
Tutorial

# F3

## Engineering Safety and Security Related Requirements for Software Intensive Systems

*Donald G. Firesmith*

**Day:** Monday 21 May 2007, Full Day Tutorial

**Venue:** Lasalle Room

**Engineering Safety- and Security-Related Requirements for Software-Intensive Systems**

ICSE'2007 Conference Tutorial

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Donald Firesmith
21 May 2007

Software Engineering Institute | Carnegie Mellon 

## Tutorial Goals

Familiarize Members of:

- *Safety* and Security Teams with:
  - Foundations of *Requirements Engineering*
  - *Common Concepts and Techniques from Both Disciplines*
- *Requirements* Teams with the Foundations of:
  - *Safety Engineering*
  - *Security Engineering*

Familiarize Members of all three Disciplines with:

- Different Types of *Safety- and Security-related Requirements*
- Common *Process* for Engineering these Requirements



## Contents





## Three Disciplines:
*Requirements, Safety, and Security Engineering*

## You Are Here

**Three Disciplines ◄**

Challenges

Common Example

Requirements Engineering Overview

Safety and Security Engineering Overview

Types of Safety- and Security-related Requirements

Common Consistent Collaborative Method

Conclusion



## Safety and Security Engineering

Safety Engineering

> the engineering discipline within systems engineering concerned with lowering the risk of *unintentional unauthorized* harm to valuable assets to a level that is acceptable to the system's stakeholders by preventing, detecting, and reacting to such harm, mishaps (i.e., accidents and incidents), hazards, and safety risks

Security Engineering

> the engineering discipline within systems engineering concerned with lowering the risk of *intentional unauthorized* harm to valuable assets to a level that is acceptable to the system's stakeholders by preventing, detecting, and reacting to such harm, misuses (i.e., attacks and incidents), threats, and security risks

Differences:

- Unintentional vs. Intentional
- Accidental vs. Malicious Harm
- Mishaps vs. Misuses
- Hazards vs. Threats



## Requirements Engineering

Requirements Engineering

> the engineering discipline within systems/software engineering concerned with identifying, analyzing, reusing, specifying, managing, verifying, and validating goals and requirements (including safety- and security-related requirements)



## Challenges:

*Combining Requirements, Safety, and Security Engineering*

## You Are Here

Three Disciplines

**Challenges ◄**

Common Example

Requirements Engineering Overview

Safety and Security Engineering Overview

Types of Safety- and Security-related Requirements

Common Consistent Collaborative Method

Conclusion



## Challenges$_1$

Requirements Engineering, Safety Engineering, and Security Engineering:

- Different *Communities*
- Different *Disciplines* with different Training, Books, Journals, and Conferences
- Different *Professions* with different Job Titles
- Different fundamental underlying *Concepts* and Terminologies
- Different *Tasks, Techniques,* and *Tools*

Safety and Security Engineering are:

- Typically treated as secondary Specialty Engineering Disciplines
- Performed separately from, largely independently of, and lagging behind the primary Engineering Workflow
  (Requirements, Architecture, Design, Implementation, Integration, Testing)



## Challenges$_2$

Current separate Processes for Requirements, Safety, and Security are Inefficient and Ineffective.

Separation of Requirements Engineering, Safety Engineering, and Security Engineering:

- Causes *poor* Safety- and Security-related Requirements.
  - Goals rather than Requirements
  - Vague, unverifiable, unfeasible, architectural and design constraints
- Inadequate and too late to drive architecture and testing
- Difficult to achieve Certification and Accreditation



## Challenges$_3$

Poor requirements are a primary cause of more than half of all project failures (defined in terms of):

- Major Cost Overruns
- Major Schedule Overruns
- Major Functionality not delivered
- Cancelled Projects
- Delivered Systems that are never used

Poor Requirements are a major Root Cause of many (or most) Accidents involving Software-Intensive Systems.

Security 'Requirements' often mandated:

- Industry Best Practices
- Security Functions or Subfunctions

## Challenges$_4$

How Safe and Secure is Safe and Secure *enough*?

Situation Cries out for Process Improvement:

- Better Consistency between Safety and Security Engineering
  - More consistent Concepts and Terminology
  - Reuse of Techniques across Disciplines
  - Less Unnecessary Overlap and Avoidance of Redundant Work
- Better Collaboration:
  - Between Safety and Security Engineering
  - With Requirements Engineering
- Better Safety- and Security-related Requirements



## Common Example:

*An Automated People Mover System*



## You Are Here

Three Disciplines

Challenges

**Common Example ◄**

Requirements Engineering Overview

Safety and Security Engineering Overview

Types of Safety- and Security-related Requirements

Common Consistent Collaborative Method

Conclusion



## Desired Characteristics

Common Ongoing Example throughout the Tutorial

Should Not Need Special Domain Knowledge

Example System should be:

- Safety-Critical
- Realistic
- SW-Intensive
- Understandable in terms of:
  - Goals and Requirements
  - Application Domain Technology
  - Safety Hazards and Security Threats
  - Accidents and Attacks

## Example Overview

*Very Large* New Zoo

Zoo Automated Taxi System (ZATS)

Example Zoo Habitat Guideway Layout

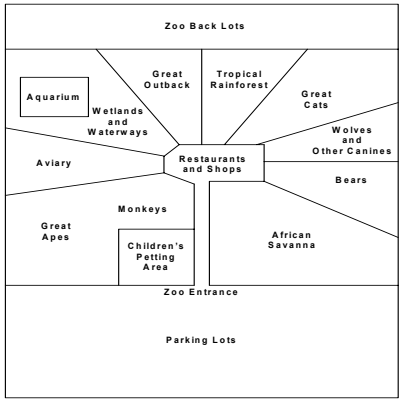
ZATS Context Diagram

Proposed ZATS:

- Taxis
- Elevated Concrete Guideway
- Taxi Stations

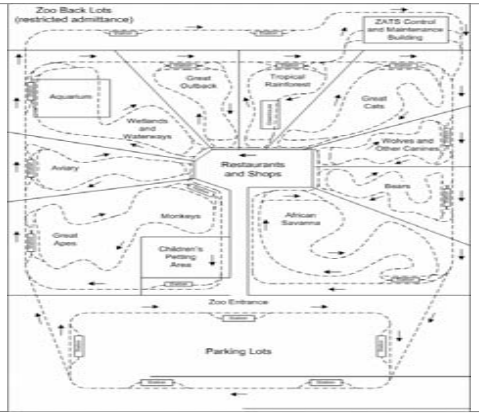


## Very Large New Zoo



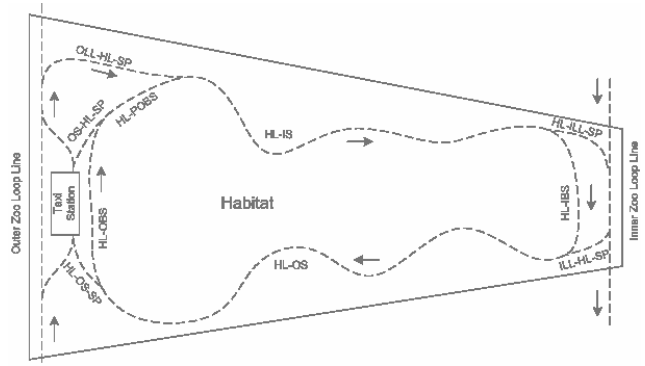



## Zoo Automated Taxi System (ZATS)

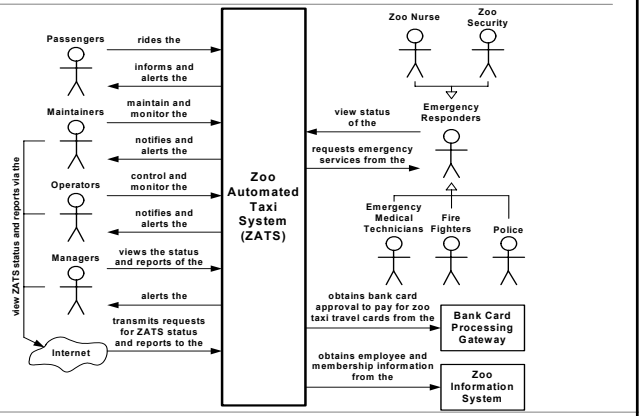





## Example Habitat Layout

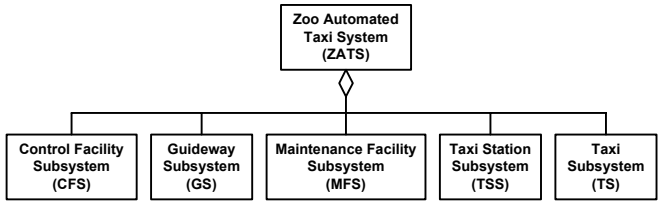


## ZATS Context Diagram



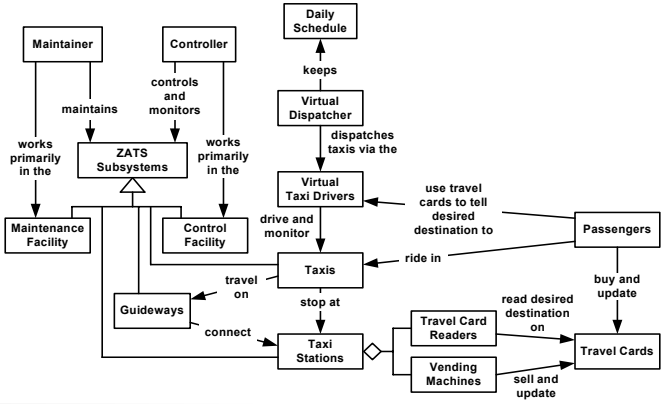



## Proposed ZATS Decomposition



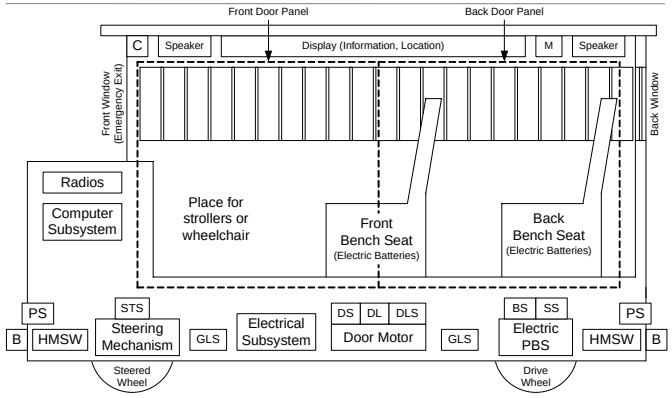



## Proposed ZATS Domain Model

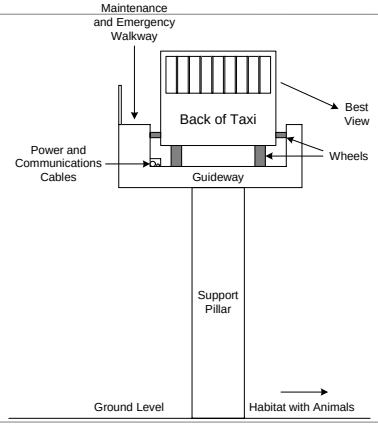





## Proposed Taxi Architecture

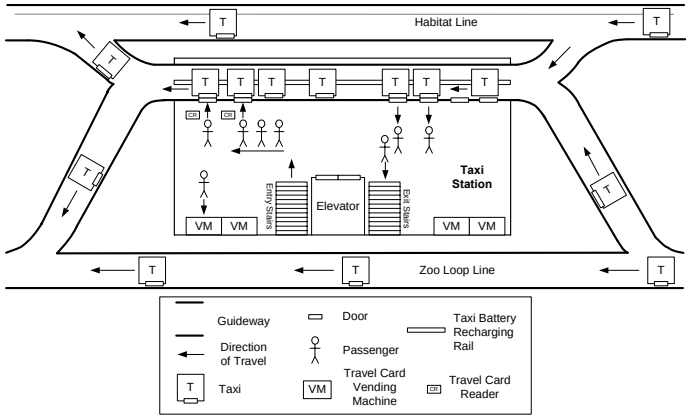


## Automated Taxis On Elevated Guideways



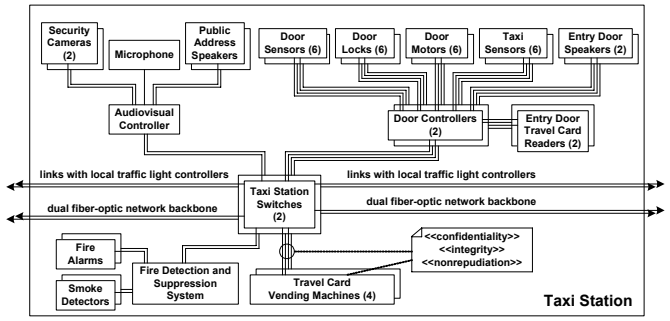

Software Engineering Institute | Carnegie Mellon
Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University
25

## Proposed Taxi Station



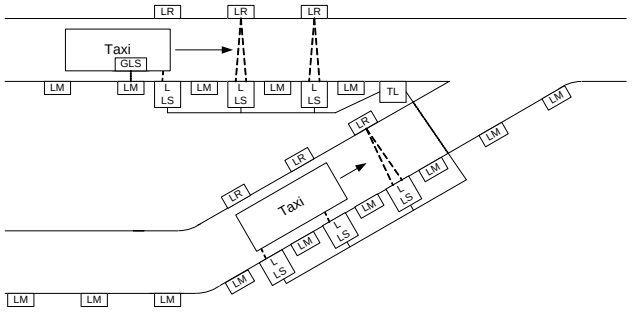

Software Engineering Institute | Carnegie Mellon
Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University
26

## Proposed Taxi Station Network Diagram




Software Engineering Institute | Carnegie Mellon
Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University
27

## Example Collision Hazard




Software Engineering Institute | Carnegie Mellon
Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University
28

## Requirements Engineering:
*An Overview*



## You Are Here

Three Disciplines

Challenges

Common Example

**Requirements Engineering Overview ◄**

Safety and Security Engineering Overview

Types of Safety- and Security-related Requirements

Common Consistent Collaborative Method

Conclusion



## Requirements Engineering Topics

Definition of Requirements Engineering

Requirements Engineering:

- Tasks
- Work Products

Importance and Difficulty of Requirements Engineering

Goals vs. Scenarios vs. Requirements

Types of Requirements

Characteristics of Good Requirements



## Requirements Engineering

Definition

> the engineering discipline within systems/software engineering concerned with identifying, analyzing, reusing, specifying, managing, verifying, and validating goals and requirements (including safety- and security-related requirements)
>
> the cohesive collection of all *tasks* that are primarily performed to produce the *requirements* and *other related requirements work products* for an *endeavor*

Today, these RE tasks are typically performed in an *iterative*, *incremental*, *parallel,* and *time-boxed* manner rather than according to the traditional Waterfall development cycle, whereby:

- Incremental means:
  - Recursively incremental from subsystem to subsubsystem
  - Incrementally scheduled from block/milestone to block/milestone
- Parallel means concurrently with the:
  - Primary work flow disciplines such as architecting, design, and testing
  - Specialty engineering disciplines such as safety and security engineering

## RE Tasks and Work Products

Business Analysis (i.e., Customer, Competitor, Market, Technology, and User Analysis as well as Stakeholder Identification and Profiling)

Visioning

**Requirements Identification (a.k.a., Elicitation)**

Requirements Reuse

Requirements Prototyping

**Requirements Analysis**

**Requirements Specification**

**Requirements Management**

**Requirements Validation**

Scope Management (Management)

Change Control (Configuration Management)

Quality Control (Quality Engineering)



## Requirements Engineering Work Products

Business Analyses

Stakeholder Profiles

**Vision Statement**

- **Goals**

**Operational Concept Document (OCD)**

- **Usage Scenarios**

**Requirements Repository and published Specifications**

- **Requirements**

Requirements Prototypes

Domain Model

Glossary



## Importance and Difficulty of Requirements Eng.

Poor requirements are a primary cause of more than half of all:

- Project failures (defined in terms of):
  - Major cost overruns
  - Major schedule overruns
  - Major functionality not delivered
  - Cancelled projects
  - Delivered systems that are never used
- Hazards and associated Mishaps (Accidents and Safety Incidents)
- Vulnerabilities

The extent of the impact of poor requirements on threats and associated misuses (Attacks and Security Incidents) is much less clear.

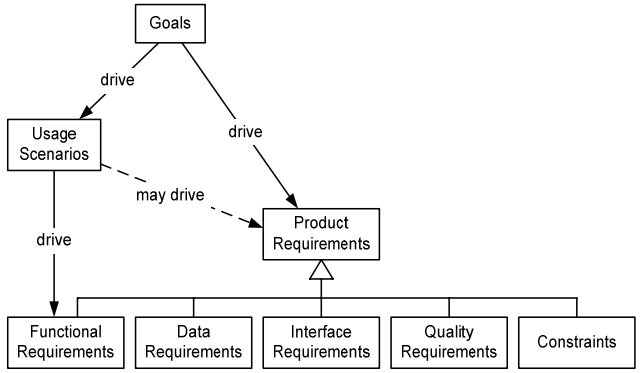


## Difficulty of Requirements Engineering

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

F. Brooks, *No Silver Bullet*, IEEE Computer, 1987

## Goals vs. Usage Scenarios vs. Requirements






## Goals

A **goal** is an *informally documented perceived need* of a *legitimate stakeholder*.

Goals are:

- *Not* requirements.
- Drive the analysis and formal specification of the requirements.
- Typically ambiguous and/or unrealistic (i.e. impossible to guarantee).

A *major* problem is safety and security goals that are specified as if they were unambiguous, 100% feasible, verifiable requirements.

Goals are typically documented in a vision statement.



## Example ZATS Goals

Functional Goals:

- ZATS must rapidly transport patrons between the parking lots and the zoo.
- ZATS must rapidly transport patrons between habitats within the zoo.
- ZATS must allow patrons to take leisurely tours of the habitats.

Data Goal:

- ZATS must record and report appropriate system usage statistics.

Capacity Goal:

- ZATS must include sufficient taxis so that patrons need not wait long for a free taxi.

Usability Goal:

- ZATS must be very easy and intuitive for patrons to use, including those who are not very good with technology.



## Example ZATS Defensibility Goals

Safety Goals:

- "ZATS taxis must be safe."
- "ZATS must not have any serious accidents."
- "ZATS taxis must never collide."
- "ZATS will never kill or injure its passengers or maintainers."

Security Goals:

- "Passenger credit card data must be secure."
- "ZATS taxi service must be protected from denial of service attacks."
- "ZATS computers must prevent infection by malware."
- "ZATS facilities must be protected against arson."
- "ZATS must restrict users to only those tasks for which they are authorized."

## Usage Scenarios

A **usage scenario** is a specific functionally cohesive sequence of interactions between user(s), the system, and potentially other actors that provides value to a stakeholder.

Usage scenarios:

- Are instances of use cases.
- Can be either "sunny day" or "rainy day" scenarios.
- Have preconditions, triggers, and postconditions.
- Are typically documented in an Operational Concept Document (OCD).
- Drive the analysis and formal specification of the [primarily functional] requirements.
- Often include potential design information.
- Can be written in either list or paragraph form.



## Example ZATS Scenario

**Ride Zoo Loop Line To Restaurants for Lunch:**

After the family enters a waiting taxi, Mr. Smith looks at the zoo map on its ceiling. A light representing their taxi is glowing at the Tropical Rainforest Habitat outer taxi station. He uses the control panel to select the inner taxi station at the habitat, which is the central taxi station near the restaurants and shops as a destination. He then swipes his zoo taxi debit card, and the display shows the remaining balance of $9.00 on the card. The taxi warns them to set down and thirty seconds later, the station and taxi exit doors close. Their taxi accelerates out of the taxi station and turns to the left onto the Zoo Loop Line.

Shortly after leaving the taxi station, they see a spur the angles off to their left towards a large building containing the taxi control center and maintenance facility. They continue around the outside of the zoo, passing other the Great Cats, the Wolves and Other Dogs, and the Bears habitats. Just before they reach the outer African Savanna taxi station, the guideway makes a sweeping turn to the right and they can see the parking lot on their left. Everyone looks to see if they can see the family van, but the parking lot is too big and they can only see the parking lot taxi station near where it is parked.

Soon, they pass the zoo entrance on their left and turn right to follow the main street to where the main restaurants and shops are. Their taxi passes the inner African Savanna taxi station on their right, circles around the central area, and soon pulls off the Zoo Loop Line to enter the inner Great Apes and Monkeys taxi station. Exiting the taxi when the doors open, they head down the elevator and outside for an early lunch at one of the many restaurants.



## Requirements

A (product) **requirement** is a *mandatory* characteristic (behavior or attribute) of a product (e.g., system, subsystem, software application, or component).

- Requirements are documented in requirements specifications.
- Requirements are driven by goals.
- Example:
  "At each taxi station while under normal operating conditions, ZATS shall provide a taxi to passengers within an average of 5 minutes of the passengers' request."
- Requirements must have certain characteristics (e.g., verifiable and feasible).



## Characteristics of Good Requirements

| | |
|---|---|
| **Mandatory** | **Complete** |
| **Correct** | Consistent |
| **Cohesive** | Usable by Stakeholders |
| **Feasible** | Uniquely Identified |
| **Relevant** | Traced |
| **Unique** | Externally Observable |
| **Unambiguous** | Stakeholder-Centric |
| **Validatable** | Properly Specified |
| **Verifiable** | Prioritized |
| **What or How Well, not How** | Scheduled |
| | Managed |
| | Controlled |
| | http://www.jot.fm/issues/issue_2003_07/column7 |

## Some Problems due to Poor Requirements

Ambiguous Requirements:

- Developers misinterpret Subject Matter Expert (SME) intentions.
- "The system shall be safe."
- How safe? Safe in what way?

Incomplete Requirements:

- Developers must guess SME intentions.
- "The system shall do X."
- Under what conditions? When in what state? When triggered by what event? How often? How fast? For whom? With what result?




## More Potential Problems

Missing Requirements:

- What shall the system do if it can't do X?
- Unusual combinations of conditions often result in accidents.
- What shall the system do if event X occurs when the system is simultaneously in states Y and Z?

Unnecessary Constraints:

- Inappropriate architecture and design constraints unnecessarily specified as requirements such as:
  - User ID and password for identification and authentication.




## Poor Requirements Cause Accidents$_1$

"The majority of software-related accidents are caused by requirements errors."

"Software-related accidents are usually caused by flawed requirements. Incomplete or wrong assumptions about the operation of the controlled system can cause software related accidents, as can incomplete or wrong assumptions about the required operation of the computer. Frequently, omitted requirements leave unhandled controlled-system states and environmental conditions."

Nancy G. Leveson, 2003

<http://www.safeware-eng.com/index.php/white-papers/accidents>




## Poor Requirements Cause Accidents$_2$

Large percentage of accidents are caused by poor requirements:

- "For the 34 (safety) incidents analyzed, 44% had inadequate specification as their primary cause."

  Health and Safety Executive (HSE), *Out of Control: Why Control Systems Go Wrong and How to Prevent Failure* (2nd Edition), 1995

- "Almost all accidents related to software components in the past 20 years can be traced to flaws in the requirements specifications, such as unhandled cases."

  Safeware Engineering, "Safety-Critical Requirements Specification and Analysis using SpecTRM", *2002*

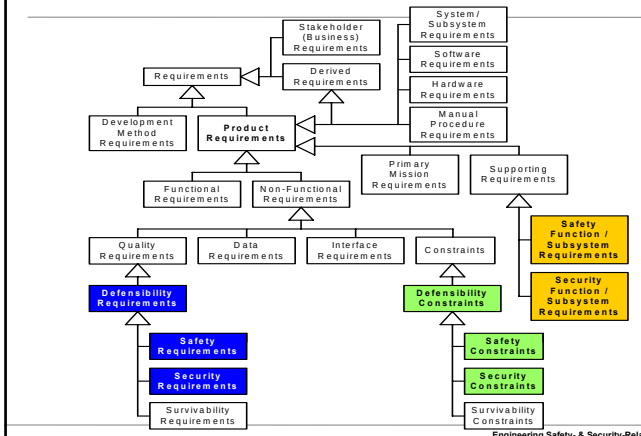
## Poor Requirements Cause Accidents$_3$

Erroneous specification is a major source of defects and subsequent failure of safety-critical systems. Many failures occur in systems using software that is perfect, it is just not the software that is needed because the specification is defective."

John C. McKnight, "Software Challenges in Aviation Systems, 2002



## Types of Requirements






## Product Requirements

A **product requirement** is a requirement for a *product* (e.g., system, subsystem, software application, or component).

- A **functional requirement** is a product requirement than specifies a mandatory *function* (i.e., behavior) of the product.
- A **data requirement** is a product requirement that specifies mandatory [types of] data that must be manipulated by the product.
- An **interface requirement** is a product requirement that specifies a mandatory interface with (or within) the product.
- A **quality requirement** is a product requirement that specifies a mandatory amount of a type of product quality.
- A **constraint** is a property of the product (e.g., design decision) that is ordinarily not a requirement but which is being mandated as if it were a normal requirement



## Representative ZATS Functional Requirement

Prepare for departure warning:

- When a taxi is in the IN SERVICE – STOPPED AT STATION state and its passengers have selected a tour of a habitat and paid for the tour, then (1) ZATS *shall* warn the passengers in the taxi and in the taxi station in front of the taxi (a) to stop boarding that particular taxi because the doors will soon close and (b) to stay away from the doors and (2) ZATS shall transition the taxi to the IN SERVICE – PREPARE FOR DEPARTURE state.

Note precondition, trigger, required behavior, and postcondition.

## Representative ZATS Data Requirement

ZATS shall record the following information about each trip:

- Taxi Identifier
- Taxi Travel Card:
  - Identifier
  - Debit Amount
  - Remaining Balance
- Starting Station
- Destination Station
- Departure Time
- Arrival Time




## Representative ZATS Interface Requirements

ZATS shall interface with the emergency responder systems (e.g., 911) to enable the:

- Operator to request emergency services
- Emergency responders to view ZATS status

ZATS shall interface with the Bank Card Processing Gateway in order to request bank card approval to pay for zoo taxi travel cards.

ZATS shall interface with the Zoo Information System to obtain employee and zoo membership information.




## Representative ZATS Constraints

Architecture Constraints:

- ZATS shall use pre-stressed reinforced concrete guideways able to support 150% max. expected loading.
- ZATS guideways shall be elevated to provide good visibility, to separate patrons from the animals, and to eliminate the possibility of collision between taxis and patrons' vehicles in the parking lots.
- ZATS shall use COTS electric motors.
- ZATS taxis shall use standard automobile tires.
- ZATS shall use a commercial real-time operating system.

Design Constraints:

- ZATS software shall be object-oriented.

Implementation Constraints:

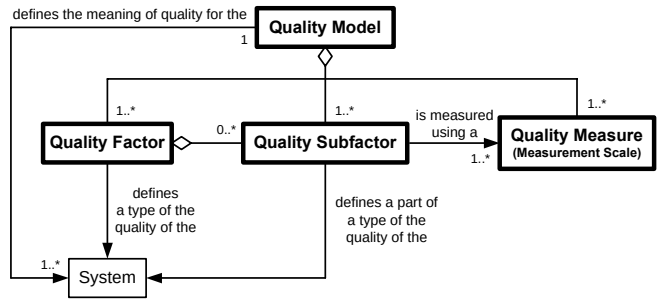
- ZATS software shall be programmed in a safe subset of C++.




## Quality Requirements:

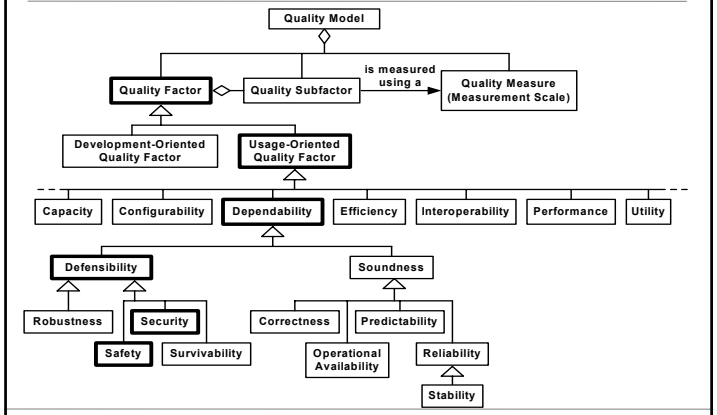
*Specify Minimum Acceptable Quality*

## Quality Model



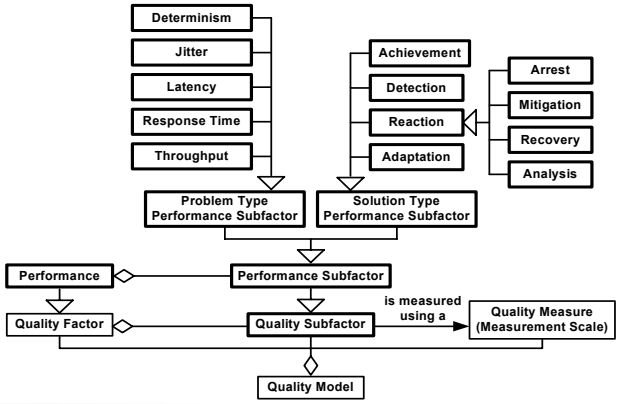



## Quality Factors



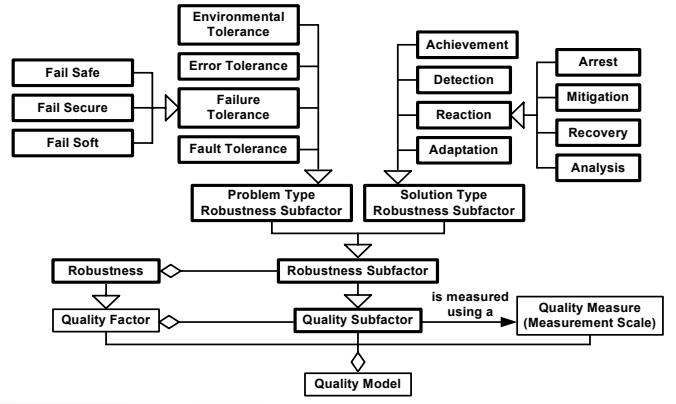



## Performance Subfactors

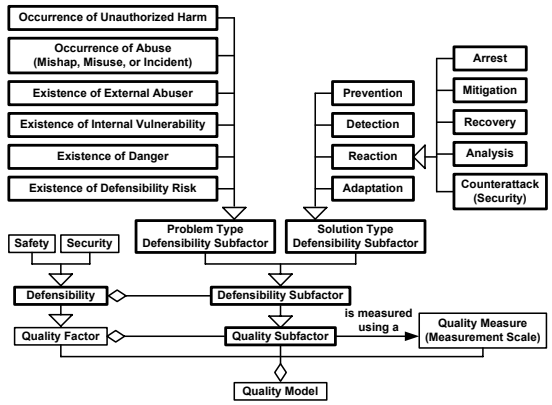





## Robustness Subfactors

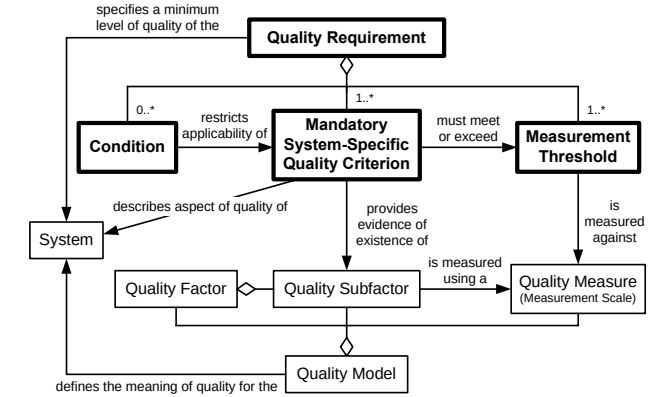


## Defensibility Quality Subfactors






## Components of a Quality Requirement






## Example Quality Requirement

Hazard Prevention Safety Requirement:
"Under normal operating conditions, a taxi shall not move when it's doors are open more than an average of once every 10,000 trips."

Component Parts:

- **Condition:**
  "Under normal operating conditions"
  (e.g., neither during maintenance nor a fire in a taxi station)
- **Mandatory System-Specific Quality Criterion:**
  "a taxi shall not *move* when it's doors are *open*"
  (The meaning of moving and open must be unambiguously defined.)
- **Measurement Threshold:**
  "more than an average of once every 10,000 *trips*."
  (A trip is defined as intentional travel from a station where passengers enter the taxi to the station where the passengers exit the taxi.)



## Importance of Measurement Threshold

Measurement Threshold is:

- Critical
- Difficult (but not impossible) to determine
- Often left out of quality requirements
- Needed to avoid ambiguity

States how much quality is necessary (adequate)

Enables architect to:

- Determine if architecture is adequate
- Make engineering trade-offs between competing quality factors

Enables tester to determine test completion criteria

## Safety and Security Engineering:
*An Overview*



## You Are Here

Three Disciplines

Challenges

Common Example

Requirements Engineering Overview

**Safety and Security Engineering Overview ◄**

Types of Safety- and Security-related Requirements

Common Consistent Collaborative Method

Conclusion



## Similar Definitions

Safety Engineering

> the engineering discipline within systems engineering concerned with lowering the risk of *unintentional unauthorized* harm to valuable assets to a level that is acceptable to the system's stakeholders by preventing, detecting, and reacting to such harm, mishaps (i.e., accidents and incidents), hazards, and safety risks

Security Engineering

> the engineering discipline within systems engineering concerned with lowering the risk of *intentional unauthorized* harm to valuable assets to a level that is acceptable to the system's stakeholders by preventing, detecting, and reacting to such harm, misuses (i.e., attacks and incidents), threats, and security risks



## Fundamental Safety and Security Concepts:
*A Foundation for Understanding*

## Fundamental Safety and Security Concepts

Safety and Security as a Quality Factors with associated Quality Subfactors

Systems responsible for Valuable Assets

Stakeholders

Accidental and Malicious Harm to Valuable Assets

Defensibility Occurrences (Accidents, Attacks, and Incidents)

Abusers (External and Internal, Malicious and Non-malicious)

Vulnerabilities (system-internal sources of dangers)

Dangers (Hazards and Threats)

Defensibility Risks (Safety and Security)

Goals, Policies, and Requirements

Defenses (Safeguards and Counter Measures)




## Safety as a Quality Factor

**Safety** is the Quality Factor capturing the *Degree* to which:

- *Accidental Harm* to Valuable Assets is eliminated or mitigated
- *Mishaps and Events (Accidents, Safety Incidents,* and *Hazardous Events*) are eliminated or their negative consequence mitigated
- *Hazards* (i.e., Hazardous Conditions) are eliminated or mitigated:
  - System Vulnerabilities
  - Non-malicious Abusers (humans, systems, and the environment)
- *Safety Risks* are kept acceptably low
- The preceding Problems are *Prevented*, *Detected*, *Reacted to*, and possibly *Adapted to*



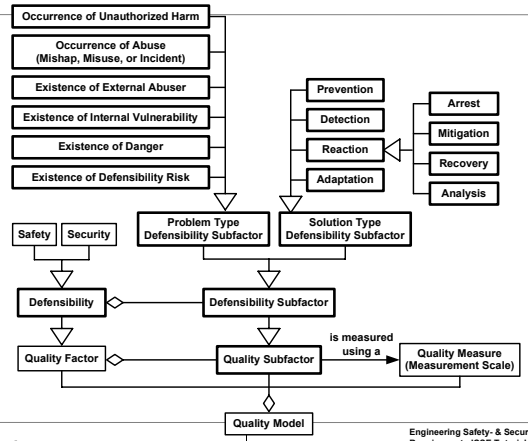

## Security as a Quality Factor

**Security** is the Quality Factor capturing the *Degree* to which:

- *Malicious Harm* to Valuable Assets is eliminated or mitigated
- *Misuses and Events (Attacks, Security Incidents,* and *Threatening Events*) are eliminated or their negative consequence mitigated
- *Threats* (i.e., Threatening Conditions) are eliminated or mitigated:
  - System Vulnerabilities
  - Malicious Abusers (humans, systems, and malware)
- *Security Risks* are kept acceptably low
- The preceding Problems are *Prevented*, *Detected*, *Reacted to*, and possibly *Adapted to*




## Defensibility Quality Subfactors

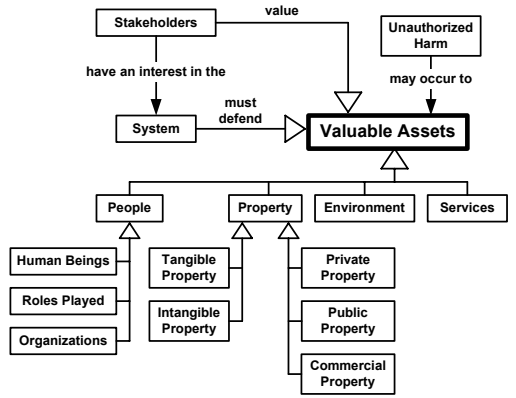


## Different Types of Defensibility Requirements

| | Unauthorized Harm | Abuse | Abuser | Vulnerability | Danger | Defensibility Risk |
|---|---|---|---|---|---|---|
| **Prevention** (current) | Prevent Occurrence of Unauthorized Harm | Prevent Occurrence of Abuse | Prevent Abuser Means or Opportunity | Prevent Existence of Vulnerability | Prevent Existence of Danger | Prevent Existence of Defensibility Risk |
| **Detection** (current) | Detect Occurrence of Unauthorized Harm | Detect Occurrence of Abuse | Detect Existence of Abuser | Detect Existence of Vulnerability | Detect Existence of Danger | Detect Existence of Defensibility Risk |
| **Reaction** (current) | React to Occurrence of Unauthorized Harm | React to Occurrence of Abuse | React to Existence of Abuser | React to Existence of Vulnerability | React to Existence of Danger | React to Existence of Defensibility Risk |
| **Adaptation** (future) | Adapt due to Unauthorized Harm | Adapt to Future Occurrence of Abuse | Adapt to Future Existence of Abusers | Adapt to Future Existence of Vulnerability | Adapt to Future Existence of Danger | Adapt due to Existence of Defensibility Risk |



## Valuable Assets






## Some ZATS Valuable Assets

People:
- Passengers
- Operators
- Maintainers

Property:
- Animals
- Passenger Bank Card Information
- Taxis
- Taxi Stations

Environment:
- Habitat

Services:
- Taxi Service



## Categories of Asset Values

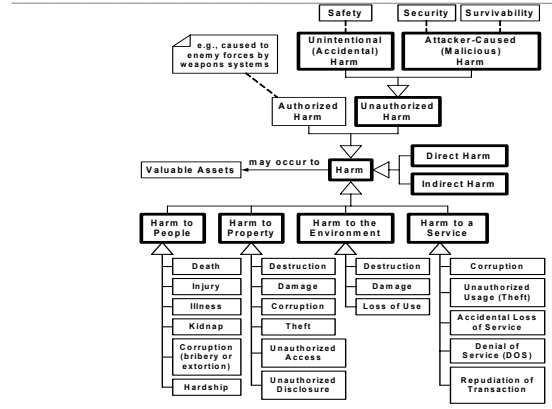
The values of assets are used to determine how to invest limited resources in protecting them from accidental harm.

Sometimes, all values are measured in terms of money in order to be able to compare “apples and oranges.”

More often, the asset values are categorized:
- Extremely valuable (i.e., invaluable or priceless)
- Major
- Moderate
- Minor
- Negligible (i.e., not worth considering)

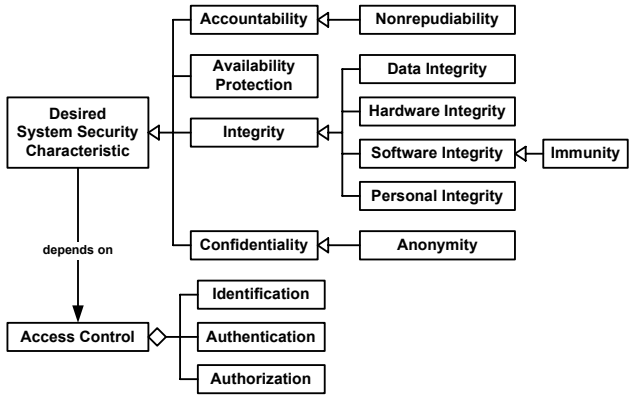
When used, categories should be well defined (e.g., unambiguous)

## Types of Harm






## Security Characteristics as Types of Harm






## Harm Severity

**Harm severity** is an appropriate categorization of the amount of harm.

Harm severity categories can be standardized (ISO, military, industry-wide) or endeavor-specific.

Harm severity categories need to be:

- Clearly identified.
- Appropriately and unambiguously defined.

Note that some standards confuse harm severity with hazard "severity" (i.e., categorization of hazard based on the severity of harm that its accidents can cause)

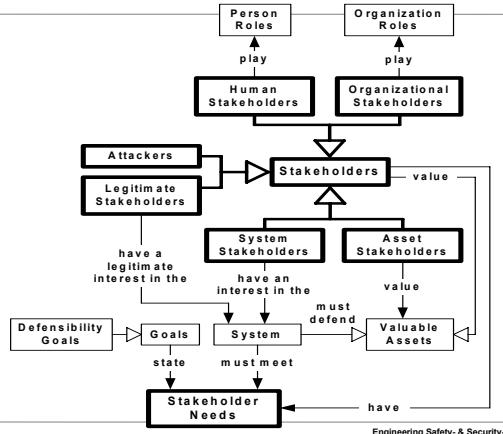


## Example Harm Severity Categories

The International Electrotechnical Commission (IEC) standard, *Medical Electrical Equipment - Part 1: General Requirements for Safety* (IE 601-1-4: 1996), defines harm severity levels as follows:

- **Catastrophic**:
  - Potential of multiple deaths or serious injuries
- **Critical**:
  - Potential of death or serious injury
- **Marginal**:
  - Potential of injury
- **Negligible**:
  - Little or no potential of injury

## Stakeholders






## Some ZATS Stakeholders

People:

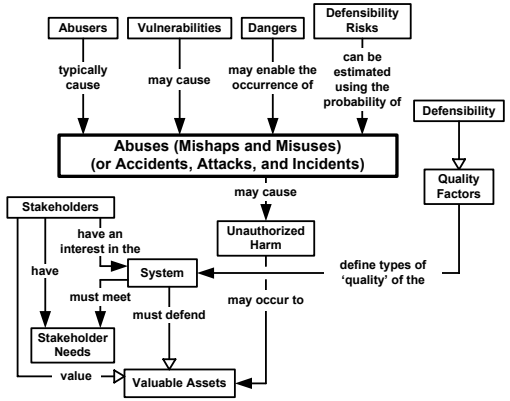
- Emergency Responders
- Passengers
- Operators
- Maintainers
- ZATS Developers
- Zoo Employees
- Zoo Management

Organizations:

- Bank Card Processing Gateway
- Safety and Security Certification/Accreditation Bodies
- Zoo Regulatory Bodies

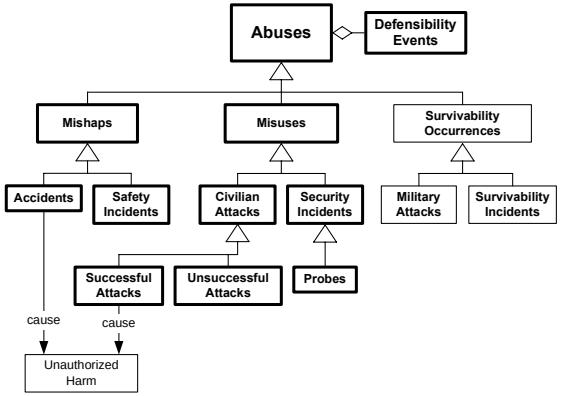


## Abuses (Accidents and Attacks)






## Types of Abuses



## Importance of Accidents

Accidents can have expensive and potentially fatal repercussions:

- Ariane 5 Maiden Launch
  - Reuse of Ariane 4 software not matching Ariane 5 specification
- Mars Climate Orbiter ($125 million)
  - English vs. Metric units mismatch
- Mars Polar Lander
  - Missing requirement concerning touchdown sensor behavior
- Therac–25 Radiation Therapy Machine
  - Timing of unusual input sequence results in unexpected output
- Patriot Missile Battery Misses SCUD
  - Missing availability (uptime) requirement



## Example ZATS Abuses (Mishaps and Misuses)

Accidents:

- Natural Disasters
- Taxi Accidents
- Taxi Station Accidents

Safety Incidents:

- Inadequate Headway
- Overspeed

Attacks:

- Arson
- Cyber-attacks

Security Incidents:

- Antivirus Software Works



## Abuse Likelihood Categories

**Abuse Likelihood Categorization** is an appropriate categorization of the probability that an abuse occurs.

Abuse Likelihood Categories:

- Can be standardized (ISO, military, industry-wide) or endeavor-specific.
- Need to be identified and defined.

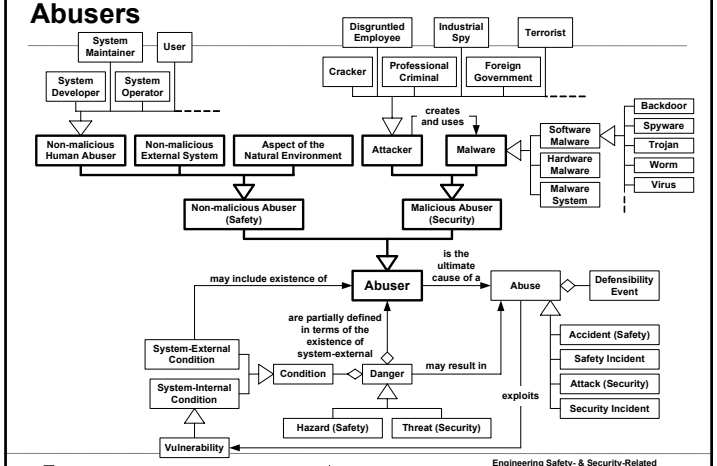
Example Abuse Likelihood Categories include:

- Frequent
- Probable
- Occasional
- Remote
- Implausible

Abuse Likelihood Categories need to be carefully and unambiguously defined.



## Abusers



## Example ZATS Abusers

Non-Malicious Abuser:

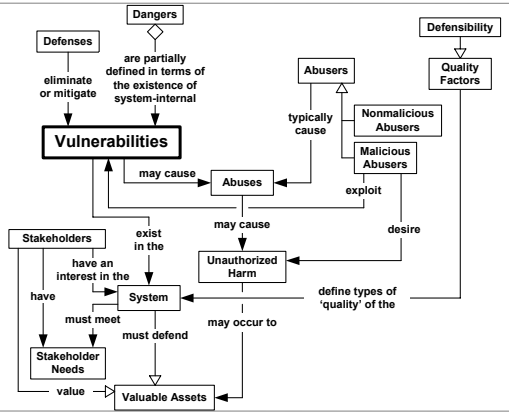
- Human Abuser (e.g., Developer, Maintainer, Operator, Passenger)
- External Systems (e.g., Communications Network, Electrical Power Grid)
- Natural Environment (e.g., River or Weather)

Malicious Abuser:

- Attackers (e.g., Arsonists, Crackers, Terrorists, Thieves)
- Malware (e.g., virus, Trojan horse, Worm)



## Vulnerabilities






## Vulnerabilities

A **vulnerability** is a potential or actual *system-internal weakness* (defect) in the architecture, design, implementation, integration, or deployment of a system that enables:

- A danger (hazard or threat) to exist.
- A safety or security incident to occur.
- An accident or attack to occur.

Only relevant to requirements if a requirement needs to be specified to prevent the vulnerability or mitigate its negative consequences.

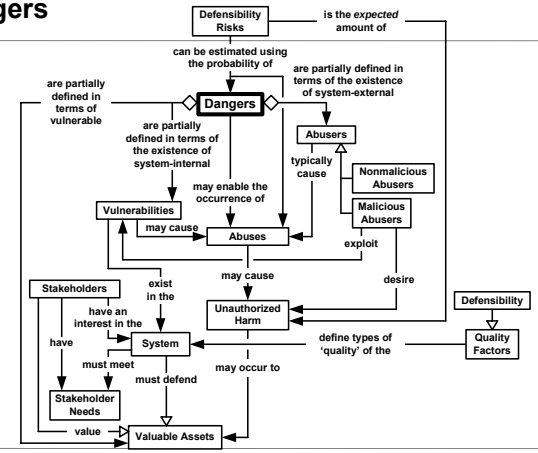
For example, if taxi doors do not have locks or lock sensors.

Ways to Identify Vulnerabilities include:

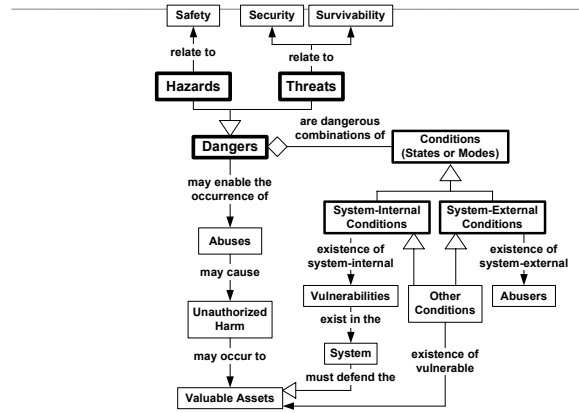
- Analyze Historical Data
- Identify Hardware or Software Defects
- Consider Hardware or Software Failures that can cause Vulnerabilities.



## Dangers



## Types of Dangers

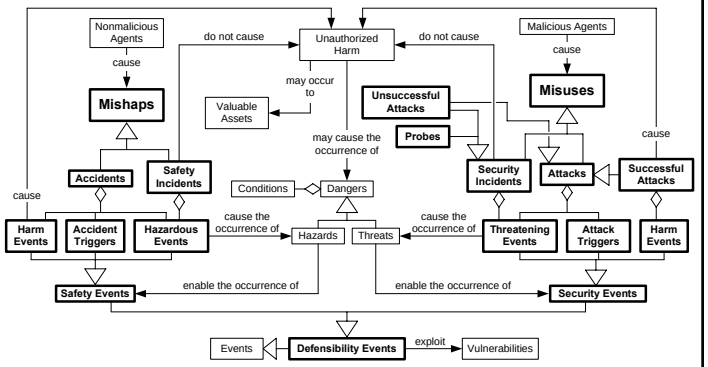



Software Engineering Institute | Carnegie Mellon

Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University

93

## Mishaps and Misuses vs. Hazards and Threats

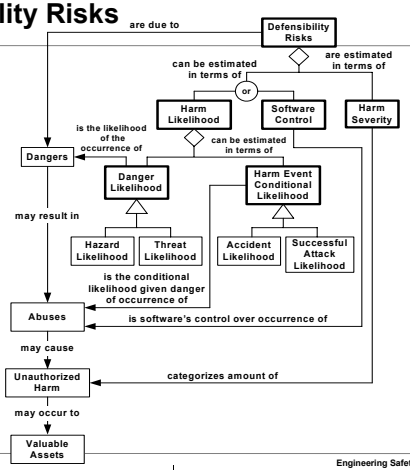



Software Engineering Institute | Carnegie Mellon

Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University

94

## Defensibility Risks

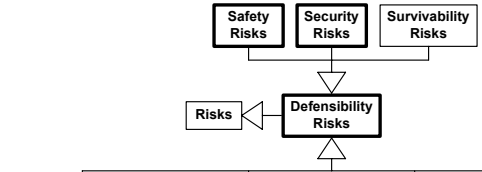



Software Engineering Institute | Carnegie Mellon

Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University

95

## Types of Risks



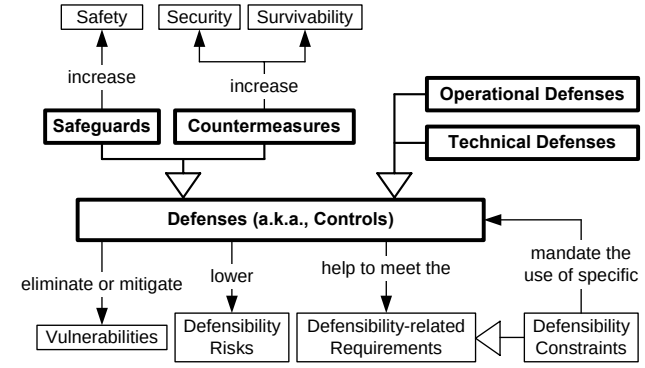

| Risks of Unauthorized Harm | Risks to Valuable Assets | Risks due to Accidents or Attacks | Risks due to Hazards or Threats |
|---|---|---|---|
| Expected Amount of Specific, Specific Types of, or All Unauthorized Harm (Regardless of Asset, Accident, Attack, Hazard, or Threat) | Expected Amount of Specific, Specific Types of, or All Unauthorized Harm to Specific or Specific Types of Valuable Assets | Expected Amount of Specific, Specific Types of, or All Unauthorized Harm due to Specific or Specific Types of Accidents or Attacks | Expected Amount of Specific, Specific Types of, or All Unauthorized Harm due to Specific or Specific Types of Hazards or Threats |

Software Engineering Institute | Carnegie Mellon

Engineering Safety- & Security-Related Requirements ICSE Tutorial
Donald Firesmith, 21 May 2007
© 2007 Carnegie Mellon University

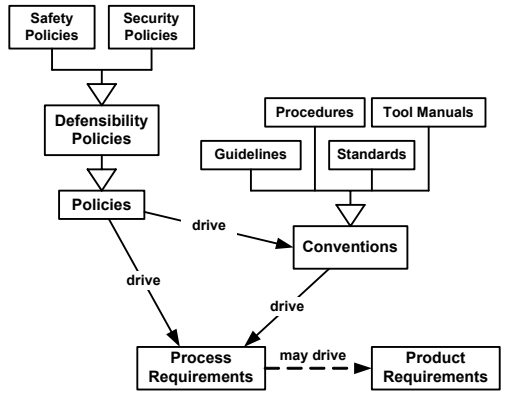
96

## Types of Defenses






## Safety and Security Policies and Conventions






## Safety and Security Policies

**Policy** – a strategic *process* decision that establishes a desired goal.

**Safety Policy** – a policy that enables the achievement of one or more *safety goals*:

- "The overall responsibility for safety must be identified and communicated to all stakeholders."
- "A hazard analysis shall be performed during early in the project."
- "All users will have safety training."

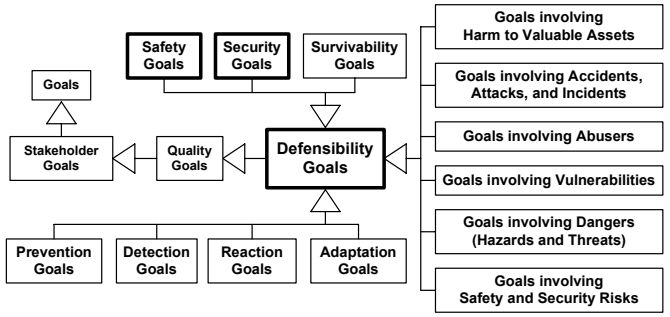
Policies are typically collected into Safety or Security Policy Documents.

In practice, policies are confused with requirements, and conversely policy documents may sometimes include requirements.

Why can this cause problems?



## Types of Defensibility Goals



## Safety- and Security-Related Requirements



## You Are Here

Three Disciplines

Challenges

Common Example

Requirements Engineering Overview

Safety and Security Engineering Overview

**Types of Safety- and Security-related Requirements ◄**

Common Consistent Collaborative Method

Conclusion



## Types of Safety- and Security-Related Requirements

Too often only a Single Type of Requirements is considered.

Not just:

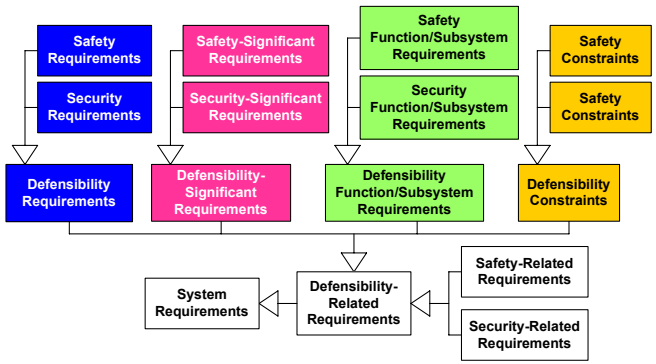
- Special Non-Functional Requirements (NFRs):
  - Safety and Security Requirements are Quality Requirements are NFRs
- Safety- and Security-Significant Functional, Data, and Interface Requirements
- Constraints on Functional Requirements
- Architecture and Design Constraints
- Safety and Security Functions/Subsystems
- Software Requirements

Reason for Presentation Title

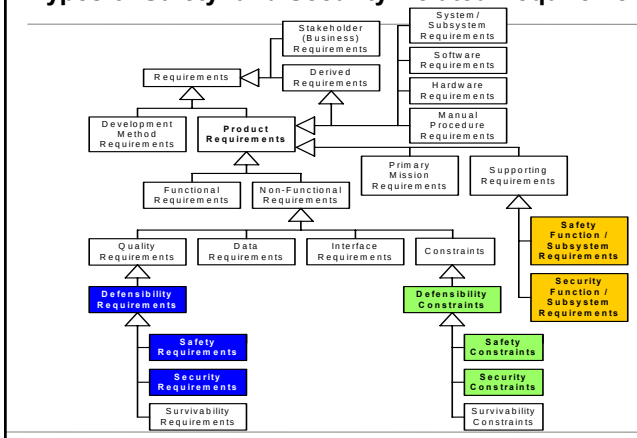
Safety- and Security-Related Requirements for Software-Intensive Systems



## Types of Defensibility-Related Requirements



## Types of Safety- and Security-Related Requirements






## Safety and Security Requirements

Safety and Security Requirements are *Quality* Requirements.

Quality Requirements are Product Requirements that specify a mandatory amount of a type of product quality (i.e., quality factor or quality subfactor).

Quality Requirements should be:

- Scalar (How Well or How Much)
- Based on a Quality Model
- Specified in Requirements Specifications
- Critically Important Drivers of the Architecture



## Example Safety Requirement Templates

"When in mode V, the system shall not cause *accidental harm* of type W to valuable assets of type X at an average rate of more than Y asset value per Z time duration."

"When in mode W, the system shall not cause *mishaps* of type X with an average rate of more than Y mishaps per Z trips."

"When in mode X, the system shall not cause *hazard* Y to exist more than an average of Z percent of the time."

"When in mode X, the system shall not have a *safety risk level* of X."

"When in mode X, the system shall *detect accidents* of type Y an average of at least Z percent of the time."

"Upon detecting an accident of type Y when in mode X, the system shall *react* by performing functions Y an average of at least Z percent of the time."



## Example Security Requirement Templates

"When in mode V, the system shall not limit the occurrence of *malicious harm* of type W to valuable assets of type X to an average rate of less than Y asset value per Z time duration."

"When in mode W, the system shall not prevent *successful attacks* of type X to an average rate of less than Y attacks per Z time duration."

"When in mode X, the system shall prevent *hazard* Y from existing for more than an average of Z percent of the time."

"When in mode X, the system shall not have a *security risk level* of X."

"When in mode X, the system shall *detect misuses* of type Y an average of at least Z percent of the time."

"Upon detecting a misuse of type Y when in mode X, the system shall *react* by performing functions Y an average of at least Z percent of the time."

## Safety- and Security-Significant Requirements

Are identified based on Safety or Security (e.g., hazard or threat) Analysis

Subset of non-Safety and non-Security Requirements:

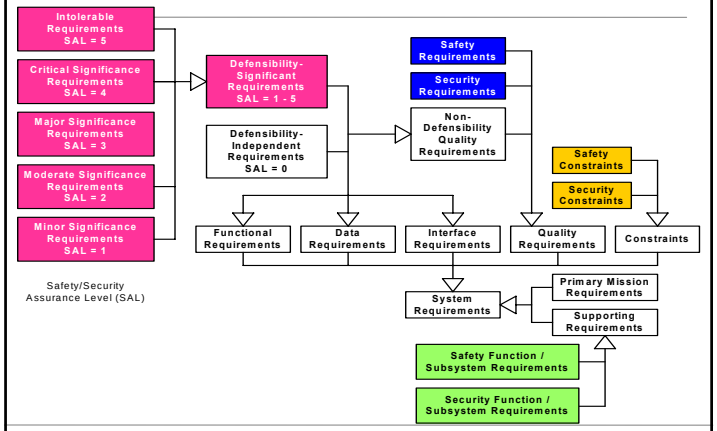
- Functional Requirements
- Data Requirements
- Interface Requirements
- Other Quality Requirements
- Constraints

Safety/Security Assurance Level (SAL) is not 0:

- May have minor Safety/Security Ramifications
- May be Safety- or Security-Critical
- May have intolerable Safety or Security Risk



## Types of Defensibility-Related Requirements






## SALs and SEALs

Safety/Security Assurance Level (SAL)

> a category of required safety or security for safety- or security-significant requirements.

Safety/Security Evidence Assurance Level (SEAL)

> a category of required evidence needed to assure stakeholders (e.g., safety or security certifiers) that the system is sufficiently safe or security (i.e., that it has achieved its required SAL).

SALs are for *requirements*

SEALs are for *components* that collaborate to fulfill requirements (e.g., *architecture*, design, coding, testing)



## Safety-Significant Requirements (cont)

Require enhanced Safety/Security Evidence Assurance Levels (SEALs) including more rigorous development process (including better requirements engineering):

- Formal Specification of Requirements
- Fagan Inspections of Requirements

Too often SEALs only apply to design, coding, and testing:

- Safe Subset of Programming Language
- Design Inspections
- Extra Testing

Architecture and Training (process) also important

## Example Safety-Significant Requirements

ZATS Safety-Significant Requirements:

- Controlling Doors (Opening, Closing, Locking)
- Accelerating and Decelerating (Power Braking System)
- Merging Traffic

Firing Missiles from Military Aircraft Requirements:

- When to Arm Missiles
- Controlling Doors before and after firing missiles
- Detecting Weight-On-Wheels

Chemical Plant Requirements:

- Mixing and Heating Chemicals
- Controlling Exothermic Reactions
- Detecting Temperature and Pressure




## Example ZATS Security-Significant Requirements

Access Control Requirements:

- Identification, Authorization, and Authorization

Integrity:

- Storage and Transmission of Sensitive Data
- Software that might get Infected by Malware
- Software that might represent Intellectual Property

Confidentiality

- Handling of Sensitive Information

Availability (under attack):

- Services Subject to Denial-of-Services Attacks

Non-repudiation:

- Transactions




## Safety and Security Function/Subsystem Rqmts.

**Defensibility Function/Subsystem Requirements** are requirements for functions or subfunctions that exist strictly to improve defensibility (as opposed to support the primary mission requirements).

- **Safety Function/Subsystem Requirements** are requirements for safety functions or subsystems.
- **Security Function/Subsystem Requirements** are requirements for security functions or subsystems.




## Example Safety Function/Subsystem Rqmts$_1$

Functions or subsystems strictly added for safety:

- Aircraft Safety Subsystems:
  - Collision Avoidance System
  - Engine Fire Detection and Suppression
  - Ground Proximity Warning System (GPWS)
  - Minimum Safe Altitude Warning (MSAW)
  - Wind Shear Alert
- Nuclear Power Plant:
  - Emergency Core Coolant System

All requirements for such functions/subsystems are safety-related.

## Example Safety Function/Subsystem Rqmts$_2$

"Except when the weapons bay doors are open or have been open within the previous 30 seconds, the weapons bay cooling subsystem shall maintain the temperature of the weapons bay below X° C."

"The Fire Detection and Suppression Subsystem (FDSS) shall detect smoke above X ppm in the weapons bay within 2 seconds at least 99.9% of the time."

"The FDSS shall detect temperatures above X° C in the weapons bay within 2 seconds at least 99% of the time."

"Upon detection of smoke or excess temperature, the FDSS shall begin fire suppression within 1 second at least 99.9% of the time."




## Example Security Function/Subsystem Rqmts

Functions or subsystems strictly added for security:

- Access Control
- Antivirus Subsystem
- Encryption/Decryption
- Firewalls
- Intrusion/Detection Subsystem

All requirements for such functions/subsystems are security-related.

Look in the Common Criteria for many reusable example security function requirements.




## Example Security Function/Subsystem Rqmts

"The system shall use encryption/decryption to protect confidential information."

"The system shall incorporate an intrusion and detection subsystem."

"The system shall incorporate a virus detection subsystem."




## Safety and Security Constraints

A **Constraint** is any Engineering Decision that has been chosen to be mandated as a Requirement. For example:

- Architecture Constraints
- Design Constraints
- Implementation Constraints
  (e.g., coding standards or safe language subset)
- Testing Constraints

A **safety constraint** is any constraint primarily intended to ensure a minimum level of safety (e.g., a mandated safeguard).

A **security constraint** is any constraint primarily intended to ensure a minimum level of security (e.g., a mandated countermeasure).

Safety and Security Standards often mandate Industry Best Practices as Constraints.

## Example ZATS Safety Constraints

"When the vehicle is stopped in a station with the doors open for boarding, the horizontal gap between the station platform and the vehicle door threshold shall be no greater than 25 mm (1.0 in.) and the height of the vehicle floor shall be within plus/minus 12 mm (0.5 in.) of the platform height under all normal static load conditions…"

Automated People Mover Standards – Part 2: Vehicles, Propulsion, and Braking (ASCE 21-98)

"Oils and hydraulic fluids shall be *flame retardant*, except as required for *normal lubrication*."

Note need to define flame retardant and normal lubrication.



## Example Security Constraints

"Servers shall be protected by firewalls."

"Users shall be identified and authenticated by textual user IDs and associated pass phrases."

"Sensitive data shall be protected by use of a COTS public key encryption/decryption product."

"Malware infection shall be prevented by the use of a COTS antivirus product."



## Common Process:

*A Basis for Effective Collaboration*



## You Are Here

Three Disciplines

Challenges

Common Example

Requirements Engineering Overview

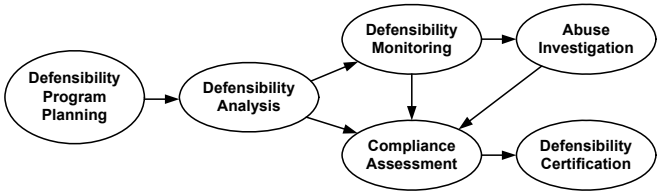
Safety and Security Engineering Overview

Types of Safety- and Security-related Requirements

**Common Consistent Collaborative Method ◄**

Conclusion

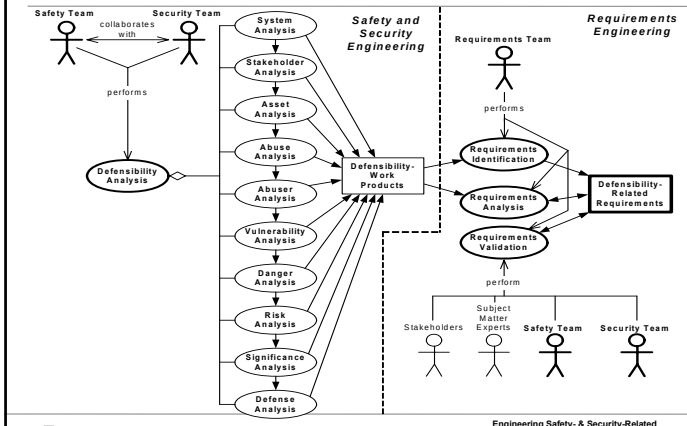
## Overall Defensibility Engineering Method







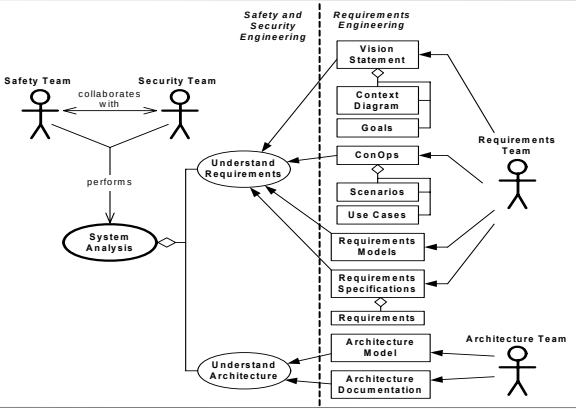
## Defensibility & Requirements Engineering







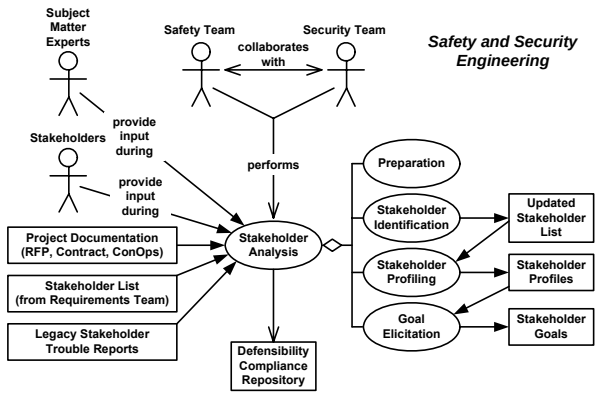
## Systems Analysis



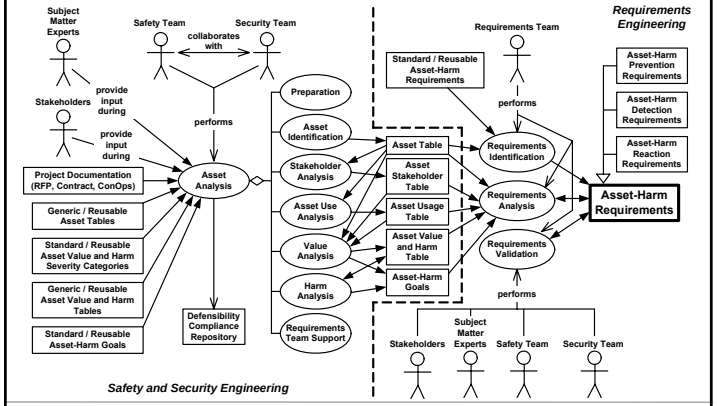




## Stakeholder Analysis

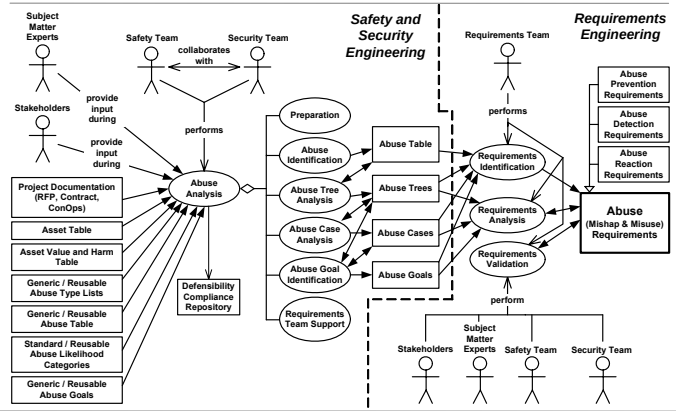


## Asset Analysis



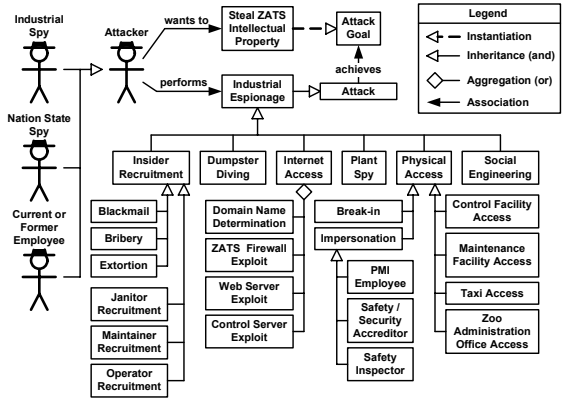



## Abuse (Misuse and Mishap) Analysis



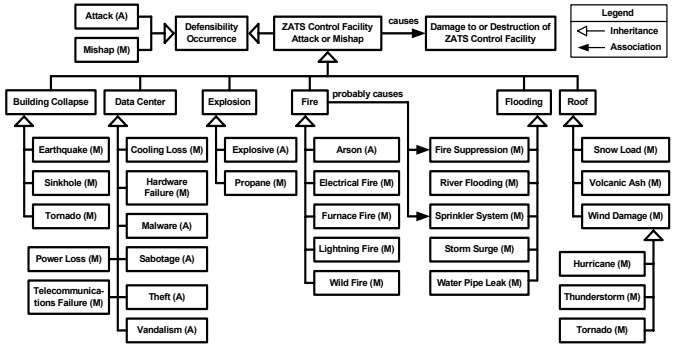



## Example Abuse (Attack) Tree

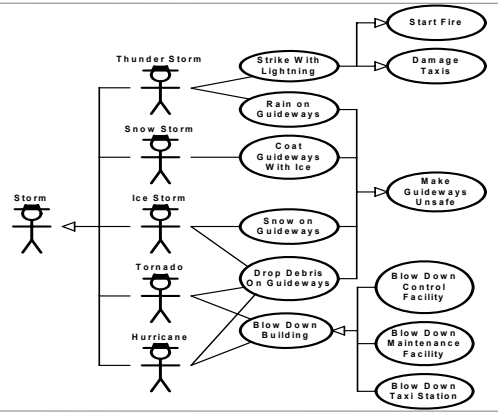





## Example Abuse (Mishap and Attack) Tree



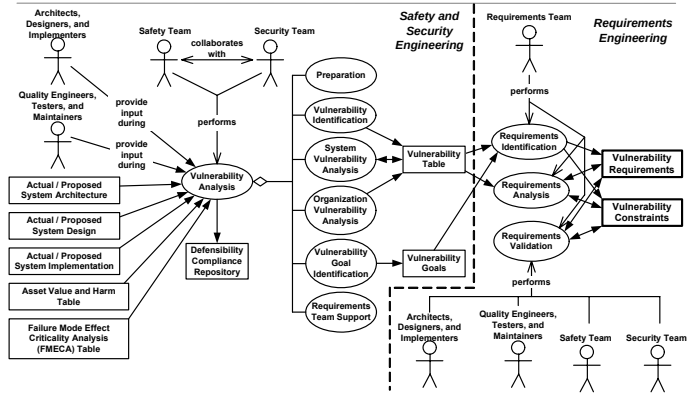
## Example Abuse (Mishap and Misuse) Cases







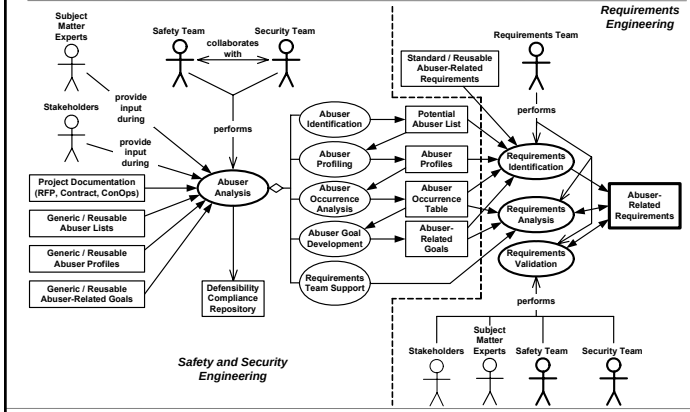
## Vulnerability Analysis







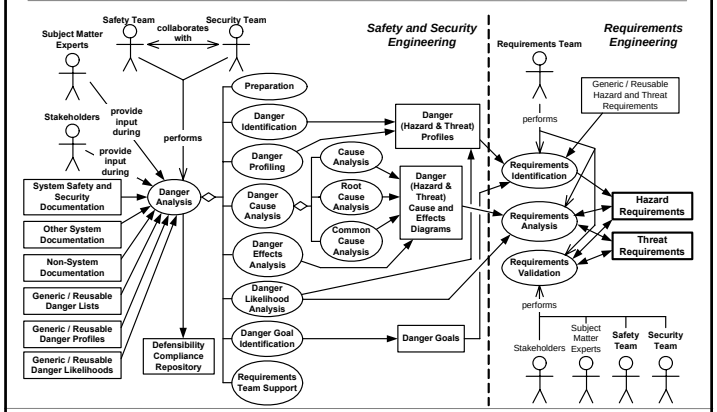
## Abuser Analysis







## Danger Analysis



## Hazard Analysis (Safety)$_1$

Hazard analysis usually implies the analysis of assets, harm, incidents, hazards, and risks.

Hazard analysis often occurs multiple times before various milestones:

- Preliminary Hazard Analysis (PHA)
- System Hazard Analysis (SHA)

Hazard analysis should probably be performed continuously.



## Hazard Analysis (Safety)$_2$

Traditional hazard analysis techniques:

- Come from reliability analysis
- Concentrate on failure analysis
- Do not address all safety concerns

Safety and reliability are not the same:

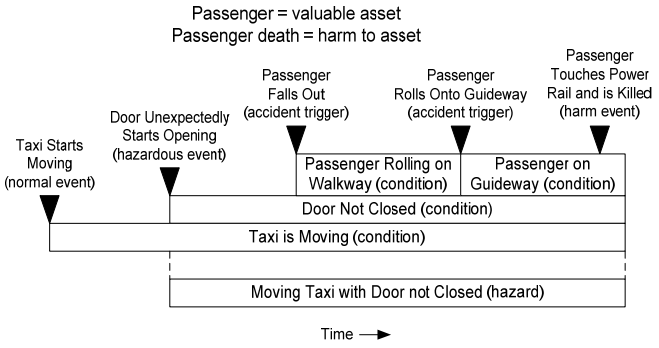
- Unreliable system that is safe (does nothing)
- Safe system that is unreliable (failures do not cause accidents)

Techniques include:

- Event Tree Analysis (ETA)
- Fault Tree Analysis (FTA)
- Hazard Cause and Effect Analysis (HCEA)
- Failure Mode Effects Criticality Analysis (FMECA)



## Example Hazard, Events, Harm, and Asset






## Fault Tree Analysis (FTA)

Develop fault trees (deductive, backwards-search, decision trees) to identify *causes of failures*.

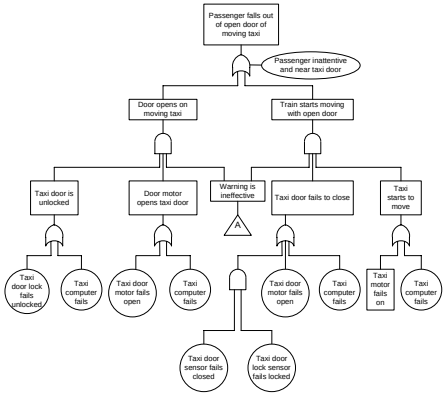
Advantages:

- Long history (1962) of successful use (system/hardware reliability)
- Good documentation
- Well known (in reliability engineering community)
- Good tool support
- Can support quantitative analysis (often impractical for SW)
- Can be used to (indirectly) identify hazards, common causes of safety events, safeguards, and associated requirements

Disadvantages:

- System architecture needed
- Only events, not states (e.g., hazards)
- Non-intuitive symbology that is inconsistent with event trees
- Very expensive and time consuming to produce
- Requires significant analyst expertise
- Ignores system mode

## Example Fault Tree






## Event Tree Analysis (ETA)

Develop Event Trees (inductive, forwards-search, decision trees) to identify *consequences of failures*.

Advantages:

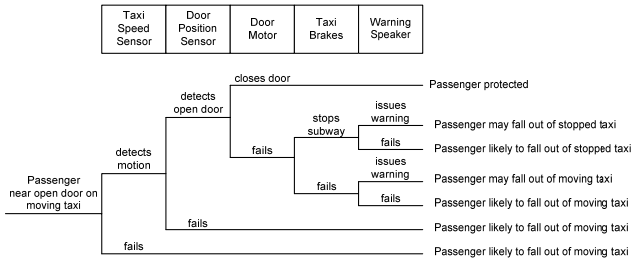
- Long history (1962) of successful use (system/hardware reliability)
- Good documentation
- Well known (in reliability engineering community)
- Good tool support
- Can be used to (indirectly) identify safety events, accidents, and associated requirements

Disadvantages:

- System architecture needed
- Only events, not states (e.g., hazards)
- Non-intuitive symbology that is inconsistent with fault trees
- Very expensive and time consuming to produce
- Not all failures lead to safety events, accidents, and incidents
- Ignores system mode



## Example Event Tree






## Hazard Cause and Effect Analysis (HCEA)

Develop cause/effect graphs (deductive and inductive, backwards and forwards search, decision trees) to identify *causes* and *consequences* of *safety events and conditions*.

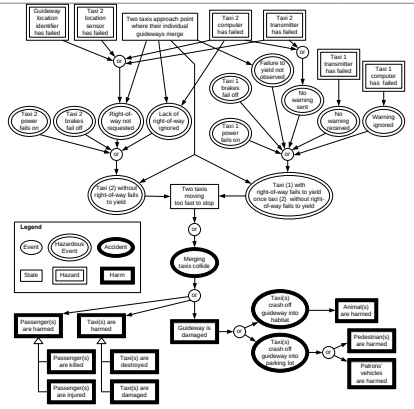
**Advantages:**

- Designed for Safety Analysis
- Emphasize both Events and States (hazards)
- Use single, compatible notation
- Identifies safety events, accidents, incidents, safety conditions, and associated requirements

**Disadvantages:**

- Short history
- Not much documentation
- Not well known
- No tool support
- Very expensive and time consuming to produce
- Ignores system modes

## Example Cause and Effect Tree






## Comparison of Graphical Techniques

| Comparison of Techniques | Event Tree Analysis (ETA) | Fault Tree Analysis (FTA) | Hazard Cause Effect Analysis (HCEA) |
|---|---|---|---|
| Analysis Type | Inductive Analysis | Deductive Analysis | Inductive and Deductive Analysis |
| Graph Type | Event Trees | Fault Trees | Cause and Effect Graphs |
| Search Direction | Forwards Search | Backwards Search | Forwards and Backwards Search |
| Scope | Cause of Failure Events (Causes of Accidents) | Effects of Failure Events (Effects of Hazards) | Causes and Effects of Safety Events and Conditions |
| Use | Identify Hazards | Identify Accidents | Identify Hazards, Accidents, and Incidents |
| Domain | Reliability (Safety) Analysis | Reliability (Safety) Analysis | Safety Analysis |



## Failure Modes Effects Criticality Analysis (FMECA)

FMECA stores relevant information in tabular form:

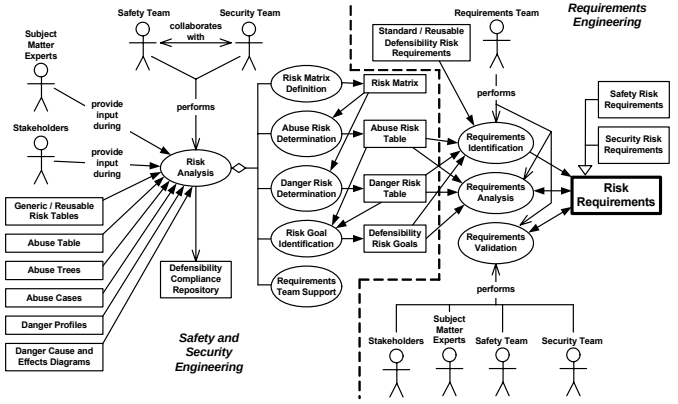
- Architecture component that can fail
- Failure Mode (of component)
- Failure Cause
- Possible Effects of failure
- Effect Severity (i.e., harm severity)
- Failure Probability
- Criticality (risk)
- Controls (reliability/safeguards)



## Example FMECA Table

| Component that fail | Failure Mode | Failure Cause | Failure Effect | Failure Severity | Failure Likelihood | Criticality (Risk) | Safety Controls |
|---|---|---|---|---|---|---|---|
| Accelerometer (Taxi sensor) | No data, Bad data (0, last value, maximum value) | Hardware failure, loss of electrical power, wiring fails | Excessive acceleration or deceleration causing passenger injury | Minor | Low | Moderate | Hardware redundancy, High-reliability COTS component, SW fault tolerance |
| Computer Hardware (Taxi) | Loss of function (complete or intermittent), bad data | CPU, electrical power (loss or spike), hard drive, motherboard, or RAM failure, high temperature | Taxi not controllable (e.g., braking, power, steering), collision between taxis, collision with guideway, unexpected or emergency braking | Severe | Moderate | Critical | Hardware redundancy, High-reliability COTS components, temperature sensor, SW fault tolerance |
| Computer Software (Taxi) | Loss of function (complete or intermittent), incorrect function, bad timing of function | CPU, electrical power (loss or spike), hard drive, motherboard, or RAM failure, high temperature | Taxi not controllable (e.g., braking, power, steering), collision between taxis, collision with guideway, unexpected or emergency braking | Severe | High | Critical | SEAL 1 applied (e.g., SW fault tolerance, real-time operating system, safe language subset, formal specification of core functions, etc.) |

## Defensibility Risk Analysis







## Example Safety Risk Matrix

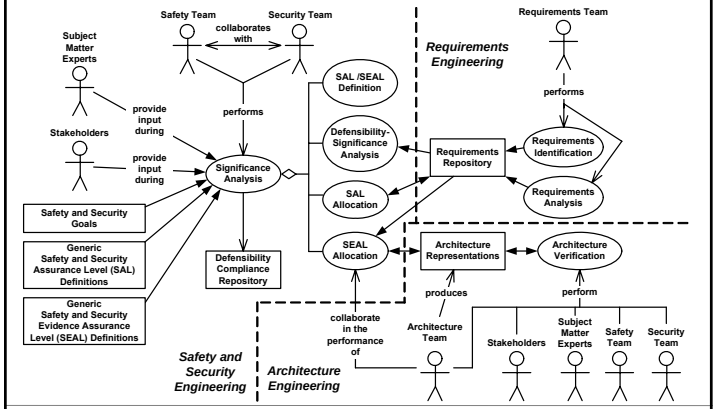
Safety Risk Matrix defines safety risk (and SAL) as a function of:

- Harm severity
- Accident/hazard frequency of occurrence.

| Safety Risks / Safety Assurance Levels (SALs) | | | | | |
|---|---|---|---|---|---|
| | Frequency of Accident / Hazard Occurrence | | | | |
| **Harm Severity** | **Frequent** | **Probable** | **Occasional** | **Remote** | **Implausible** |
| **Catastrophic** | Intolerable | Intolerable | Intolerable | Undesirable | ALARP |
| **Critical** | Intolerable | Intolerable | Undesirable | ALARP | ALARP |
| **Major** | Undesirable | Undesirable | ALARP | ALARP | Acceptable |
| **Minor** | Undesirable | ALARP | ALARP | Acceptable | Acceptable |
| **Negligible** | ALARP | ALARP | ALARP | Acceptable | Acceptable |




## Defensibility Significance Analysis







## Safety/Security Assurance Levels (SALs)

**Safety/Security Assurance Levels (SALs)** are categories of requirements based on their associated safety/security risk level.

SALs can be determined for:

- Individual requirements.
- Groups of related requirements (e.g., features or functions).

SALs should be appropriately, clearly, and unambiguously defined.

## Another Example of Safety/Security Assurance Levels (SALs)

**Intolerable:**
The risk associated with the requirement(s) is totally unacceptable to the major stakeholders. The requirement(s) *must* therefore be deleted or modified to lower the associated risk.

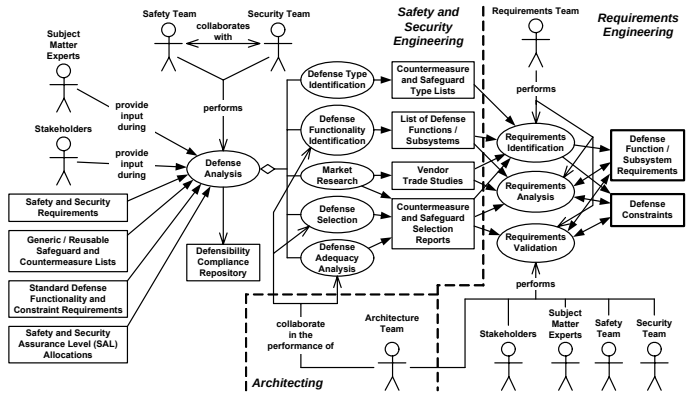
**Undesirable:**
The risk associated with the requirement(s) is so high that major (e.g., architecture, design, implementation, and testing) steps should be taken to lower the risk (e.g., risk mitigation and risk transfer) to lower the risk.

**As Low As Reasonably Practical (ALARP):**
Reasonable practical steps should be taken to lower the risk associated with the requirement(s).

**Acceptable:**
The risk associated with the requirement(s) is acceptable to the major stakeholders and no additional effort must be taken to lower it.

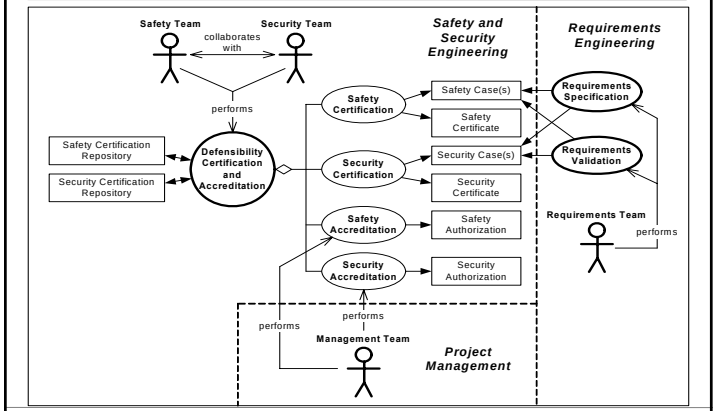


## Defense Analysis






## Defense Certification and Accreditation






## Conclusion:

*Process Improvement Recommendations*

## You Are Here

Three Disciplines

Challenges

Common Example

Requirements Engineering Overview

Safety and Security Engineering Overview

Types of Safety- and Security-related Requirements

Common Consistent Collaborative Method

**Conclusion ◄**



## Conclusion$_1$

Engineering safety-significant requirements requires *appropriate*:

- Concepts
- Methods
- Techniques
- Tools
- Expertise

These must come from*:*

- Requirements Engineering (Safety- and Security-related Requirements)
- Safety Engineering (Analysis and Safety Goals)
- Security Engineering (Analysis and Security Goals)



## Conclusion$_2$

There are four types of Safety- and Security-related Requirements:

- Safety and Security Quality Requirements
- Safety- and Security-Significant Requirements
- Safety and Security Function/Subsystem Requirements
- Safety and Security Constraints

Different Types of Safety- and Security-related Requirements have different Structures.

These different Types of Requirements need to be identified, analyzed, and specified differently.



## Conclusion$_3$

Processes for Requirements Engineering, Safety Engineering, and Security Engineering need to be:

- Properly interwoven.
- Consistent with each other.
- Performed collaboratively and in parallel (i.e., overlapping in time).

## Process Improvement Recommendations

Ensure close Collaboration among Safety, Security, and Requirements Teams.

Better Integrate Safety and Security Processes:

- Concepts and Terminology
- Techniques and Work Products
- Provide Cross Training

Better Integrate Safety and Security Processes with Requirements Process:

- Early during Development Cycle
- Clearly define Team Responsibilities
- Provide Cross Training

Develop all types of Safety- and Security-related Requirements.

Ensure that these Requirements have the proper Properties.



## Final Thoughts

Look for my upcoming book by the same title to be published by Auerbach.

The slides for this tutorial will be put onto the SEI Website in the next 2 weeks, probably on the ASP Publications webpage:

www.sei.cmu.edu/programs/acquisition-support/presentations.html

Questions?

Donald Firesmith
Acquisition Support Program (ASP)
Software Engineering Institute (SEI)
Pittsburgh, Pennsylvania, USA 15213
412-216-0658 (cell)
dgf@sei.cmu.edu



Software Engineering Institute | Carnegie Mellon