

Software Reliability and Safety

CSE 8317 — Fall 2005

Prof. Jeff Tian, tian@engr.smu.edu
CSE, SMU, Dallas, TX 75275
(214) 768-2861; Fax: (214) 768-3085
www.engr.smu.edu/~tian/class/8317.05f

SSE.1: SSE Basics and SSP

- Motivation and Concepts
- Defining Embedded Systems
- Software Safety Program (SSP)

Software Safety Engineering

- SSE.1: SSE basics and SSP
 - ▷ SSE basics: “Safeware” Parts I-III
 - ▷ SSP (software safety program)
 - “Safeware”, Part IV (Ch.11-18)

- SSE.2: Hazard analysis and resolution
 - ▷ Focus: accidents and pre-conditions (hazards), not other failures
 - ▷ “Safeware” Ch.13-16 & SQE Ch. 16.4
 - ▷ Identification and analysis
 - ▷ Resolution: elimination/reduction/control

- Formal verification related:
 - ▷ Main part: SSE.3, SQE Ch. 15.
 - ▷ PSC: SSE.4, SQE Ch. 16.5

Safety: Why?

- Risk in modern society:
 - ▷ Serious accidents:
 - “Safeware” Appendix A-D
 - medical/aerospace/chemical/nuclear/etc.
 - ▷ techniques for reducing risks
 - ▷ cost-benefit analysis

- Risk factors in industrialized society:
 - ▷ new technology \Rightarrow new hazard
 - ▷ increasing complexity
 - ▷ interdependency \Rightarrow exposure \uparrow
 - ▷ increasing amount of energy
 - ▷ automation \uparrow of manual operations
 - ▷ increasing centralization and scale
 - ▷ increasing pace of tech. change

Computers and Risk

- Computer in safety-critical systems
 - ▷ controller/control subsystem:
 - application-specific computer
 - general-purpose computer
 - ▷ functionality and flexibility
 - ▷ fact of life
 - ▷ critical functions (later)

- Software safety: difficulties
 - ▷ continuous vs. discrete states
 - ▷ the “curse of flexibility”
 - “Safeware” Fig.2.4 (p.35)
 - ▷ complexity and invisible interface
 - ▷ lack of historical usage information
 - ▷ pure SE approach inadequate ⇒ SSE

SSE: Pure SE?

- Pure SE (S/w Eng.) approach
 - ▷ Safety constraints \Rightarrow requirements
 - ▷ Carried/verified in development stages
 - ▷ Fig. 18.1 (a)
 - ▷ Basis: myths below.

- Software myths (“Safeware” Ch. 2.2):
 - ▷ lower cost than other devices
 - ▷ software is easy to change
 - ▷ computers provide greater reliability
 - ▷ software reliability $\uparrow \Rightarrow$ safety \uparrow
 - ▷ testing/formal-veri. eliminate defects
 - ▷ reusing software \Rightarrow safety \uparrow
 - ▷ computers reduce risk over mechanical systems

SSE: Problems and Solutions

- Assumptions and problems
 - ▷ Level of quality (LoQ) required
 - ▷ LoQ provided by existing practice (SRE?)
 - ▷ Fault-tolerant techniques
 - particularly NVP, intrinsic problems
 - LoQ still not enough
 - ▷ Formal verification
 - LoQ and scalability problem

 - Problems and solutions:
 - ▷ Scalability and coverage
 - ▷ Correctness of *everything*
 - ▷ Not focus on safety-related artifacts
- ⇒ SSE, particularly Leveson's SSP

Basic Definitions

- Accident or mishap:
 - ▷ unplanned (series of) events
 - ▷ leading to unacceptable loss
 - death, injury, illness
 - equip./property/environment damage
 - ▷ excess energy/dangerous substance
 - ▷ computers relatively safe
 - ▷ but computer control \Rightarrow accidents

- Hazard:
 - ▷ a set of conditions leading to accidents under certain environmental conditions
 - ▷ e.g.: guard gates at rail-crossing
 - ▷ safety focus: control factors (vs. env. factors beyond control)
 - ▷ analysis and resolution \Rightarrow SSE

Basic Definitions

- Risk: function of 3 elements
 - ▷ likelihood(hazard)
 - ▷ likelihood(hazard \Rightarrow accident)
 - ▷ worst possible loss due to accident
(compare to expected loss)

- (System) safety engineering:
 - ▷ ensuring acceptable risk
 - ▷ scientific/management/engineering
 - ▷ reducing risk factors
 - ▷ context for software safety
 - ▷ hazard identification, assessment,
analysis, and resolution

Safety and Embedded Systems

- *Safety*: The property of being accident-free for (embedded) software systems.
 - ▷ Accident: failures with severe consequences
 - ▷ Hazard: condition for accident
 - ▷ Special case of reliability
 - ▷ Specialized techniques
 - ▷ Focus on prevention and tolerance

- Embedded systems
 - ▷ Failure and consequences
 - ▷ Interaction among sub-systems
 - ▷ Safety: software vs. system

System/Software Definitions

- System (general vs embedded):
 - ▷ Physical systems or processes
 - ▷ A set of components
 - ▷ Common purposes/objectives
 - ▷ Description: input/output/time
 - ▷ Self-regulating vs. controlled

- Controller/Control subsystem:
 - ▷ Providing control to system
 - order events
 - regulate variable values
 - ▷ Help achieve overall objectives
 - ▷ Example control systems
 - ▷ Use of computers in control

System Definitions: Control Function

- Function (mathematical?) to be achieved
 - ▷ input, output and time
 - ▷ dynamic (differential) equation(s)
 - ▷ state variables and matrices
 - ▷ traditional vs. modern analysis
 - ▷ use of computers and implications

- Traditional analyses
 - ▷ single input, single output
 - ▷ transformations: Fourier & Laplace
 - ▷ stability criteria
 - ▷ performance and other analysis
 - ▷ pre-requisite for safety

System Definitions: Control Function

- Modern control system analyses
 - ▷ state variables and set of equations
 - ▷ controllability & observability
 - ▷ other concerns:
 - optimality, robustness, adaptability, etc.
 - ▷ computer controller
 - ▷ continuous vs. discrete system
 - ▷ Z-transformation

- Example control systems
 - ▷ traditional feedback control
 - ▷ state variable based
 - ▷ sampling and discrete systems
 - ▷ computer control (examples later)

Analysis and Constraints

- Previous analyses unconstrained (provide necessary but not sufficient condition for safety)

- Constraints on operating conditions
 - ▷ quality considerations
 - effect of defects in system
 - performance and other measures
 - ▷ equipment capacity
 - time and/or energy constraints
 - volume, rate, etc.
 - ▷ process characteristics
 - above factors fit into process
 - given vs. adjustable aspects
 - ▷ safety constraints (next)
(derived from analysis of above)

System Definitions: Safety Constraints

- Safety constraints:
 - ▷ Derived from safety process
 - particularly hazard id. FTA & ETA
 - ▷ Example: pressure threshold
 - ▷ Integration to other functions?
 - ▷ Discrete vs. continuous functions

- Handling of safety constraints:
 - ▷ Constrained optimization
 - feasibility and practicality problems
 - ▷ Usually handled separately:
 - different/conflicting concerns
 - different characteristics
 - feasibility of functional representation?
 - liability and regulatory concern

System Definitions: Software Safety

- Software functions in control systems:
 - ▷ data logging
 - ▷ control function implementation
 - direct digital control (via actuators)
 - supervisory control (values/parameters)
 - ▷ maintenance of safety conditions
 - ▷ example: nuclear plant

- Relating safety constraints to software:
 - ▷ data logging: no direct impact
 - ▷ other two: possible safety problems
 - ▷ subsequence analysis

Software Safety Program (SSP)

- Leveson's approach
 - ▷ Limited goals
 - ▷ Safety analysis and hazard resolution
 - ▷ Safety verification: Fig. 18.1 (c)
 - few things carried over (dotted line)
 - ▷ Part IV, "Safeware"
 - particularly Chapters 15-18.

- Software safety program (SSP)
 - ▷ Formal verification/inspection based
 - ▷ But restricted to safety risks
 - ▷ Based on hazard analyses results

SSP in Software Lifecycle

- Major activities
 - ▷ Hazard identification and analysis
 - ▷ Hazard resolution (design for safety)
 - ▷ Safety verification
 - ▷ Change analysis and operational feedback
 - ▷ Fit in s/w process; Fig. 13.2 (p.293)

- Safety constraints and verification
 - ▷ Identify problems early
 - ▷ Carry over as design/code constraints
 - ▷ Distributed verification effort
 - ▷ Cascading:
 - using safety/design/code constraints
 - represented as formal specs
 - verifying req./HLD/LLD/code

SSP in Software Lifecycle

- SSP in early (concept formation) phase:
 - ▷ Initial risk assessment: identify
 - critical areas/hazards/design criteria
 - ▷ Preliminary hazard list
 - ▷ Audit trail: tracking/evaluating
 - ▷ Hazard analysis of previous accidents

- SSP in requirement stage
 - ▷ SRS (s/w req. specifications)
 - ▷ SRS consistent/satisfy safety constraints
 - ▷ Conflicts and tradeoffs?
 - ▷ SRS in a formal language
 - able to handle timing and failure

SSP in Software Lifecycle

- SSP in High-Level Design (HLD)
 - ▷ Identify safety-critical items
 - based on FTA and ETA
 - ▷ Design for safety: key!
 - isolation/encapsulation
 - protection and security, etc.
 - ▷ Use of safety invariants for modules
 - ▷ Link to pre/post safety verifications

- SSP in Low-Level Design (LLD)
 - ▷ Safety invariants/etc. preserved
 - ▷ (dynamic) interconnection properties
 - ▷ Same design for safety issues
 - but finer granularity/less flexibility

SSP in Software Lifecycle

- SSP in code analysis
 - ▷ Further refinement
 - ▷ Preserving safety invariants/properties?
 - ▷ Combination of techniques
 - testing/inspection/formal veri., etc.
 - safety-focus: based on FTA&ETA
 - ▷ Yih/Tian approach later

- SSP in configuration control/maintenance
 - ▷ Change during verification/operation
 - ▷ Change effect analysis:
 - how does it affect safety
 - problem identification and resolution
 - use FTA and ETA with modifications
 - ▷ Importance of separation/isolation
 - ▷ Above \Rightarrow informed safety management

Perspectives

- State-of-the-Practice:
 - ▷ Computer used in safety-critical appl.
 - ▷ S/w Eng.: V&V, SRE, FT, FM
 - ▷ Gap between safety goal and QA

- SSE: Augment S/w Eng.
 - ▷ Overall framework: Leveson's SSP
 - ▷ Analysis to identify hazard
 - ▷ Design for safety/hazard resolution
 - ▷ Safety constraints and verification

- Link to other topics:
 - ▷ In addition to: V&V, SRE
 - ▷ Important elements: FM and FT
 - ▷ New development: prescriptive specs