with them) are foreseeable.
- Even those that can be predicted are programmed by error-prone human beings.

Advantages of Humans

- Human operators are adaptable and flexible.
 - Able to adapt both goals and means to achieve them.
 - Able to use problem solving and creativity to cope with unusual and unforeseen situations.
 - Can exercise judgement.
- Humans are unsurpassed in
 - Recognizing patterns.
 - Making associative leaps.
 - Operating in ill-structured, ambiguous situations
- Human error is the inevitable side effect of this flexibility and adaptability.

Reducing Human Errors

- Make safety enhancing actions easy, natural, and difficult to omit or do wrong.

Stopping an unsafe action or leaving an unsafe state should require one keystroke.

- Make dangerous actions difficult or impossible.

Potentially dangerous commands should require two or more unique actions.

- Provide references for making decisions.

Reducing Human Errors (2)

- Follow human stereotypes.
- Make sequences dissimilar if need to avoid confusion between them.
- Make errors physically impossible or obvious.
- Use physical interlocks (but be careful about this).

Providing Information and Feedback

- Analyze task to determine what information is needed.
- Provide feedback:
 - About effect of operator's actions
To detect human errors
 - About state of system
To update mental models
To detect system faults
- Provide for failure of computer displays (by alternate sources of information).

Instrumentation to deal with malfunction must not be disabled by the malfunction.

Providing Information and Feedback (2)

- Inform operators of anomalies, actions taken, and current system state.
- Fail obviously or make graceful degradation obvious to operator.
- Making displays easily interpretable is not always best.

Feedforward assistance:

Predictor displays

Procedural checklists and guides (be careful)

Alarms

- Issues:
 - Overload
 - Incredulity Response
 - Relying on as primary rather than backup (management by exception)
- Guidelines:
 - Keep spurious alarms to a minimum.
 - Provide checks to distinguish correct from faulty instruments.
 - Provide checks on alarm system itself.
 - Distinguish between routine and critical alarms.
 - Indicate which condition is responsible for alarm
 - Provide temporal information about events and state changes.
 - Require corrective action when necessary.

Training and Maintaining Skills

- May need to be more extensive and deep.
 - Required skill levels go up (not down) with automation.
- Teach how the software works.
- Teach about safety features and design rationale.
- Teach for general strategies rather than specific responses.

Mode Confusion

- General term for a class of situation–awareness errors
- High tech automation changing cognitive demands on operators
 - Supervising rather than directly controlling
 - More cognitively complex decision making
 - Complicated, mode–rich systems
 - Increased need for cooperation and communication
- Human–factors experts complaining about technology–centered automation
 - Designers focus on technical issues, not on supporting operator tasks
 - Leads to "clumsy" automation
- Errors are changing, e.g. errors of omission vs. commission

Mode Confusion (2)

- Early automated systems had fairly small number of modes.
 - Provided passive background on which operator would act by entering target data and requesting system operations.
- Also had only one overall mode setting for each function performed.
 - Indications of currently active mode and of transitions between modes could be dedicated to one location on display.
- Consequences of breakdown in mode awareness fairly small.
 - Operators seemed able to detect and recover from erroneous actions relatively quickly.

Mode Confusion (3)

- Flexibility of advanced automation allows designers to develop more complicated, mode-rich systems.
- Result was numerous mode indications spread over multiple displays each containing just that portion of mode status data corresponding to a particular system or subsystem.
- Designs also allow for interactions across modes.
- Increased capabilities of automation create increased delays between user input and feedback about system behavior.
- These changes have led to:
 - Increased difficulty of error or failure detection and recovery
 - Challenges to human's ability to maintain awareness of
 - active modes
 - armed modes
 - interactions between environmental status and mode behavior
 - interactions across modes

Mode Confusion Analysis

- Identify “predictable error forms”
 - accidents and incidents
 - simulator studies
- Model blackbox software behavior
- Identify modeled software behavior likely to lead to operator error.
- Reduce probability of error occurring:
 - Redesign the automation
 - Design appropriate HCI
 - Change operational procedures and training

Design Flaws

1. Interface interpretation errors

- Software interprets input wrong
- Multiple conditions mapped to same output

Mulhouse (A320):

Crew directed automated system to fly in TRACK/FLIGHT PATH mode, which is a combined mode related both to lateral (TRACK) and vertical (flight path angle) navigation. When they were given radar vectors by the air traffic controller, they may have switched from the TRACK to the HDG SEL mode to be able to enter the heading requested by the controller. However, pushing the button to change the lateral mode also automatically changes the vertical mode from FLIGHT PATH ANGLE to VERTICAL SPEED, i.e., the mode switch button affects both lateral and vertical navigation. When the pilots subsequently entered "33" to select the desired flight path angle of 3.3 degrees, the automation interpreted their input as a desired vertical speed of 3300 ft. Pilots were not aware of active "interface mode" and failed to detect the problem. As a consequence of too steep a descent, the airplane crashed into a mountain.

— Operating room medical device:

The device has two operating modes: warmup and normal. It starts in warmup mode whenever either of two particular settings are adjusted by the operator (anesthesiologist). The meaning of alarm messages and the effect of controls are different in these two modes, but neither the current device operating mode nor a change in mode are indicated to the operator. In addition, four distinct alarm-triggering conditions are mapped onto two alarm messages so that the same message has different meanings depending on the operating mode. In order to understand what internal condition triggered the message, the operator must infer which malfunction is being indicated by the alarm.

— Display modes: In some devices, user-entered target values interpreted differently depending on active display mode.

2. Inconsistent behavior

- Harder for operator to learn how automation works
 - Important because pilots changing scanning behavior
- In go-around below 100 feet, pilots failed to anticipate and realize autothrust system did not arm when they selected TOGA power because it did so under all other circumstances where TOGA power is applied (found in simulator study of A320).
- Cali
- Bangalore (A320): a protection function is provided in all automation configurations except the ALTITUDE ACQUISITION mode in which autopilot was operating.

Design Flaws

3. Indirect mode changes

- Automation changes mode without direct command
- Activating one mode can activate different modes depending on system status at time of manipulation.

Bangalore (A320):

Pilot put plane into OPEN DESCENT mode without realizing it. Resulted in aircraft speed being controlled by pitch rather than thrust, i.e., throttles went to idle. In that mode, automation ignores any preprogrammed altitude constraints. To maintain pilot-selected speed without power, automation had to use an excessive rate of descent, which led to crash short of runway.

How could this happen?

Three different ways to activate OPEN DESCENT mode:

- 1) Pull altitude knob after select lower altitude.
- 2) Pull speed knob when aircraft in EXPEDITE mode.
- 3) Select a lower altitude while in ALTITUDE ACQUISITION mode.

Pilot must not have been aware that aircraft was within 200 feet of previously entered target altitude (which puts into ALTITUDE ACQUISITION mode). Thus may not have expected selection of lower altitude at that time to result in mode transition. So may not have closely monitored his mode annunciations. Discovered what happened at 10 secs before impact -- too late to recover with engines at idle.

Design Flaws

4. Operator authority limits

- Prevents actions that would lead to hazardous state
- May prohibit maneuvers needed in extreme situations

– Warsaw

– During one A320 approach, pilots disconnected the autopilot while leaving the flight director engaged. Under these conditions, the automation provides automatic speed protection by preventing aircraft from exceeding upper and lower airspeed limits. At some point during approach, after flaps 20 had been selected, the aircraft exceeded airspeed limit for that configuration by 2 kts. As a result, the automation intervened by pitching the airplane up to reduce airspeed back to 195 kts. The pilots, who were unaware that automatic speed protection was active, observed the uncommanded automation behavior. Concerned about the unexpected reduction in airspeed at this critical phase of flight, they rapidly increased thrust to counterbalance the automation. As a consequence of this sudden burst of power, the airplane pitched up to about 50 degrees, entered a sharp left bank, and went into a dive. The pilots eventually disengaged the autothrust system and its associated protection function and regained control of the aircraft.

Design Flaws

5. Unintended side effects

- An action intended to have one effect has an additional one

Because approach is such a busy time and the automation requires so much heads down work, pilots often program the automation as soon as they are assigned a runway.

In an A320 simulator study, discovered that pilots were not aware that entering a runway change AFTER entering the data for the assigned approach results in the deletion of all previously entered altitude and speed constraints even though they may still apply.

Design Flaws

6. Lack of appropriate feedback

- Operator needs feedback to predict or anticipate mode changes
- Independent information needed to detect computer errors

Bangalore (A320): PF had disengaged his flight director during approach and was assuming PNF would do the same. Result would have been a mode configuration in which airspeed is automatically controlled by the autothrottle (the SPEED mode), which is the recommended procedure for the approach phase. However, the PNF never turned off his flight director, and the OPEN DESCENT mode became active when a lower altitude was selected. This indirect mode change led to the hazardous state and eventually the accident. But a complicating factor was that each pilot only received an indication of the status of his own flight director and not all the information necessary to determine whether the desired mode would be engaged. The lack of feedback or knowledge of the complete system state contributed to the pilots not detecting the unsafe state in time to reverse it.

Example: Oops, It Didn't Arm (Everett Palmer, NASA Ames)

"Automation surprises" occur when the automation behaves in a manner that is different from what the operator is expecting. In the following case, a reasonable sequence of pilot actions performed in a high workload situation resulted in an unusual and undesirable result -- an automation surprise. In this case, the automation worked as designed.

Altitude deviations are the most common incident reported to the Aviation Safety Reporting System -- they are reported at the rate of about one per hour. In the following case, an automatic mode transition leads to an altitude deviation. This is such a common problem that it has been given a name, a "kill-the-capture" bust. Hundreds of similar altitude deviations have been reported to the ASRS. The only unique thing about this incident is that it occurred during a full-mission simulator study and was recorded for later analysis.

The incident took less than 20 seconds to play out. The crew had just made a missed approach and had climbed to and leveled at 2,100 feet. They received the clearance to "climb now and maintain 5000 feet". The Flight Mode Annunciator (FMA) showed:

	THRUST	ARM	ROLL	PITCH
A. Level at 2100 ft.	SPD 186		VOR CAP	ALT HLD

They received the clearance to : "... climb now and maintain 5000 feet ..." After some communication confusion on the cleared altitude (5,000 or 15,000) and which radial to hold on (0-0-6 or 0-6-0), the Captain set the MCP altitude window to 5,000 feet,

	THRUST	ARM	ROLL	PITCH
B. Enter 5000 in MCP	SPD 186	ALT	VOR CAP	ALT HLD

set the autopilot pitch mode to vertical speed with a value of approximately 2,000 feet per minute,

	THRUST	ARM	ROLL	PITCH
C Set VERT/SPD.	SPD 186	ALT	VOR CAP	VERT SPD

and set the autothrottle to SPD mode with a value of 255 knots.

	THRUST	ARM	ROLL	PITCH
D. Enter 255 in MCP speed window	SPD 255	ALT	VOR CAP	VERT SPD

Climbing through 3,500 feet, the Captain called for flaps up and at 4,000 feet he called for slats retract. As the aircraft climbed from 4,000 to 5,000 feet, the first officer was copying the holding clearance. Climbing through 4,000 feet, the FMA showed:

	THRUST	ARM	ROLL	PITCH
E. Approaching 4000 feet.	SPD 255	ALT	VOR TRK	VERT SPD

Passing through 4000 feet, the Captain pushed the IAS button on the MCP. The pitch mode became IAS and the autothrottles went to CLAMP mode.

	THRUST	ARM	ROLL	PITCH
F. Push IAS	CLMP	ALT	VOR TRK	IAS

Altitude capture was still armed. Three seconds later, the autopilot automatically switched to altitude capture mode. The FMA arm window went blank and the pitch window showed ALT/CAP.

	THRUST	ARM	ROLL	PITCH
G. Automatic altitude capture	SPD 255		VOR TRK	ALT CAP

A tenth of a second later, the Captain adjusted the vertical speed wheel to a value of about 4000 feet a minute. This caused the pitch autopilot mode to switch from altitude capture to vertical speed.

	THRUST	ARM	ROLL	PITCH
H Adjust vertical speed	SPD 255		VOR TRK	VERT SPD

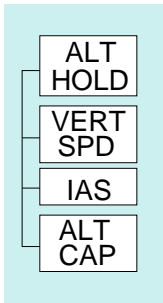
Climbing through 4500 feet, the approaching altitude light was on. As the altitude passed through 5,000 feet at a vertical velocity of about 4,000 feet per minute, the Captain remarked "Five thousand. Oops, it didn't arm." He pushed the MCP ALT/HLD button and switched off the autopilot. The aircraft continued to climb to about 5,500 feet and the "ALTITUDE – ALTITUDE" voice warning sounded repeatedly.

Exercise: What was the problem in the automation design that led to this incident?

Comments: As in many incidents involving automation, the error was first detected by the pilots not by using the autoflight displays such as the Flight Mode Annunciators that tell the state of the automation, but by the basic aircraft displays such as the altimeter and the vertical speed indicator. The crew was apparently aware of the state of the aircraft but not aware of the state of the automation. The error was detected by observing the unexpected state of the basic aircraft displays, not the automation display. Woods has observed that most errors that result from the use of automation are detected by observing the system response and not the automation mode display.

In this incident, the automation display (the FMA) indicated what was actually happening; however the immediate response of the aircraft and the primary aircraft instruments were normal. The unusual and unexpected aircraft behavior occurred later. Although this is an error tolerant system, error detection was delayed beyond the point where that was possible. Why might this be the case? What makes it difficult to use the information in the FMA to verify the correct autoflight mode? A number of possible reasons. First, the FMA must be read and its meaning interpreted. Sometimes what must be "read" and interpreted is the absence of information. Second, the FMA's physical location away from the MCP requires that the pilot act in one place and check the outcome of the action in another place. Finally, the FMA does not provide a direct display of what the pilot needs to know to stay ahead of the aircraft, i.e., What trajectory have I set up the automation to fly the aircraft on?

PITCH MODE



→ VRT SPD

		OR	
AND	AP in-mode On	T	T
	Pilot adjusts V/SPD wheel	T	□
	Pilot pushes V/SPD button	□	T

RESULT:
Change Pitch annunciator to VRT SPD

→ ALT HOLD

		OR	
AND	AP in-mode On	T	T
	Pilot pushes HOLD	T	□
	Alt acquired	□	T
	In-mode ALT CAP	□	T

RESULT:
Change Pitch annunciator to ALT HOLD

→ IAS

		T
AND	AP in-mode On	T
	Pilot pushes IAS	T

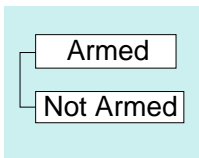
RESULT:
Change Pitch annunciator to IAS
Autothrottle goes to CLAMP mode

→ ALT CAP

		T
AND	AP in-mode On	T
	Condition to start leveling off	T
	Capture in-mode Armed	T

RESULT:
Start leveling off
Change Pitch annunciator to ALT CAP

CAPTURE MODE



→ Not Armed

		OR	
AND	Capture in-mode Armed	T	T
	Pilot pushes ALT	T	□
	Pitch in-mode ALT CAP	□	T

RESULT:
Change Arm annunciator to blank

→ Armed

		OR	
AND	Capture in-mode Not Armed	T	T
	Pilot sets new higher alt	T	□
	Pilot pulls ALT	□	T

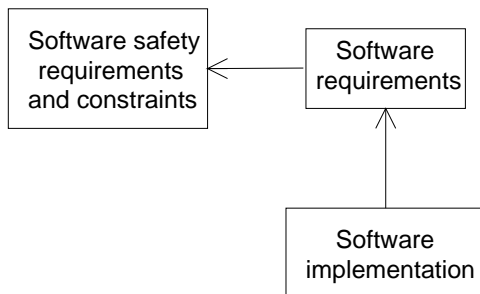
RESULT:
Change Arm annunciator to ALT

Verification of Safety

Unlike the fairy tale Rumpelstiltskin, do not think that by having named the devil that you have destroyed him. Positive verification of his demise is required.

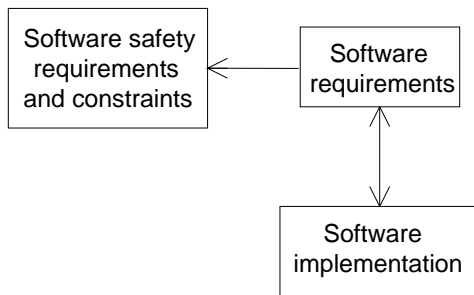
System Safety Handbook for the
Acquisition Manager, U.S. Air Force

Approaches to Verifying Safety

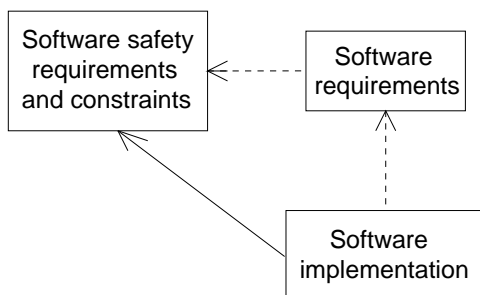


Showing the implementation satisfies the specification and the specification satisfies the safety requirements and constraints is not adequate.

Software can do more than is in the requirements.



This solves the unintended function problem but is impractical.



Verification of the implementation directly against the safety requirements and constraints.

Testing for Safety

Testing for safety starts from system safety design constraints.

Goal is to show that software will not do anything that will lead to violating the constraints.

- Test for hazardous outputs.
- Test effectiveness of any specific safety design features.
(Write them first)
- Focus on what software will do that it is not supposed to do.
- Like any testing, specify early in development cycle and evolve as design and hazard analysis evolves.
- Test as you fly

Testing for Safety (2)

Need to test:

- Critical functions for hazardous behavior.
- Boundary conditions.
- Special features (e.g., firewalls) upon which protection of critical functions based.
- Incorrect and unexpected inputs, input sequences, and timing.
- Reaction of software to system faults and failures (environmental stress testing)
- Go/No-Go and fail-safe modes.
- Operator interface.

Role of Software Safety Engineer

- Review test plans.
- Recommend tests based on hazard analyses, safety standards, checklists, previous accidents and incidents, interface analyses
- Specify conditions under which test is to be conducted.
- Review test results for any safety-related problems that were missed in analysis or other testing.
- Monitor tests for unexpected failure modes and hazardous states.

Limitations of Dynamic Analysis

- Testing for safety is essentially intractable.
- Accidents occur only infrequently and usually in ways not anticipated by designer or tester.
- Testing usually doesn't find requirements errors.

Uses for Dynamic Analysis

- Limitations don't mean don't test — just that there is a limit to the confidence that can be acquired through dynamic analysis.
- Also limits to static analysis. Need testing to verify:
 - Accuracy of model to constructed system
 - Satisfaction of assumptions underlying math techniques
 - Things not covered by static analysis, e.g., performance.
- Test planning can use the results of analysis.

Formal Verification

- Will not make complex systems simple. There is no magic here.
- Limitations:
 - May be same size as program or larger.
 - Often difficult to construct.
 - Therefore, likely to contain errors.
 - Some limited aspects can be mathematically proven.
 - Are these the errors most likely to be found in testing?
 - Are they likely to cause accidents?
- Probably little effect on safety unless aimed at safety constraints.

Operations

Change control

- Evaluate all changes for potential effect on safety
- Update all safety–related documentation
- Need to plan during development to make this feasible

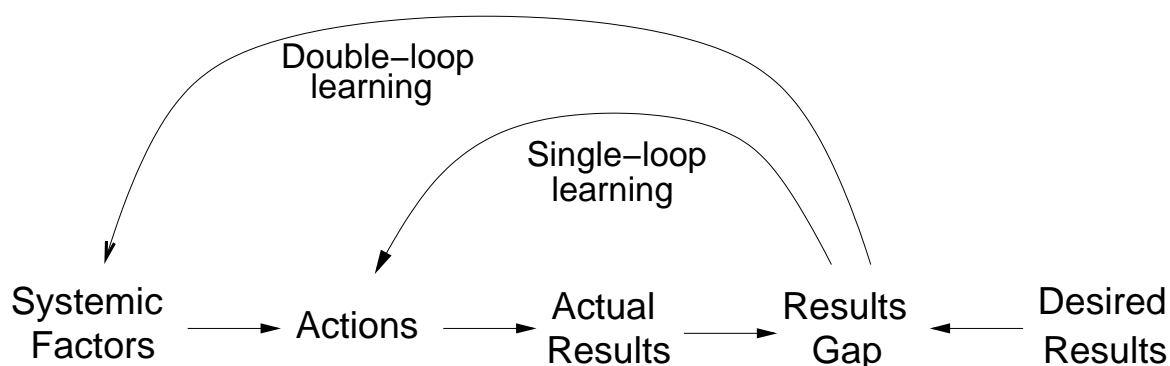
Performance Monitoring and Audits

- Data Collection
 - Use assumptions of analysis as preconditions for operations
 - Check whether operations have deviated
 - Just looking at incidents is not enough
- Data Analysis
 - Always perform root cause analysis
 - Do not assume problems are caused by hardware
 - Look at why operators made mistakes or deviated from written procedures (maybe it is system design that needs to be changed)

Performance Monitoring and Audits (2)

- Information Dissemination and Use
 - Integrate information into decision making tools and environment
 - Blame and discipline can lead to lack of reporting of problems
 - Distinction between a fixing orientation and a learning one

Single loop vs. Double loop learning



Organization/Management/Culture

It is not enough to talk about “absolute safety” and of “zero accidents.” There must also be proper organisation and management to ensure that actions live up to words.

British Department of Transport
*Investigation into the Clapham
Junction Railway Accident*

Safety Culture and Management

Definitions of "culture"

- A shared set of norms and values
- A way of looking at and interpreting the world and events around us (our mental model) and taking action in a social context.
- An ongoing, proactive process of reality construction (Morgan)
 - Organizations are socially constructed realities that rest as much in the heads of members as in sets of rules and regulations.
 - Sustained by belief systems that emphasize the importance of rationality: the "myth of rationality"

"helps us to see certain patterns of action as legitimate, credible, and normal, and hence to avoid the wrangling and debate that would arise if we were to recognize the basic uncertainty and ambiguity underlying many of our values and actions."

Safety Culture and Management (2)

- Safety culture is subset of culture that reflects general attitude and approaches to safety and risk management.
- Trying to change culture without changing environment in which it is embedded is doomed to failure
- Simply changing organizational structures may lower risk over short term, but superficial fixes that do not address the set of shared values and social norms are likely to be undone over time.
- "Culture of denial"
 - Risk assessment unrealistic and credible risks and warnings are dismissed without appropriate investigation.

Safety Culture and Management (3)

- Need to bring operational practices and values into alignment
 - Identify desired organizational safety principles and values
 - Establish organizational infrastructure to achieve values and sustain them over time.
 - Understand why operational practices have deviated from stated principles.
 - Make adjustments
 - Institute protections against future misalignments
- Goal is to create a culture and organizational infrastructure that can resist pressures against applying good safety engineering practices and procedures
- No one single safety culture in a large organization

Organizational Structure and Culture

- Structure drives behavior
- Where should safety activities be put?
 - Safety permeates every part of development and operations (e.g., engineering design, risk management, IV&V, quality assurance, operational performance monitoring, maintenance)
 - Need not be located in one place but common methods and approach will strengthen these disciplines
 - If distributed, need a clear focus and coordinating body
 - Basic principles:
 1. System Safety needs a direct link to decision makers and influence on design-making (influence and prestige).
 2. System Safety needs to have independence from project management (but not engineering)
 3. Direct communication channels are needed to most parts of the organization (oversight and communication)

Organizational Structure and Culture (3)

- Independent Technical Authority (CAIB)
 - Inside program but independent of Program Manager and schedule/budget concerns
 - Outside program to authorize and provide:
 - Tailoring or relaxing of safety standards
 - Amount and type of safety to be applied to program
 - Standards creation
 - External safety review
 - e.g. WSESRB

Organizational Structure and Culture (4)

- Oversight and Communication
 - Oversight vs. insight (transition usually done poorly)
 - Responsibility for safety cannot be delegated to contractors
 - Use of working groups for communication
 - Very effective in DoD
 - Different groups at different levels
 - Responsible for coordinating safety efforts at each level, reporting status of outstanding safety issues, providing information to other levels and to external review boards.

Cultural and Organizational Risk Analysis and Performance Monitoring

- Apply STAMP and STPA at organizational level plus system dynamics modeling and analysis
- Goals:
 - Evaluating and analyzing risk
 - Designing and validating improvements
 - Monitoring risk ("canary in the coal mine")
 - Identifying leading indicators of increasing or unacceptable risk

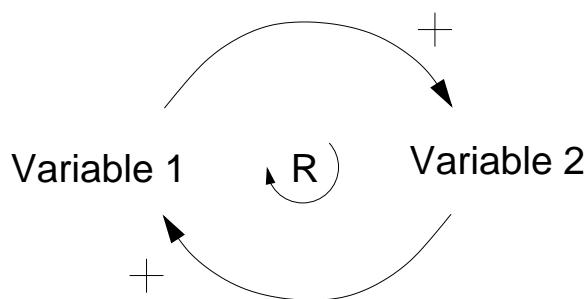
System Dynamics

- Created at MIT in 1950s by Forrester
- Used a lot in Sloan School (management)
- Grounded in non-linear dynamics and feedback control
- Also draws on cognitive and social psychology, organization
 - Cognitive and social psychology
 - Organization theory
 - Economics
 - Other social sciences
- Use to
 - Understand changes over time (dynamics of a system)
 - Design and evaluate policies for sustained improvement

System Dynamics (2)

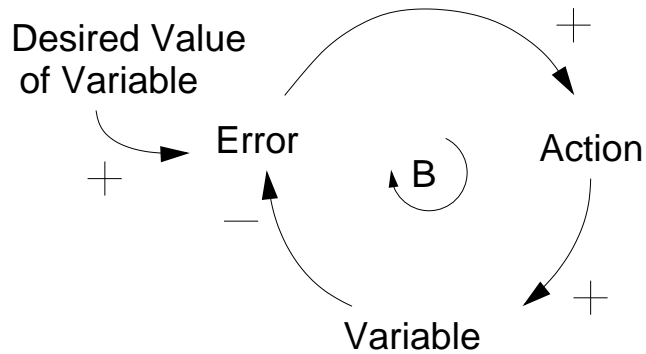
- Model system behavior using
 - Feedback (causal loops)
 - Stocks (levels)
 - Flows (rates)
 - Non-linearities created by interactions among components
- Behavior over time (the dynamics of a system) explained by interaction of positive and negative feedback loops
- Models constructed from three basic building blocks:
 - Positive feedback or reinforcing loops
 - Negative feedback or balancing loops
 - Delays

Reinforcing Loop



- Feeds on itself to produce growth or decline.
- Positive feedback loop in control theory
- Generates growth, amplifies deviations, reinforces change.
- Because initial growth often slow, may be unnoticed until becomes rapid.
At that point may be too late to control growth/decline.

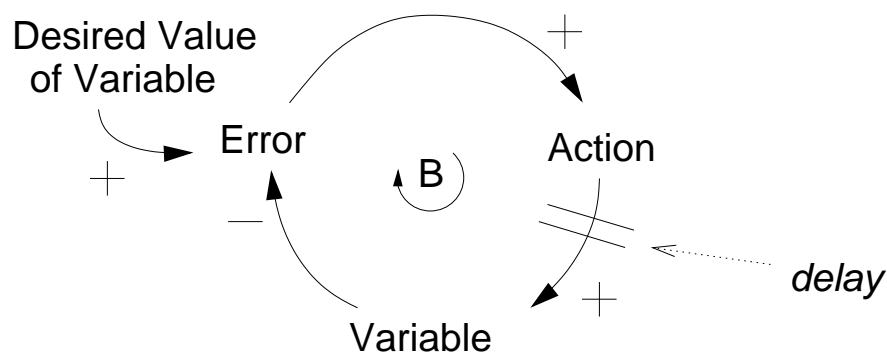
Balancing Loop



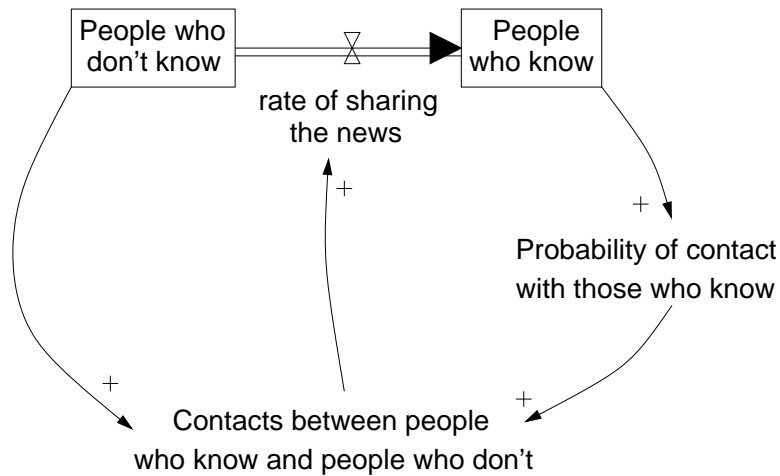
Structure that attempts to move a current state to a desired or reference state through some action

- Negative feedback loop in control theory.
- Difference between current state and desired state is perceived as an error.
- Action proportional to error taken to decrease error
 - Over time current state approaches desired state.
- Because action proportional to error
 - Current state initially rapidly approaches desired state
 - As error decreases, rate at which current state approaches desired state decreases

Balancing Loop with a Delay



- Used to model time that elapses between cause and effect.
- May result in unstable system behavior.
(Overcorrection because of inertia delaying changes)



$$\text{People who know (t)} = \int_0^t \text{Rate of sharing the news}$$

$$\text{People who know (0)} = 1$$

$$\text{People who don't know (0)} = 99$$

$$\text{People who don't know (t)} = \int_0^t - \text{Rate of sharing the news}$$

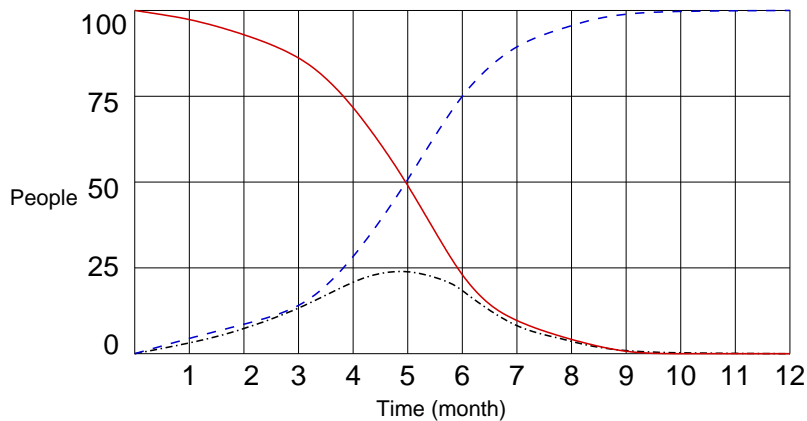
$$\text{Total people} = \text{People who don't know (t)} + \text{People who know (t)}$$

$$\text{Rate of sharing the news (t)} =$$

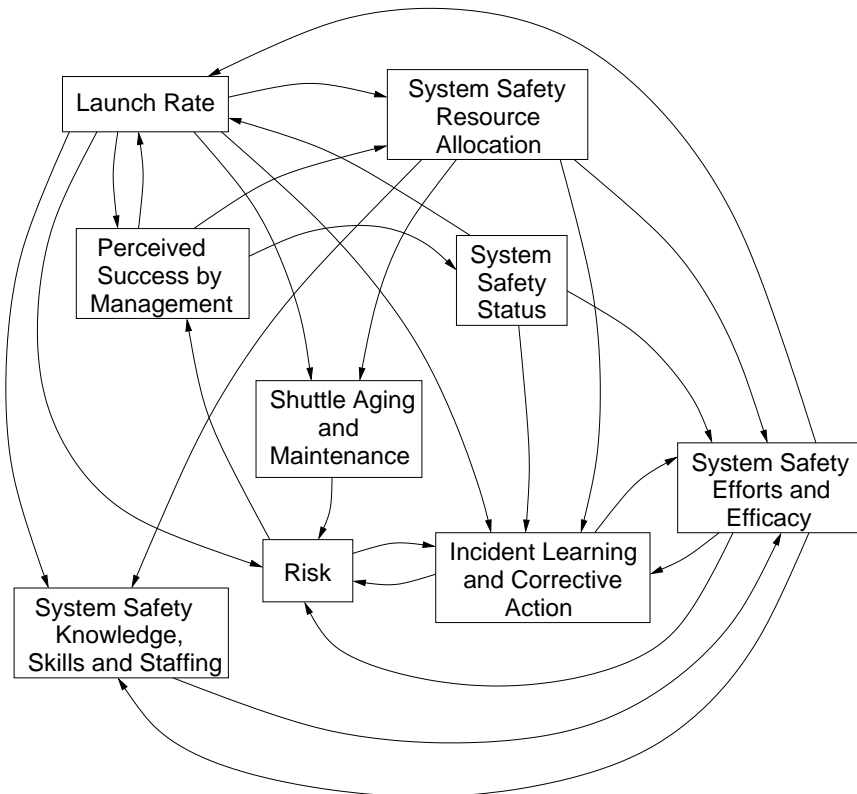
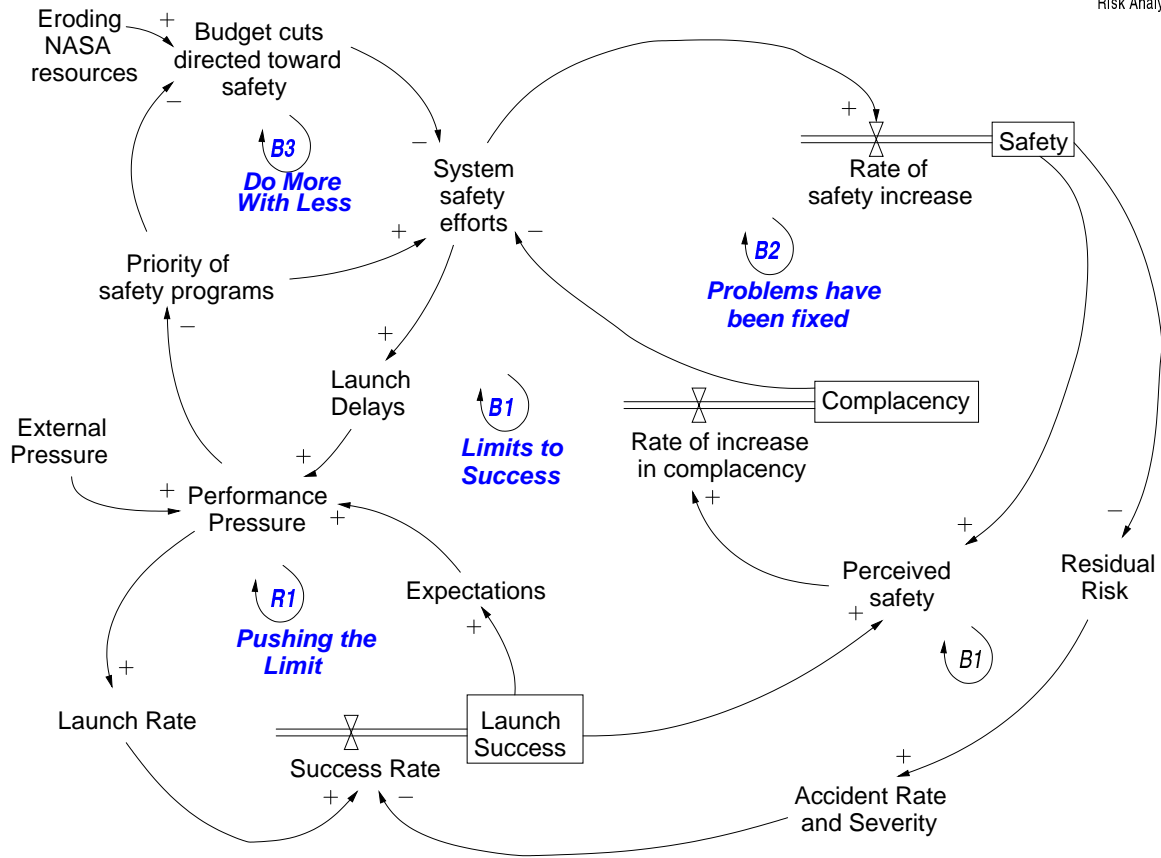
$$\text{Contacts between people who know and people who don't (t)}$$

$$\text{Contacts between people who know and people who don't (t)} =$$

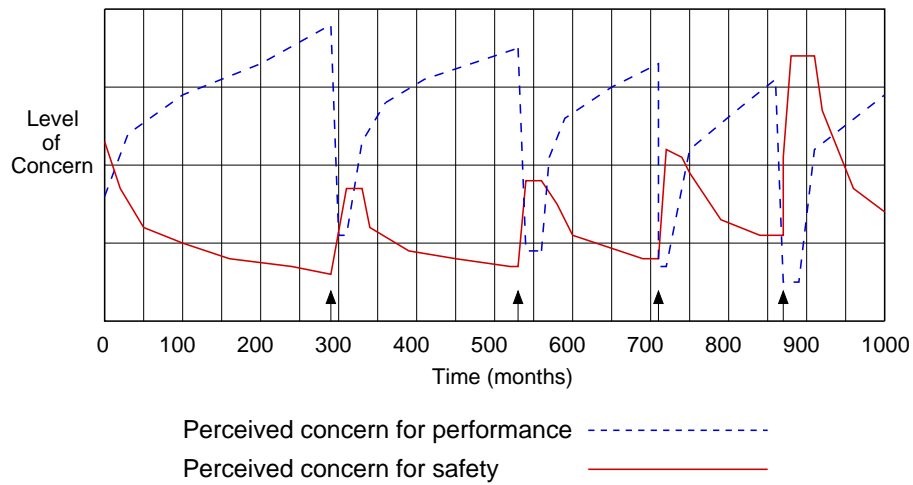
$$\frac{\text{People who don't know (t)} \times \text{People who know (t)}}{\text{Total People}}$$



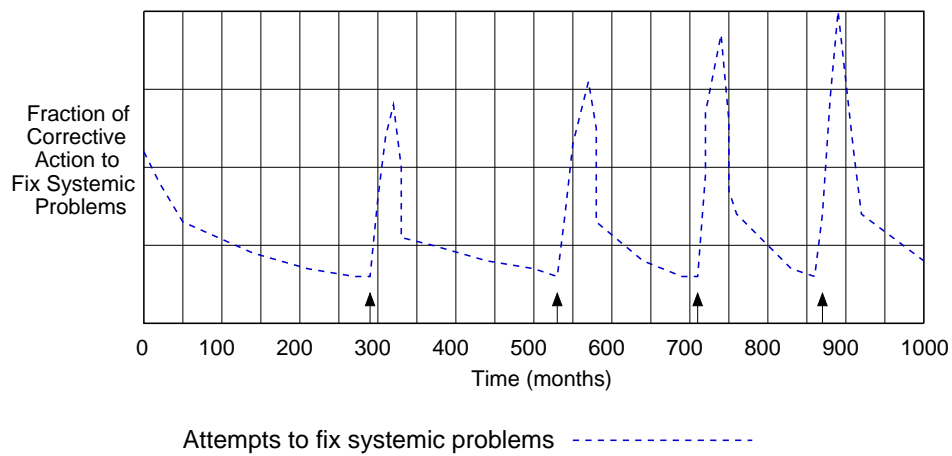
People who know - - - - -
 People who don't know - - - - -
 Rate of sharing the news - · - · - · -



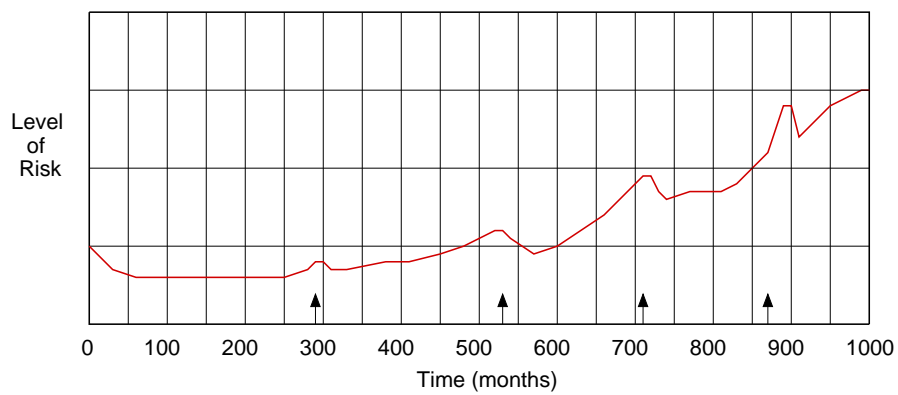
Accidents lead to a re-evaluation of NASA safety and performance priorities but only for a short time:



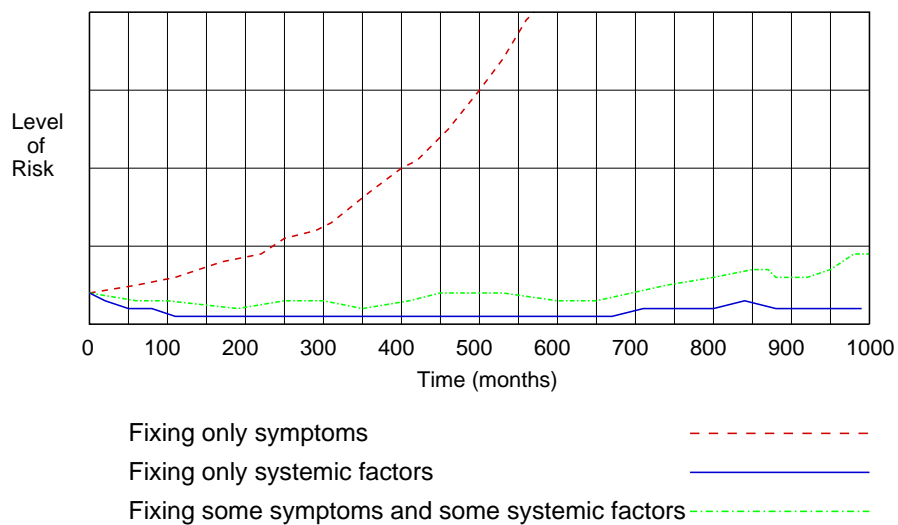
Attention to fixing systemic problems lasts only a short time after an accident



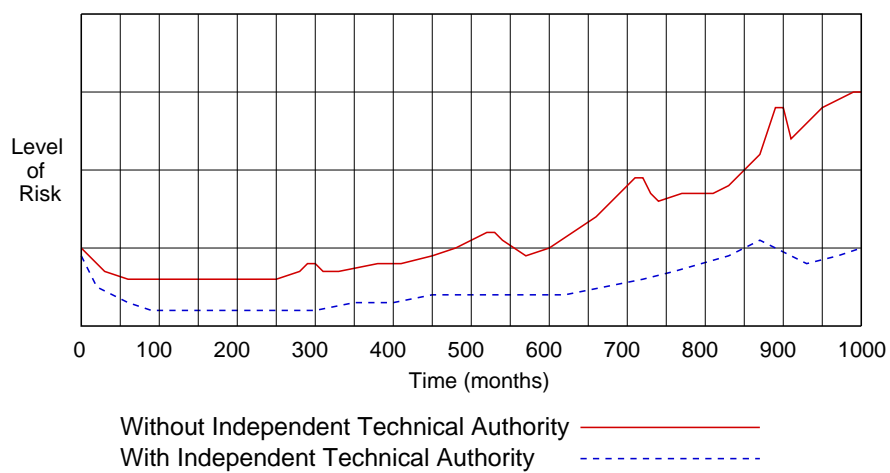
Responses to accidents have little lasting impact on risk



Scenario 1: Impact of fixing systemic factors vs. symptoms



Scenario 2: Impact of Independent Technical Authority



Rest of notes on risk analysis provided separately

Conclusions

We pretend that technology, our technology, is something of a life force, a will, and a thrust of its own, on which we can blame all, with which we can explain all, and in the end by means of which we can excuse ourselves.

T. Cuyler Young
Man in Nature

System Safety Process

Safety must be specified and designed into the system and software from the beginning.

- Program/Project Planning
 - Develop policies, procedures, etc.
 - Develop a system safety plan
 - Establish management structure, communication channels, authority, accountability, responsibility
 - Create a hazard tracking system
- Concept Development
 - Identify and prioritize system hazards
 - Generate safety-related system requirements and design constraints

System Safety Process (2)

- System Design
 - Apply hazard analysis to design alternatives
 - Determine if and how system can get into hazardous states
 - Eliminate hazards from system design if possible
 - Control hazards in system design if cannot eliminate
 - Identify and resolve conflicts between design goals
 - Trace hazard causes and controls to components (hardware, software, and human)
 - Generate component safety requirements and design constraints from system safety requirements and constraints

System Safety Process (3)

- System Implementation
 - Design safety into components
 - Verify safety of constructed system
- Configuration Control and Maintenance
 - Evaluate all proposed changes for safety
- Operations
 - Incident and accident analysis
 - Performance monitoring
 - Periodic audits

Software Safety Tasks

- Establish software safety management structure, authority, responsibility, accountability, communication channels, etc.
- Develop a software hazard tracking system and link to system hazard tracking system.
- Trace identified system hazards and system safety design constraints to software interface.
- Translate identified software-related hazards and constraints into requirements and constraints on software behavior.
- Evaluate software requirements with respect to system safety design constraints and other safety-related criteria.

Software Safety Tasks (2)

- Design software and HMI to eliminate or control hazards.
Design safety into the software.
 - Defensive programming
 - Assertions and run-time checks
 - Separation of critical functions
 - Elimination of unnecessary functions
 - Exception handling
 - etc.
- Analyze the behavior of all reused and COTS software for safety (conformance with safety requirements and constraints)
- Trace safety requirements and constraints to the code.
Document safety-related design decisions, design rationale, and other safety-related information.

Software Safety Tasks (3)

- Perform special software safety analyses
e.g.
 - human-computer interaction and interface
 - formal or informal walkthroughs or proofs
 - interface between critical and non-critical software
- Plan and perform software safety testing.
Review test results for safety issues.
Trace identified hazards back to system level.
- Analyze all proposed software changes for their effect on safety.
- Establish feedback sources. Analyze operational software and relate to hazard analysis and documented design assumptions.

Conclusions

- There are no simple solutions. Requires:
 - Time and effort
 - Special knowledge and experience
- Our most effective tool for making things safer is simplicity and building systems that are intellectually manageable.
- Complacency is perhaps the most important risk factor.
- Safety and reliability are different— don't confuse them.

Conclusions (2)

- The safety of software cannot be evaluated by looking at it alone. Safety can be evaluated only in the context of the system in which it operates.
- Building safety into a system will be more effective than adding protection devices onto a completed design.
- The job of the system safety engineer is to identify safety constraints and ensure they are enforced in design and operations. System safety must work closely with the system designers.
- Placing all responsibility for safety on human operators does not ensure safety. It merely provides a scapegoat.

Conclusions (3)

- Our technology must be used by humans. Human error can be reduced by appropriate design.
- The earlier safety is considered in development, the better will be the results.
- To prevent accidents, we will need to remove systemic factors.
Concentrating only on technical issues and ignoring managerial and organizational deficiencies will not result in effective safety programs.
- Safety is a system problem and can only be solved by experts in different disciplines working together.

A life without adventure is likely to be unsatisfying, but a life in which adventure is allowed to take whatever form it will, is likely to be short.

Bertrand Russell