

# System and Software Safety

Nancy G. Leveson

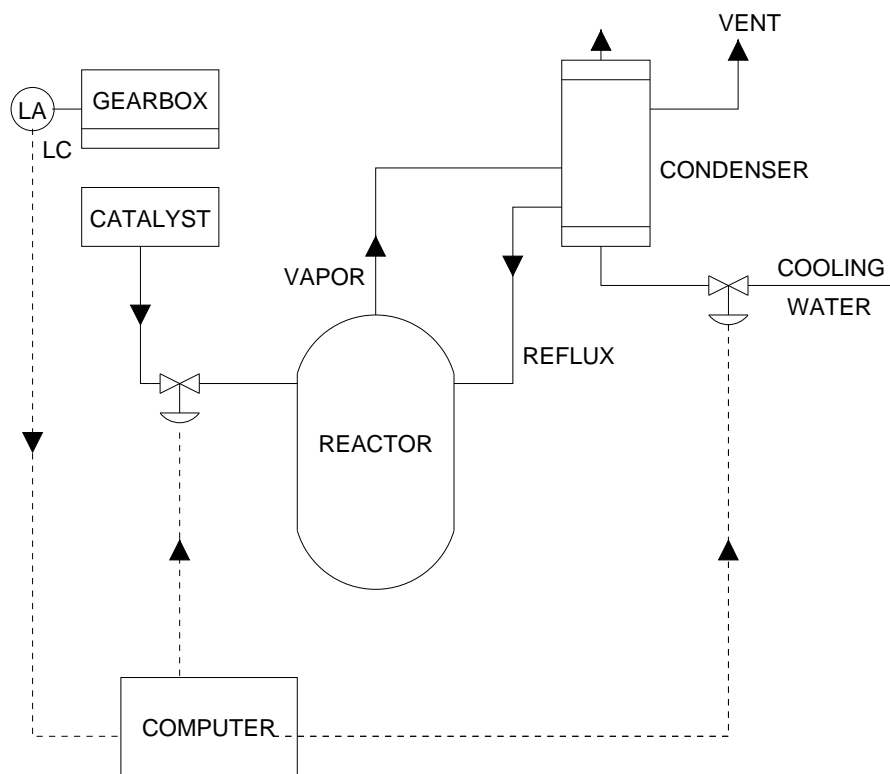
MIT Aero/Astro Dept.  
Safeware Engineering Corp.

© Copyright by the author, June 2001. All rights reserved. Copying without fee is permitted provided that the copies are not made or distributed for direct commercial advantage and provided that credit to the source is given. Abstracting with credit is permitted.

## 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

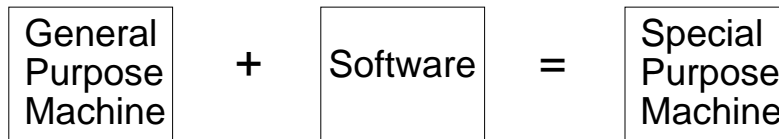
- Complexity is a moving target
- The underlying factor is intellectual manageability
  1. A "simple" system has a small number of unknowns in its interactions within the system and with its environment.
  2. A system is intellectually unmanageable when the level of interactions reaches the point where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

## 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

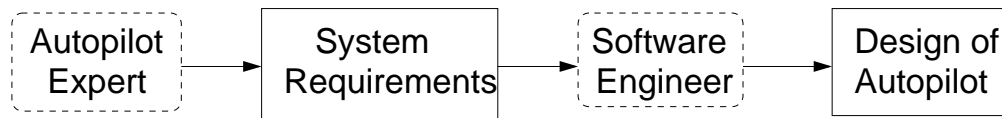
- Software is the resting place of afterthoughts
- No physical constraints
  - To enforce discipline on design, construction and modification
  - To control complexity
- So flexible that start working with it before fully understanding what need to do
- “And they looked upon the software and saw that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

## 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.

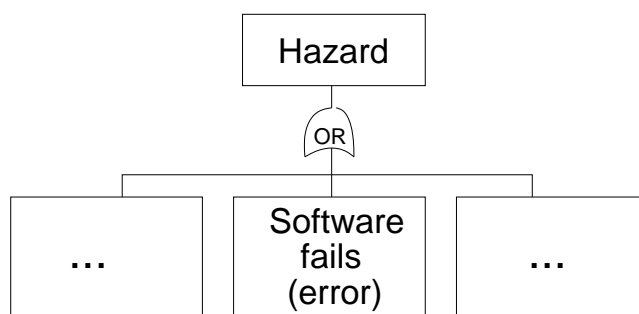
# Abstraction from Physical Design

- 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

## Typical Fault Trees



Hazard Cause	Probability	Mitigation
Software Error	0	Test software

## Black Box Testing

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

- Need to test every possible input

$x := y * 2$

(since black box, only way to be sure to detect this is to try every input condition)

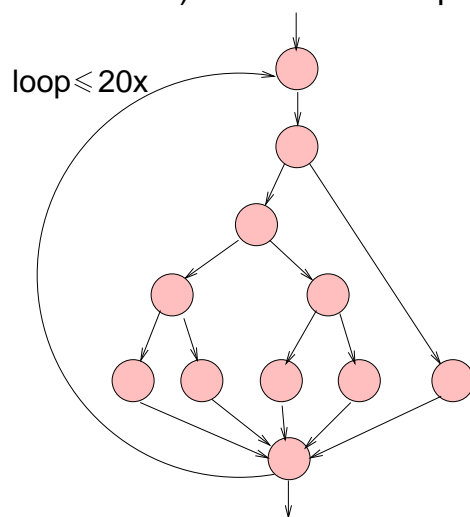
- Valid inputs up to max size of machine (not astronomical)
  - Also all invalid input (e.g., testing Ada compiler requires all valid and invalid programs)
  - If program has “memory”, need to test all possible unique valid and invalid sequences.
- So for most programs, exhaustive input testing is impractical.

## 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.

## Mathematical Modeling Difficulties

- Large number of states and lack of regularity
- Lack of physical continuity: requires discrete rather than continuous math
- Specifications and proofs using logic:
  - May be same size or larger than code
  - More difficult to construct than code
  - Harder to understand than code

Therefore, as difficult and error-prone as code itself

- Have not found good ways to measure software quality



## 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

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

## 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.

## The Problem to be Solved

- The primary safety problem in computer-based 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."

## Does Software Fail?

**Failure:** Nonperformance or inability of system or component to perform its intended function for a specified time under specified environmental conditions.

A basic abnormal occurrence, e.g.,

- burned out bearing in a pump
- relay not closing properly when voltage applied

**Fault:** Higher-order events, e.g.,

- relay closes at wrong time due to improper functioning of an upstream component.

All failures are faults but not all faults are failures.

## 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 software-related 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.

## 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!***

## 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.

## A Little Systems Theory

Systems theory can act as an alternative to reliability theory for dealing with safety.

## Ways to Cope with Complexity

- Analytic Reduction (Descartes)
  - Divide system into distinct parts for analysis purposes.
  - Examine the parts separately.
  
- Three important assumptions:
  1. The division into parts will not distort the phenomenon being studied.
  2. Components are the same when examined singly as when playing their part in the whole.
  3. Principles governing the assembling of the components into the whole are themselves straightforward.

## 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.

## What about software?

- 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.

## Systems Theory

- Developed for biology (Bertalanffy) and cybernetics (Norbert Wiener)
  - For systems too complex for complete analysis
    - Separation into non-interacting subsystems distorts results
    - Most important properties are emergent.
  - and too organized for statistical analysis
- Concentrates on analysis and design of whole as distinct from parts (basis of system engineering)
  - 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.

## Systems Theory (2)

- Two pairs of ideas:
  1. Emergence and hierarchy
    - Levels of organization, each 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.

## Systems Theory (3)

2. Communication and control
  - Hierarchies characterized by control processes working at the interfaces between levels.
  - A control action imposes constraints upon the activity at one level of a 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

# An Overview of The Approach

*Engineers should recognize that reducing risk is not an impossible task, even under financial and time constraints. All it takes in many cases is a different perspective on the design problem.*

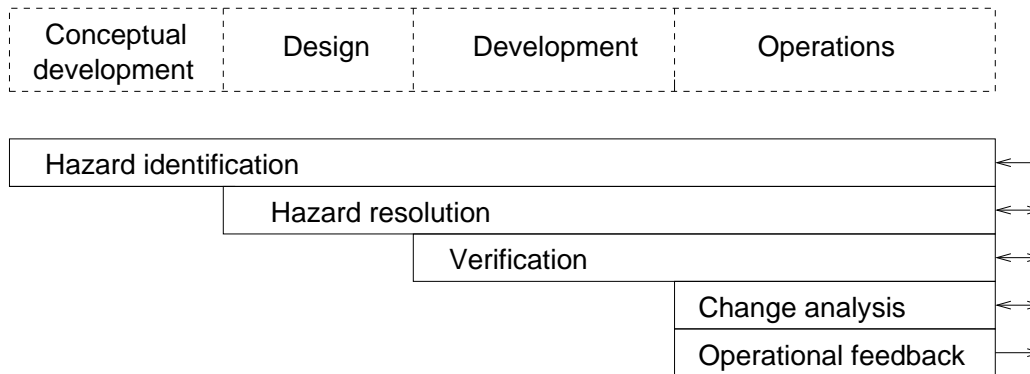
Mike Martin and Roland Schinzinger  
*Ethics in Engineering*

## 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)

- Hazard analysis and control is a continuous, iterative process throughout system development and use.



- 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

## System Safety Engineering

- Emphasizes building in safety rather than adding it on to a completed design.
- Looks at systems as a whole, not just components
- Takes a larger view of hazards than just failures.
- Emphasizes hazard analysis and design to eliminate or control hazards.
- Emphasizes qualitative rather than quantitative approaches.

# 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).

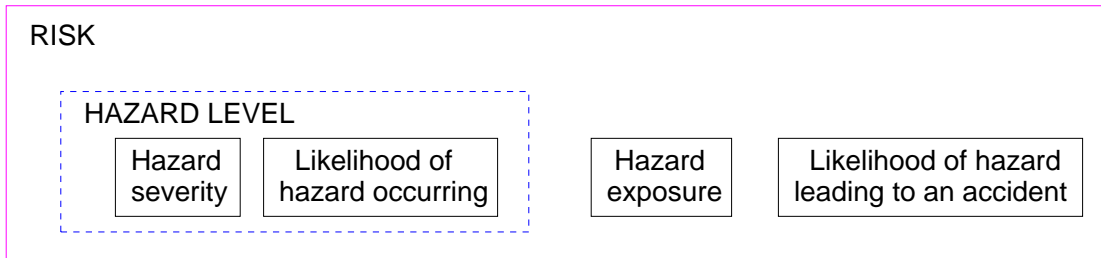
Note that a hazard is NOT equal to a failure.

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

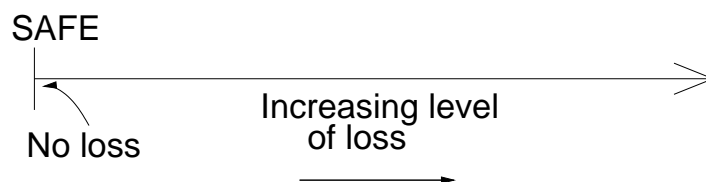
*C.O Miller*

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

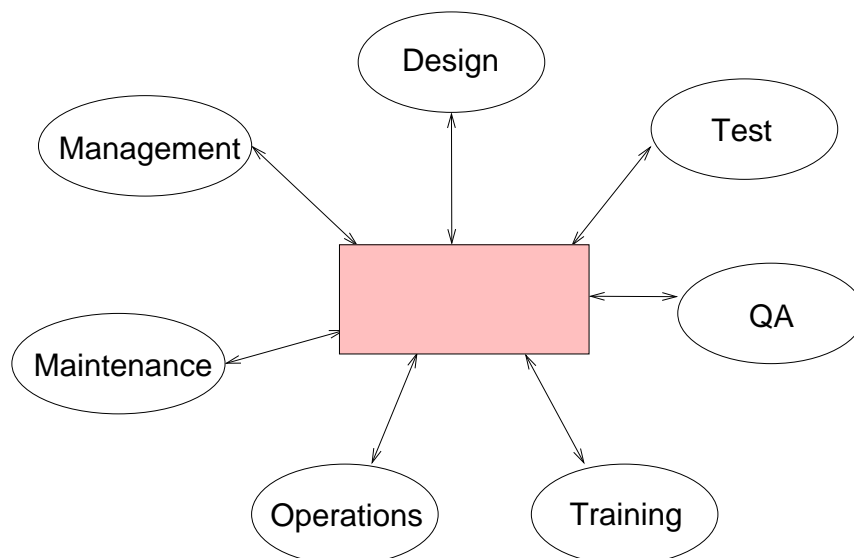
**Risk:** The hazard level combined with the likelihood of the hazard leading to an accident plus exposure (or duration) of the hazard.



**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.

## Hazards must be translated into design constraints.

HAZARD	DESIGN CRITERION
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 PHA for 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.

## Example PHA for ATC Approach Control (2)

HAZARDS	REQUIREMENTS/CONSTRAINTS
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.

HAZARDS	REQUIREMENTS/CONSTRAINTS
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.