Software Safety
(Safeware)

István Majzik

Technical University of Budapest
Department of Measurement and Information Systems

September 14, 2000

Overview

- Terminology and basic concepts
- Design process
- Hazard analysis
  - Checklist
  - Hazard indices
  - Fault tree analysis
  - Event tree analysis
  - Cause-consequence analysis
  - Hazard and operability analysis
  - Interface analysis
  - Failure modes and effects analysis
  - Failure modes, effects and criticality analysis
  - State machine hazard analysis
  - Human error analysis
- Risk reduction
  - Hazard elimination
  - Hazard reduction
  - Hazard control
  - Damage reduction
- Software safety analysis
  - Specification completeness and consistency checking

Part I

Terminology and basic concepts

Terminology

Accident: undesired and unplanned event that results in a specified level of loss
(unplanned - not as sabotage)

Incident: event that involves no loss, but with the potential of loss in other circumstances

Hazard: state of a set of conditions of the system that together with conditions of the environment will lead inevitably to an accident

- defined in respect of the environment (hazard in computer systems: react to environment)
- depends on system boundaries (flammable vapor can not be separated from ignition)
- characteristics:
  - endogenous: inherent in the system
  - exogenous: external phenomena (e.g. lightning)
- hazard level:
  - severity (damage)
  - likelihood

Terminology (continued)

Risk: hazard level combined with

- the likelihood of leading to an accident and hazard duration (the longer $\rightarrow$ higher risk)
  (relationship between hazard and accident)
- Risk analysis: involves analysis of environmental conditions and hazard duration

Safety: freedom from accidents

- (a relative definition: enabling ”acceptable” loss - by whom it is judged?)
- it can only be approached asymptotically

General issues

- safety = building in safety, not adding it to a complete system
  (part of the initial phases - minimal negative impact)
- safety deals with systems as a whole
  (safety is not a component property)
  (interfaces, effects on another component are important)
- larger view of hazards than failures
  (failure $\neq$ hazard)
  (hazard $\leftarrow$ also in the case of functioning components)
- analysis rather than past experience and standards
  (pace of change not allows to accumulate)
  (prevent before they occur!)
- qualitative rather than quantitative approach
  (early stages: no quantitative information)
  (accuracy of quantitative models is questionable; e.g. accidents are caused by failures, testing is perfect, failures are random and independent, good engineering)
- safety recognizes the importance of tradeoffs and conflicts in design
  (safety is a constraint)
- safety is more than system engineering
  (also political, social, management, cognitive psychological issues)

Design for safety (overview)

- hazard elimination
- hazard reduction:
  minimize the occurrence
- hazard control:
  mitigate the effects if the hazard has occurred
  Examples:
  passive control (do not require a positive action to prevent
  hazard if the control breaks, the default action is to prevent
  gravity switches (railway semaphore)
- damage reduction:
  isolation, emergency actions

Software safety

Software safety: software will execute without contributing to hazards

- exhibiting behavior (output, timing)
- failing to recognize and handle hazards

Safety-critical software: contribute to the occurrence of hazardous system state

Safety-critical functions: correct/incorrect/lack of operation may contribute in hazard

Software errors: to deal with them by

- correct requirements (safe, all behaviors)
- correct coding (theoretically possible)
- software fault tolerance (not enough)
- apply system safety techniques (analysis, elimination, reduction, ...)

Accident models

Energy model: uncontrolled and undesired release of energy (chemical, thermal, electrical etc.)

- to reduce: barriers, flow control
- accidents:
  - energy transformation accident: energy is transformed to an other object
  - energy deficiency action: energy is not available
- consequence: software can not cause an accident (but together with hardware)
- limited scope:
  - limited to energy processes
  - loss of mission is not treated

Domino model: emphasizing unsafe acts over unsafe conditions removing a domino will prevent the accident

1. ancestry or social environment
2. fault of a person
3. unsafe act or condition
4. accident

Accident models (continued)

More general model

1. management structure (organization, objectives, operations)
2. operational errors (supervisor behavior)
3. tactical error (employee behavior, work conditions)
4. accident

Chain-of-events model

- multiple factors (actions, conditions) are treated
- if the chain can be broken, the accident will not happen
- AND, OR relationships between actions $\rightarrow$ logic tree
- actors can be involved: parallel horizontal event tracks by the actors
- external influences: perturbations; actors have to adapt; unable to adapt $\rightarrow$ accident
- correction of the path can prevent accident
- role of change is important (non-routine operation: TMI, Chernobyl)

Accident models (continued)

System theory models: what went wrong within the system to allow accident

- accident: interaction which violates constraints
  lack of constraints
  - boundary areas (interfaces)
  - overlap zones (influence on the same object)
  - asynchronous evolution of subsystems
- dynamic equilibrium: feedback loops and control

Accident: disturbations are not handled correctly

Human task and error models

Hard to compute quantitative measures

Part II

Design process

The system and software safety process

Integrating function: safety considerations are involved early

Managing safety: POLC: plan, organize, lead, control

- responsibility
- authority (right to command)
- accounting (measurement of results)

Overview:

- Conceptual development
- System design
- System production and deployment
- System operation

Conceptual development task

Essential groundwork:

- develop system safety program plan
  - identifying software-related hazards: turn to requirements
  - consistency of safety constraints with requirements
  - identify safety-critical parts
  - trace safety requirements, develop a tracking system
  - develop test plans
  - assembly safety-related information into documentation
- establish information and documentation files
- establish hazard auditing and log file (tracking system)
- review applicable documents (similar systems)
- establish certification and training
- participate (safety engineer) in system concept formation
- define the scope of analyses: objective, basis, hazard types, required standards
- identify hazards and safety requirements
- identify design, analysis and verification requirements
- establish organizational structure (working groups etc.)

System design task

Design phase:

- update analyses (update previous analysis in new design phase)
- participate in system tradeoff studies (design decisions)
- ensure incorporation of safety requirements
- ensure identified hazards being eliminated
- identify safety critical components
- trace system hazards into components/subsystems $\rightarrow$ software
- review test and evaluation procedures, training
- evaluate design changes
- document safety information

System production and deployment tasks

- update hazard analyses
- perform system level safety evaluation
- perform safety inspections
- incorporate safety related info in documentation
- review change proposals
- perform a final evaluation

System operation tasks

- update procedures (new hazard modes)
- maintain information feedback system (logs, reports)
- conduct safety audits (periodically + triggered by needs)
- review changes and maintenance

Case study of a system safety project: Zurich underground rail station

Environment: electric rail system: platform+tracks, ramp, tunnel, shopping mall, stairs, escalators, elevators, office building

Process:

- safety personnel + involving external experts (also an insurance company)
- more information in design space → more detailed analysis
- complex analysis (maximum depth) at defined stages

1. Definition of scope: safety personnel

- project documentation → information to be used

2. Hazard identification: system engineers

- project documentation → HAZARD CATALOG

hazard	cause	level	effect	category
...	...	...	...	...

Case study (continued)

3. Evaluate hazard levels: system engineers

- hazard catalog $\rightarrow$ RISK MATRIX
- 6 levels: probability of occurrence: frequent, moderate, occasional, remote, unlikely, impossible
- 4 effects: catastrophic, critical, marginal, negligible
- risk matrix:

hazard levels	**hazard effects**		
	catastrophic	critical	...
frequent			
moderate			
...			

4. Review hazard levels: interdisciplinary group

- risk matrix $\rightarrow$ revised risk matrix
- interdisciplinary knowledge is involved (transport, psychology)

Case study (continued)

5. Determine protection level: management

- revised risk matrix → protection level
- protection level: line in risk matrix (priorities, cost limitations)
  risk reduction: above the line
- types of risk reduction:
  - (re)design required
  - hazard must be controlled
  - hazard control desirable if cost effective

6. Revise hazards and risk matrix: experts (specialists)

- hazards in protection level → corrected hazard catalog and risk matrix

Case study (continued)

7. Recommend risk reduction measures: experts (specialists)

- expert knowledge $\rightarrow$ catalog of corrective actions (RISK REDUCTION CATALOG)

risk profile	hazard	corrective action	by/date
...	...	...	...

- corrective actions above department authority:
  - sent to upper management level with cause, effect, action, cost
  - decision $\rightarrow$ sent back to involved departments
  - action taken $\rightarrow$ crossed off in the list; open items are visible

8. Quality assurance check of risk reduction measures: responsible experts

- catalog of corrective actions $\rightarrow$ verified catalog

9. Review of progress: management + safety personnel

- verified catalog $\rightarrow$ fact sheet
  - fact sheet for non-experts to document progress
  - remaining non-reduced risk: further, deeper analysis

Part III

Hazard analysis

Basics

Central role, continuous effort

Phases of design:

- in development: identify potential hazards
- in operation: improve safety
- in licensing: demonstrate safety
  evaluate the effects of hazards that cannot be avoided

Types:

- Preliminary hazard analysis: early phase
  - identify critical system functions
- System hazard analysis: mature design
- Subsystem hazard analysis: subsystem design phase
  - studies of possible hazards
  - identifying hazards
  - determine causes, effects
  - find ways how to avoid/eliminate/control
  - planned modifications
- Operating and support hazard analysis: system use and maintenance
  - human-machine interfaces

Basics (continued)

Qualitative analyses (quantitative: effect of incorrect measures)

- General features:
  - continual and iterative
- Steps:
  - definition of objectives, scope, system, boundaries
  - identification of hazards: magnitude, risk
  - collection of data (historical record, standards)
  - ranking of hazards
  - identification of causal factors
  - identification of preventive measures: design criteria
  - verification of implementation
  - quantification of unresolved hazards and risks
  - feedback and operational experience

Basics (continued)

Hazard level:

MIL-STD-822b

- I: catastrophic (death, system loss)
- II: critical (injury, major system damage)
- III: marginal (minor injury)
- IV: negligible

NASA:

- 1: loss of life or vehicle
- 2: loss of mission
- 3: all others

Design criteria (used to derive requirements)

- train starts with open door: must not be capable of start with open doors
- door opens while train moves: doors must remain closed
- etc.

Basics (continued)

General types of analysis:

- forward (inductive) search
  - initiating event is traced forward in time/causality
  - look at the effects
  - problem: state space
- backward (deductive) searches
  - final event is traced back
  - accident investigations
- bottom-up search: subsystems are put together
  - problem: combinations of subsystems
- top-down search: higher level abstractions are refined (subsystems, components)

Problems: unrealistic assumptions

- good engineering, testing, etc.
- discrepancy between documentation and system
- changing conditions

Models and techniques

Overview

- Checklist
- Hazard indices
- Fault tree analysis
- Event tree analysis
- Cause-consequence analysis
- Hazard and operability analysis
- Interface analysis
- Failure modes and effects analysis
- Failure modes, effects and criticality analysis
- State machine hazard analysis
- Human error analysis

Checklist

- to check earlier experiences: they guide thinking
- dynamical update is needed
- phases:
  - hazard analysis: not to overlook known hazards
  - design: conformance to existing codes, standards
  - operational: periodic audits
- problem: grows large and difficult to use
  - false confidence about safety (incomplete checklist)

Hazard indices

- measure fire, explosion, chemical hazards (in processes)
- Dow Index: 1964
- plant = units, measured on the basis of tables/equations (fireable material etc.)
- problem: mainly for process industry, or in early stages (minimum data)
- only hazard level, no causes/elimination/reduction

Fault tree analysis

- aerospace, avionics, electronics industry
- analyzing causes of hazards (not to identify them)
- Boolean logic methods are used
- top-down method:
  - top level: foreseen, identified hazard
  - intermediate level: events necessary and sufficient to cause event shown at the upper level
  - pseudo-events: combination of the sets of primary events
  - primary events: no further development is possible (resolution limit)
- analysis:
  - reducing pseudo-events
  - simplifying Boolean expressions
  - show combinations sufficient to hazard
  - frequency (probability.) of the hazard based on probabilities of primary events
- basic steps:
  1. system definition
  2. fault tree construction
  3. qualitative analysis
  4. quantitative analysis

Fault tree analysis (continued)

1. System definition

- determining top event (hazard) $\rightarrow$ for all significant top events
  initial conditions
  existing events, impermissible events
- using: functional / flow diagrams, design representation

2. Fault tree construction

- elements: top event + causal events + logical relations
- graphical representation: symbol set, readability (underlying: Boolean algebra, truth table)
  - AND gate: causes of the event above
  - OR gate: re-expressions of the event above
  - NOT (inhibit) gate: used to express ”both” property
- automatic techniques: mainly for hardware (DF-like)

Fault tree analysis (continued)

3. Qualitative analysis

- reduce the tree to an equivalent form
- cut sets: relationships primary and top events
- minimal cut set: cannot be reduced further
- tree: OR gate + minimal cut sets (including the same event is possible)
- identify weakness: by ranking of primary events (importance: structure, occurrences in tree)

4. Quantitative analysis

- tree: sum of the probabilities of (disjunct) minimal cut sets
- cut set: product of probability. of primary events
- problem: events in multiple cut sets
- probability. density functions $\rightarrow$ Monte-Carlo simulation

Fault tree analysis (continued)

Properties:

- fault tree for software:
  - after the implementation, with manual assistance
  - only qualitative analysis
- phase in life cycle:
  - after implementation, proving safeness
  - early phases: problem of incomplete specification
- advantages:
  - helps the understanding of the system
  - identifying scenarios leading to hazards
  - minimal cut trees: potential weak points
    * small number of events, single-point failures
    * components in multiple cut sets: important effects
    * independence of events: common cause failures
      common influencing factors, to be reduced
      fault propagation (domino)

Fault tree analysis (continued)

Properties:

- limitations of qualitative analysis:
  - constructed after the implementation (detailed design)
  - cause and effect relationship and little more
  - simplified model, without
    - time- and rate-dependent events
    - partial failure
    - dynamic behavior
  - ordering and delay is not covered (fault tree is a snapshot)
    $\rightarrow$ DELAY node is required $\rightarrow$ loss of simplicity
  - sequence of events is not handled
  - multiple phases of system operation requires separate trees
- limitations of quantitative analysis:
  - common-mode failures
  - input data is unavailable, unrealistic

Event tree analysis

Decision tree formalism:

- forward analysis to find effects of an event, determine all sequences
  - initial state: failure of a component
  - next states: other system components
  - ordering: chronological, from left to right
  - decision: success/failure of other components
  - path probability: product of event/state probabilities
- reduction: eliminate illogical/meaningless events
- timing issues: phased-mission analysis
- example: failure + protection system components in nuclear station
- phase in life cycle: after the design is completed

Event tree analysis (continued)

Advantages:

- fault tree: snapshot of the system state; event scenarios combinations of component failures leading to hazard
- event tree: sequences of events; notion of continuity, ordering
- useful:
  - analyzing protection systems
  - identifying top events (for FTA)
  - displaying accident scenarios

Limitations:

- complexity
- separate tree for each initiating event
- multiple events - a problem
- ordering of events is critical

Cause - consequence analysis

Both time dependency and causal relationship:

- procedure:
  1. selection of a critical event
  2. backward search for factors that cause
  3. propagation of effects of the critical event
- attached to a consequence chart
  - cause charts: alternative prior event sequences and conditions
  - fault trees: for events and conditions
- table of symbols:
  - events and conditions
  - gates between events, vertices between conditions
  - decision boxes
- automatic construction is possible

Advantages:

- shows sequence of events (sequential control)
- combinations of events (additional event trees)

Disadvantage:

- separate diagrams for each critical event

Hazards and operability analysis

Developed for chemical industry:

- accidents are caused by deviations from the design / operating conditions.
- procedure:
  - identify all possible deviations
  - identify hazards associated with the deviations (consequences)
  - identify causes of deviations
  - systematic search: defined by a flowchart
- guide-words: applied to any variables of interest (flow, temperature, time)
  - NO, NONE: result is not achieved (e.g. no flow)
  - MORE: more result than should be (e.g. more flow)
  - LESS: less result than should be (e.g. less flow)
  - AS WELL AS: additional activity, more components
  - PART OF: only some of the design intentions are achieved (e.g. mix)
  - REVERSE: opposite of what was intended
  - OTHER THAN: something different happens

Hazards and operability analysis (continued)

Phase in life cycle:

- after the design documentation is available
- hazards are controlled by additional devices (detector, emergency valve etc.)

Advantage:

- simplicity, easy to use

Disadvantage:

- labor intensive, experts of the process are needed

Interface analyses

- structured walk-through, to examine the propagation of faults
- output types:
  - no
  - degraded
  - erratic
  - excessive
  - unprogrammed output
- undesired side effects

Failure modes and effects analysis

Developed for reliability analysis:

- procedure:
  - list all components with failure modes and probabilities
  - identify the effects on other components/system
  - forward search
  - system failure modes are calculated with probability
- input: failure probabilities (based on statistical data)
- output: tabular form

component	failure mode	failure prob.	% failures	effects (prob.)
...	...	...	...	...

- phase in life cycle: hardware items are identified

Advantages:

- identifies redundancy, fail-safe design, single point of failure
- spare part requirements

Disadvantages:

- all failure modes have to be known
- effects of multiple failures?

Failure modes, effects and criticality analysis

- FMEA extended with failure criticality data (rankings 1..10 etc.)
- preventive/corrective actions are also described

component	failure mode	failure prob.	criticality	effects	actions
...	...	...	...	...	...

State machine hazard analysis

- state machine: states + transitions + conditions + actions
- safety analysis: determine if the model contains hazardous states
  - theoretical: initial state $\rightarrow$ forward to states (computation tree)
  - practical: search backward to determine how to avoid hazardous state
- for hardware and also for software;
- safety and fault-tolerance analysis
- phase in life cycle: at any stage where a state-machine model is available
- advantages:
  - automated analysis
  - close to the view of engineers
- disadvantages:
  - logic and algebraic models and languages:
    - hard to understand and use
    - (external experts can not be involved)
    - mathematical proofs are not understood by reviewers
  - state space explosion in real systems
    hierarchical view is required (statechart)

Human error analysis

- Task = series of actions
- Qualitative techniques: examine for each action:
  - criticality
  - mental and physical demands
  - possible failures (forget, wrong ordering)
  - performance deviations (too slow, too fast)
  - equipment availability
- Quantitative techniques:
  - assign probability of human errors
  - factors that are effective:
    - psychological stress
    - quality of controls and displays (human engineering)
    - quality of training
    - quality of (written) instructions
    - coupling of human actions (dependencies)
    - personnel redundancy (inspectors)
  - probability by data collection in documented environments
  - safety analysis by event tree (path probabilities)
  - emergency: greater probabilities!
    - best: repetitive actions, long response time
    - worst: emergency, short time, complex tasks

Part IV

Risk reduction techniques

Basics

Safety analysis data have to be used in the design process.

In early phases of development:

- to be efficient
- poorly designed additional modules may increase risk
- additional efforts like operators may fail or will be tricky

Software specialties:

- new hazards
  - safety dependent on software errors
  - software errors are difficult to tolerate, they are unpredictable
  - hardware is much more simple: it may fail into a well-known state (short/open/stuck-at etc.)
- new possibilities to be more powerful

Design process

Two basic approaches:

1. Standards and experiences:

- for hardware it is well-defined: how to use a valve, electrical standards etc.
- no standard for software
  reliability, maintainability standards may even increase risk
  no generic software hazards

2. Guided by hazard analysis:

- identify software-related safety requirements and constraints
- identify parts of software which controls safety-critical operations
- elaborate behavior in erroneous states
- formal technique: data-flow based analysis
  $\rightarrow$ identification of critical nodes
  $\rightarrow$ formal safety constraints
  $\rightarrow$ design to be certifiable + verifiable
- documentation: record of safety-related decisions + assumptions
  $\rightarrow$ to be taken into account in software update

Risk reduction procedures

Overview:

1. Hazard elimination: Eliminating the hazardous state or the negative consequences
   - substitution
   - simplification
   - decoupling
   - elimination of specific human errors
   - elimination of hazardous materials or conditions
2. Hazard reduction
   - design for controllability
   - barriers: lockout, lockin, interlock
   - failure minimization: safety factors and margins, redundancy
3. Hazard control: If a hazard occurs, reducing the likelihood leading to an accident
   - reducing exposure
   - isolation and containment
   - protection systems and fail-safe design
4. Damage reduction
   - Accidents: often outside the system boundary
   - warnings, emergency actions

Hazard elimination I: Substitution

Substitution of materials, equipments:
new risks may arise, but they should be minimal

- chemical processes: flammable heat transfer to water hydraulic instead of pneumatic (avoid rupture and shock wave)
- missile propulsion: hybrid systems instead of gas
- gas cooled reactors (cooled also by convection if the cooling fails)
- simple mechanical locks instead of computer systems (e.g. automatically open the circuit if the door is open)

Hazard elimination II: Simplification

Minimizing the number of parts, modes, interfaces

- $\rightarrow$ fewer opportunities to fail
- e.g. chemical industry: fewer leakage points
- accidents $\leftarrow$ tight coupling, interactive complexity
- simple interfaces $\rightarrow$ testability

Software: easy to use complex interfaces and systems

- $\rightarrow$ special care has to be taken
- simple control structures needed
  (Honeywell autopilot: no interrupts, procedures and back branches;
  one loop which is executed at fixed rate
  factors to be determined at design time)
- avoiding nondeterminism is crucial
  - time periodicity in RT systems
  - predict algorithm behavior
  - test software (avoid ”transient” faults)
  - operator: rely on consistency
  - $\rightarrow$ static scheduling (polling)
  - $\rightarrow$ exclusive modes
  - $\rightarrow$ state transition depends only on the current state

Simplification (continued)

Software simplification (continued):

- requirements:
  - testability (deterministic, no interrupts, single tasking)
  - readability (sequence of events processed)
  - interactions limited
  - worst-case timing done by code analysis
  - minimum features
- avoiding the effect of hardware failures
  - state encoding: redundant
  - message encoding: only the necessary functions ("0 missiles" $\neq$ "I am alive")
- reducing the unknown events caused by unproven technology:
  - space: "flight-proven" hardware
  - new design only if requirements are not met by old ones
- problems:
  - adding hazard control $\leftrightarrow$ system simplicity
  - flexibility $\leftrightarrow$ leakage points
  - reliability (redundancy) $\leftrightarrow$ complexity increase

Hazard elimination III: Decoupling

Efficient but often not safe:

- failure modes:
  - tightly coupled system: interdependent
  - failure → rapidly affect others
  - hard to isolate erroneous parts
- hazards: unplanned interactions → domino effect
- examples of decoupling:
  - firebreaks
  - over/underpasses
- computers: increase coupling
  - control multiple systems (coupling agent)
- software:
  - modularization: crucial how to split up safety critical functions into a module
  - information hiding: non-critical system does not affect critical one
  - safety kernel: enough to ensure safety on a firewall: (virtual) computer for safety-related functions

Hazard elimination IV:
Elimination of specific human errors

Reduce the opportunities for errors:

- incorrect assembly is impossible (interfaces, connectors)
- color coding

Clear status indications:
$\rightarrow$ next chapter

Software: the question of programming language

- pointers,
- complex control structures,
- implicit/default actions
- overloading

Hazard elimination V: Reduction of hazardous materials or conditions

Reduction:

- well-known in chemical industry
- software: no unused code $\leftrightarrow$ COTS

Change of conditions:

- lower temperature,
- lower pressure
- etc.

Hazard reduction: Safeguards

Passive safeguards:

- maintain safety by their presence (shields, barriers)
- fail into safe states (e.g. weight-operated sensors, relays which are open)

Active safeguards:
require some actions to provide protection (control systems)

- monitoring (detecting a condition)
- measuring state variables
- diagnosis

Hazard reduction I: Design for controllability

Make the system easier to control:

- incremental control: critical actions not in a single step
  - feedback from the plant
  - corrective actions
- intermediate states: not only run/shutdown
  - multiple levels of functionality
  - ”emergency mode”: only critical functions
- decision aids: assist in controlling the plant
  - alarm analysis: e.g. in nuclear plant
  - disturbance measures: measured data → cause-consequence analysis → correction
  - action sequencing: e.g. valve sequences
- monitoring: detecting a problem
  - checking conditions of potential problem
  - validating assumptions used during the design
  - Detecting:
    - condition exists
    - device is ready/busy
    - input/output is satisfactory
    - limits are exceeded

Design for controllability (continued)

- ideal monitors:
  - detect problem fast, at low level ($\rightarrow$ time for correction)
  - independent (limited: info + system assumptions)
  - as little complexity as possible
  - easy to maintain, check, calibrate
  - self-checking
- monitoring computer systems:
  - Levels of checking:
    - hardware level checks: memory access, control flow, signals, checksums, coding
    - code level: assertions
    - audit level: data consistency, independent monitoring
    - system level: supervisory checks
  - Checks are better at lower levels:
    - less delay $\rightarrow$ no erroneous side-effects
    - ability to isolate/diagnose
    - ability to fix (rather than backward recovery)
  - Structure: without additional risk
    - safety kernel

Hazard reduction II: Barriers

Lockout: make access to a dangerous process/state difficult
Lockin: make difficult to leave a safe state
Interlock: enforce a sequence of events/actions

Lockout: prevents a dangerous event or to enter dangerous state

- physical barriers:
  - avoid electromagnetic interference
  - (aircraft radio system, electromagnetic particles)
- authority limiting:
  - prevent dangerous actions (e.g. correcting user inputs in autopilots)
  - $\rightarrow$ do not prohibit necessary actions!
- software: access to safety-critical code/variables
  - security techniques
    access rights (for users)
    access control list (for resources)
    capabilities (ticket to enter)
  - reference monitor: controlling all access
  - multiple confirmations
  - restricted communication
  - security kernel (low-level)

Barriers (continued)

Lockin: maintain a condition

- keep humans in an enclosure (seat belts, doors)
- contain harmful/potentially harmful products
- maintain controlled environment (space suits)
- constrain a particular event (safety valves)
- software: tolerate erroneous inputs

Interlock: enforcing correct sequence of events

- inhibit: event does not occur inadvertently (sequence check)
- inhibit: event does not occur if condition C (dead-man switch)
- sequencer: event A occur before event B (traffic signals)
- interlock fails $\rightarrow$ function should safely stop
- danger: maintenance removal of interlocks
- software: often the hardware interlocks have to be kept;
  - software only monitors interlocks;
  - keeps safe sequences
- software mechanisms:

- programming. language synchronization features: error prone (hardware, software)
- baton: passed to a function; checked before execution: prerequisite tasks have to modify it
- come-from check: process receives data from valid source

Barriers (continued)

Example: Nuclear detonation system

- isolation: separating critical elements
- incompatibility: unique signals
  - signal pattern to start
  - different channels (energy, information)
- inoperability: keeping in inoperable state (without ignition)

Hazard reduction III: Failure minimization

Reducing failure rate $\rightarrow$ reducing risk:

- safety margins
- redundancy
- error recovery

Safety margins:

- in a design many uncertainties: failure rates, conditions
- safety factors: designing a component to withstand higher stress
  nominal (expected) strength / nominal stress > 1

- problem: probability density functions (may overlap)
  probability(stress) functions
  $\rightarrow$ safety margin has to be defined

Failure minimization (continued)

Redundancy:

- many forms: replica, design diversity
- often conflict between safety and reliability
  - e.g. redundancy: more power consumption
  - increased complexity $\rightarrow$ new faults (redundancy management)
  - effective against random failures
- well-designed redundancy is required
  - no common mode failures
  - reduced dependencies (also during test and maintenance)
  - specification has to be elaborated more precisely
- reasonableness checks: difficult to write

Recovery:

- forward and backward recovery have to be used together (time + environment state)
- avoiding domino effect: complex algorithms which are error prone
- forward recovery is proposed, if the error can be identified and fixed

Hazard control

Limiting exposure:

- normal (default) state is safe
- starting in a safe state
- error $\rightarrow$ automatic shutdown to safe state
- trigger is required to go to unsafe state

Isolation:

- barriers and shields
- plants located in isolated area (no population)
- transport of dangerous material

Hazard control (continued)

Protection systems:

- detectors (gas, fire, water etc.) $\rightarrow$ moving to safe state
- panic button (training is required)
- watchdog timers: separate power etc.
- passive devices are safer
- protection system: should signal that it works
  it can also cause damage (emergency destruct)
- fallback states:
  - partial shutdown
  - hold (no new function, maintain safe state)
  - emergency shutdown
    normal: cut power form all circuits
    production: after the current task is completed
    protection: keep only necessary functions
  - restart
- subsystems:
  - sensor to detect hazardous condition
  - challenge subsystem to test the sensor
  - monitor to watch the interruption of the challenge-response sequence

Damage reduction

- emergency procedures: prepared, trained, practiced
- point of no return: turn to emergency actions instead of continue to save the system
- warning: too frequent → insensitive people
- techniques: escape route + limiting damage

Part V

Software safety analysis

Basics

Accidents in which software involved: due to requirement flaws

- incompleteness
- wrong assumptions
- unhandled conditions
- (coding errors affect reliability, not safety;
  + unintended functions)

→ General criteria required: checklist for requirement completeness and safety

- top-down analysis is possible
- bottom-up analysis is not practical (too much states)

Components in requirements:

1. Basic function or objective, safety criteria included
2. Constraints on operating conditions
   limit the set of possible designs
   e.g. physical constraints, performance, process characteristics
3. Prioritized quality goals (to help design decisions)

Basics (continued)

Completeness: the most important property of specifications

- distinguish from any undesired behavior
- ”lack of ambiguity”
- ambiguous: subject to more than one implementation

Software model:

- controller + sensors + actuators + plant
- state machine model (describing behavior, black box)
- model of the plant in the software:
  - must be synchronous with real plant
  - must completely describe the real plant
  - complete trigger specification is required

Human-computer interface criteria

- alert queue:
  - events,
  - ordering (time or priority),
  - notification mechanism,
  - review and disposal,
  - deletion
- transactions: multiple events/actions in one
- displaying data:
  - cause events identified
  - refreshing: time, new events, operator required
  - disappearing

State completeness

- the system and software must start in a safe state
  - interlocks initialized
- internal model of the plant must be updated after startup
  - (plant changes when the software not running)
  - (manual actions have to be taken into account)
- system and local variables (including. clocks) must be initialized upon startup
  - (complete startup or after off-line phase)
  - (detecting loss of information: message numbers, timestamps)
- to be specified: handling inputs before startup / after shutdown
  - (some hardware can retain inputs)
- the maximum time the computer waits for the first input is specified
  - no input $\rightarrow$ alarm for operator;
  - the internal model of the plant cannot be synchronized

State completeness (continued)

- paths from fail-safe states must be specified, the time
  - spent in reduced-function state must be minimized
  - (non-normal processing modes are limited)
- there must be a response for inputs in any state
  - indeterminate states are included
  - (also for ”unexpected” inputs)
  - (unexpected input indicates a malfunction)
  - examples:
    - aborting twice,
    - opening something twice,
    - etc.

Input or output variable completeness

- regarding sensors and actuators
- all information from the sensors must be used in the specification
  - unused input $\rightarrow$ omission in specification;
    what to do with it?
- legal output values which are never produced should be checked
  - e.g. spec. only opens a valve, without closing it

Trigger event completeness

- robust system: correct answer to unexpected inputs
- unexpected inputs/behavior checked by environment constraints
- logging unexpected inputs is important
- events that trigger state changes must satisfy:
  - every state has a transition for every possible input
  - all conditions (input patterns) have to be taken into account
  - every state has a defined time-out if no input occurs
- behavior of the state machine must be deterministic
  - (one transition for each input pattern; disjoint conditions)
  - (predictable behavior is required)
- all incoming values should be checked;
  - response specified for out-of-range values
  - (indicator of malfunctions / out of synchrony)
- all inputs must be bounded in time;
  - behavior specified if the limits are violated / unexpected inputs arrive
  - (”exactly at” is not a good specification style)

Trigger event completeness (continued)

- a trigger involving the non-existence of an input must be bounded in time
  - (given by clocks or using other events)
- minimum and maximum load assumptions must be specified for interrupts
  - whose arrival rate is not limited
- minimum-arrival rate checks should be included
  - (the software must query the empty communication channels)
- response to overload conditions must be specified
  - alarm
  - trying to reduce load (controlling the plant)
  - lock out interrupts (masking)
  - reduced accuracy output generation
  - reduced functionality (process selected interrupts only)
- performance degradation should be graceful, operators must be informed
  - (predictably and not abrupt degradation)
- if reconfiguration is used, hysteresis delay must be included
  - (to avoid ping-pong)

Output specification completeness

Safety-critical outputs are checked for reasonableness.

Capacity

- the absorption rate of the output environment must be higher than the input/computing rate
  - (to avoid output saturation)
- action should be specified if the output rate is exceeded
- human operators should not be overloaded
  - (actions and responses should not be mixed)
- automatic update and deletion of human interface must be specified
  - (events negated or updated by other events, becoming irrelevant)
- specify what to do when the event is displayed and when removed
  - (e.g. removing events only after operator commit)

Data age

- all inputs used by output events must be limited in the time they can be used
  - (data age; validity time of messages)
- incomplete transaction should be canceled after a time-out
  - (operator should be informed)
  - (incomplete transaction: higher risk case)
- revocation (undo) of actions require:
  - specification of conditions and times when it could be done
  - operator warnings

Latency

- latency factor is specified if the output is triggered by an interval of time without a specified input
- action to be specified: what to do if an input arrives late, while the ”late output” is generated
- latency factor: data display for operator changes just prior to a new command from the operator
  - (ask the operator: the change was noted or not)
  - (the operator has opportunity to observe the change)
- hysteresis must be specified for human interface data,
  - (to allow time for interpretation)
  - specified: what to do if data changes in hysteresis period

Output to trigger event relationships

- basic feedback loops has to be involved with checks on the inputs
  - (to detect the effect of any output of the software)
  - (not only limits, but also trends are important)
  - (expected behavior of the plant is checked)
- for every output detected by an input there must be specification
- for normal response
- for abnormal (missing, late, early etc.) response
- too early inputs must be detected and responded as abnormal
  - (considering output latency)
- stability requirements must be specified when the plant
  - seems to be unstable

Specification of transitions between states

- all specified states must be reachable
  - (otherwise no function or missing state transition)
- states should not inhibit the production of later required outputs
  - (otherwise reachability problems may inhibit the output)
- output commands should be reversible
  - (cancel or reverse some actuator commands)
- states reversing the commands should be reachable
  - (reachability analysis)
- preemption requirements should be specified
  - normal processing in parallel
  - refusing the new action
  - preemption of the partially completed transaction
- soft and hard failure modes should be eliminated from all hazardous outputs
  - soft failure mode: an input is required to go from a
    - ∗ given state with A to all others with B;
    - ∗ missing of this input is a soft failure mode
  - hard failure mode: an input is required to go from all
    - ∗ states with A to all others with B;
    - ∗ missing of this input is a hard failure mode

Specification of transitions between states (continued)

- multiple paths should be provided for state changes that maintain or enhance safety
  - (a single failure should not prevent taking actions)
- multiple inputs should be required for paths from safe to
  - hazardous state

Constraint analysis

- transitions must satisfy software safety requirements
  - failing to perform a required function
  - unintended function, wrong answer
  - function at the wrong time, wrong order
  - failing to recognize a hazardous condition (no correction)
  - producing wrong response to hazardous condition
- reachable hazardous states should be eliminated,
  - or at least reduced in time and frequency
- general safety policy:
  - no paths to catastrophic states
  - always path(s) from hazardous to safe state
  - paths from hazardous state to minimum risk state

Checking the specification

- automated reachability analysis
- constrained specification language
  - (e.g. time bounds of inputs have to be specified)