Software Safety Program at NREL
(It is Not Just for Nuclear Sites)

CMMI Conference – 2011

Denver Technology Center

Tim Kasse

14 November 2011

Software Safety

What Does it Mean?

How Do We Do It?

Software and Systems Safety

Creating a Safe System is a “team effort”

Software is a vital part of most systems

- What hardware and software does it control?
- How do we know if our software is “safe” or “unsafe”?
- What are the hazards that software contributes to or that software controls?

Why Should Anyone Care About Software Safety?

Why should anyone care about software safety?

- When a device or system can lead to injury, death, the destruction of loss of vital equipment or damage to the environment, software safety is critical!
- Software can respond quickly to potential problems, provide more functionality than equivalent hardware and can even be changed in real time
  - Software is flexible as long as those who develop it and integrate it into the other system components understand the consequences of its possible failure
- Software safety includes all software that can impact hazardous software or hardware
  - All such software is "safety-critical"

Hazardous and Safety-Critical Software Definitions

Hazard – A hazard is the presence of a potential risk situation that can result in or contribute to a mishap

- Every hazard has at least one cause which can turn into a number of effects (damage, illness, failure)

Hazard Cause – A hazard cause may be a defect in hardware or software, a human operator error or an unexpected input or event which results in a hazard

Hazard Control – A hazard control is a method of preventing the hazard, reducing the likelihood of the hazard occurring, or the reduction of the impact of that hazard

Hazardous and Safety-Critical Software Definitions - 2

Hazard Controls use:

- Hardware (e.g., pressure relief or valve)
- Software (e.g., detection of stuck valve and automatic response to open secondary valve
- Operator procedures
- Combination of methods to avert the hazard

Hazard Control Method – For every hazard cause, there must be at least one control method which is usually a design feature or a procedural step

Hazardous and Safety-Critical Software Definitions - 3

Hazard Control Verification – Each hazard control will require verification

- – Test
- – Analysis
- – Inspection
- – Demonstration

Software – Software can be used to detect and control hazards, but software failures can also contribute to the occurrence of hazards!

Safety-Critical Software

Safety-critical software includes:

- Hazardous software which can directly contribute to, or control a hazard
- All software that influence this hazardous software
- Software that monitors hazardous or safety-critical hardware or software

EXAMPLE: Software that controls a high-powered laser is hazardous and safety-critical

Safety-Critical Software - 2

Software that provides information required for a safety-related decision falls into the safety-critical category

- If a human must shut down a piece of hardware when the temperature goes over a threshold, the software that reads the temperature and displays it for the human operator is safety-critical
- All the software along the chain, from reading the hardware temperature sensor, converting the value to appropriate units, to displaying the data on the screen are safety-critical

Safety-Critical Software - 3

Software is safety-critical if it performs any of the following:

- Controls hazardous or safety-critical hardware or software
- Monitors safety-critical hardware or software as part of a hazard control
- Provides information upon which a safety-related decision is made
- Verifies hardware or software hazard controls
- Can prevent safety-critical hardware or software from functioning properly

Software Safety Program

A System Safety Program Plan is a prerequisite to performing development or analysis of safety-critical software. The Safety Plan should:

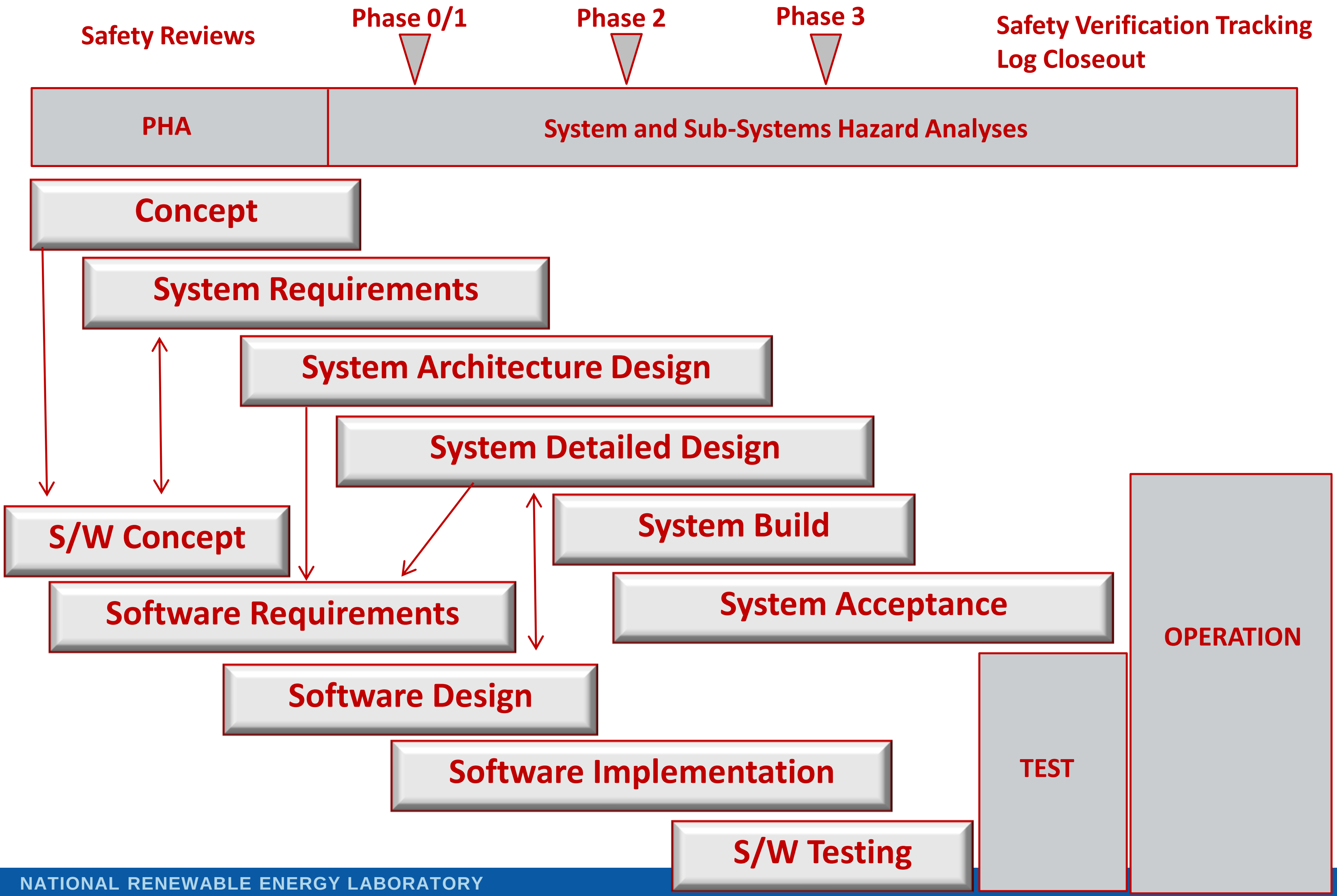
- Describe forms of analysis
- Provide a schedule for performing a series of these system and subsystem level analysis throughout the development schedule

System safety analysis follow the lifecycle of the system development efforts

- The system is compromised of the hardware, the software, and the interfaces between them, including human operators

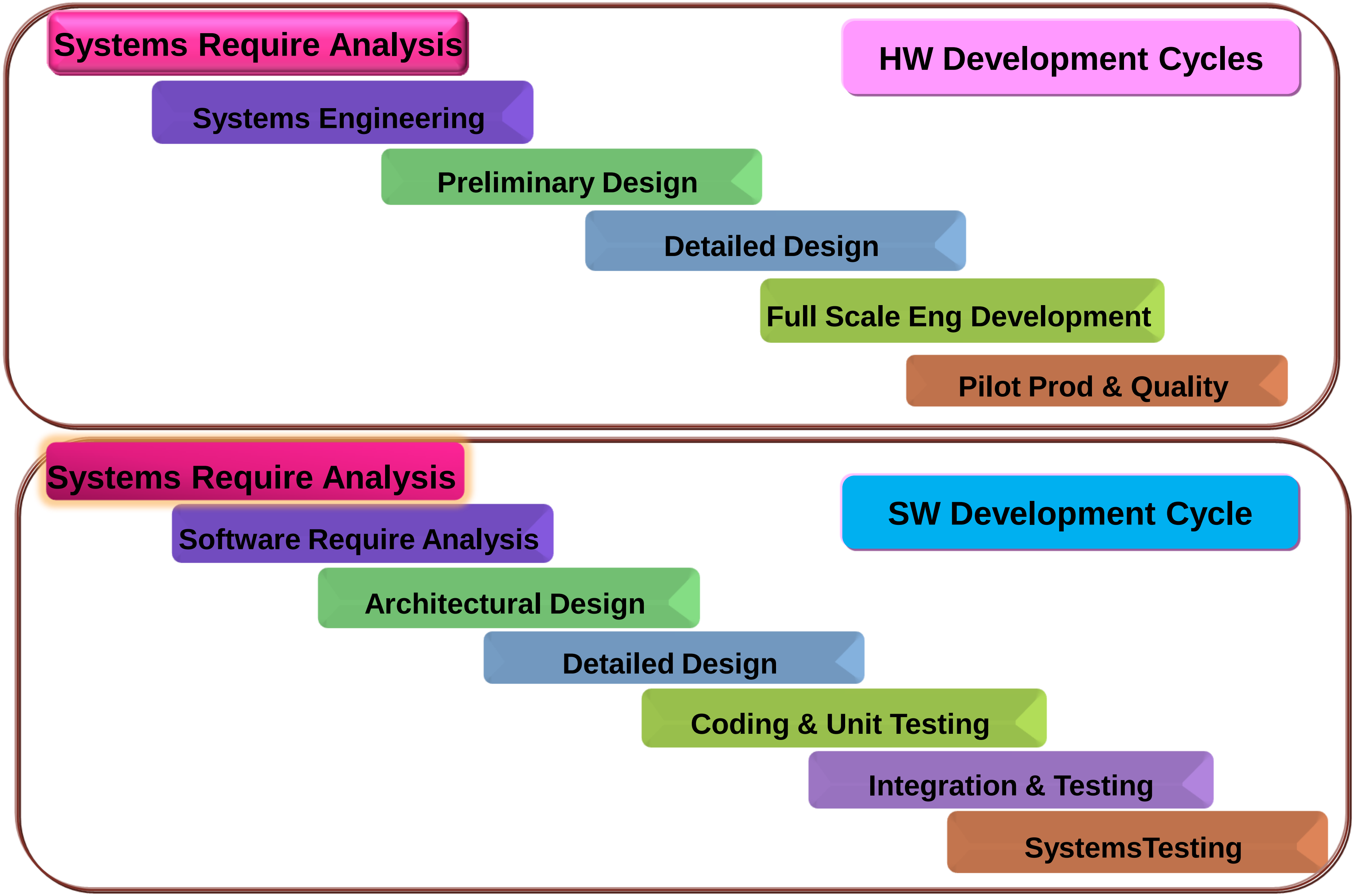
System/ Software Safety Lifecycles

Safety, System, and Software Timeline
(All Inclusive View)



Systems Lifecycle
Software Cycle
Software Safety
Lifecycle Overlay

Systems / Software Product Lifecycles



HW Development Cycles

Systems Require Analysis

Systems Engineering

Preliminary Design

Detailed Design

Full Scale Eng Development

Pilot Prod & Quality

SW Development Cycle

Systems Require Analysis

Software Require Analysis

Architectural Design

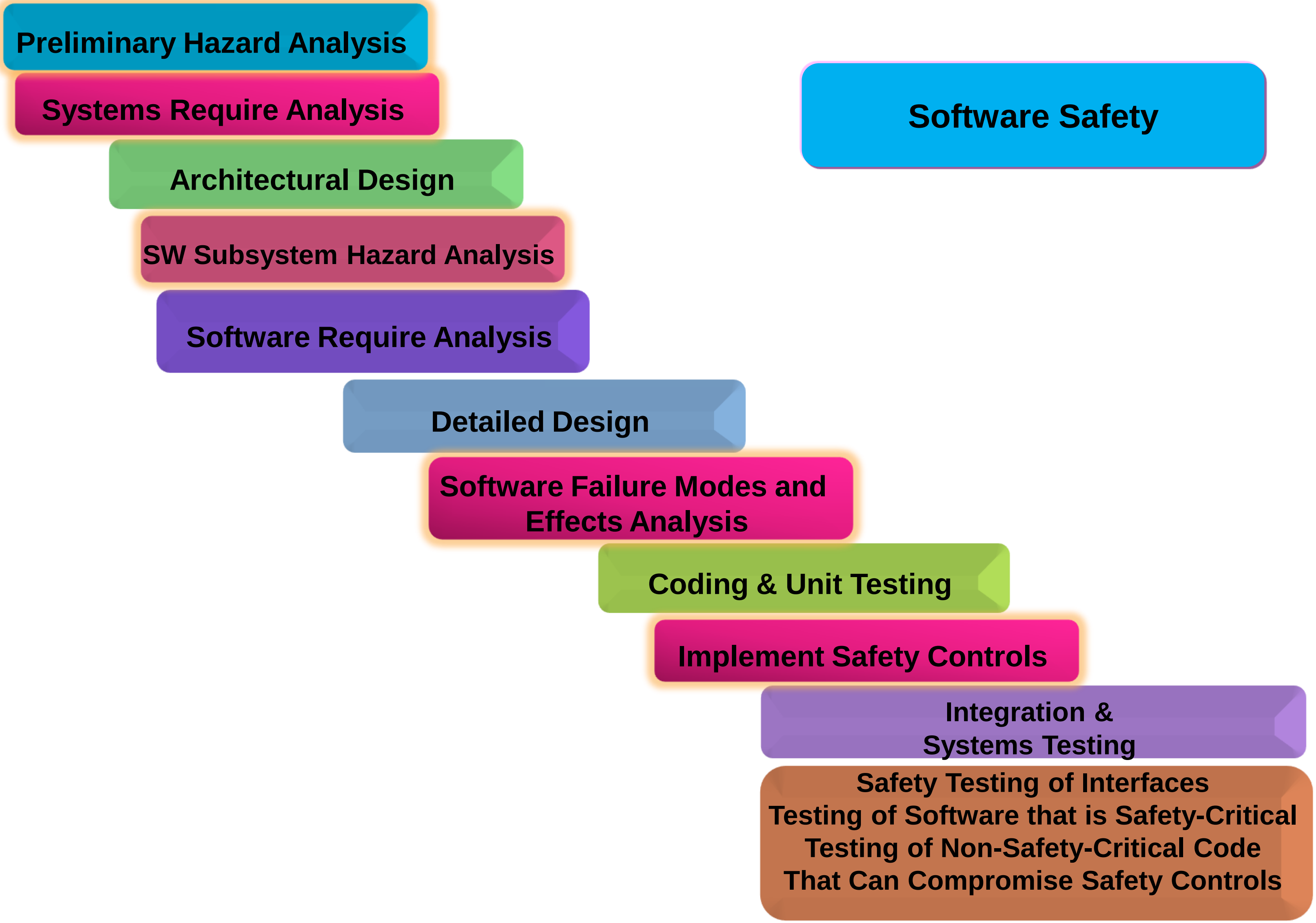
Detailed Design

Coding & Unit Testing

Integration & Testing

SystemsTesting

Software Safety Lifecycle Overlay



Preliminary Hazard Analysis

Systems Require Analysis

Architectural Design

SW Subsystem Hazard Analysis

Software Require Analysis

Detailed Design

Software Failure Modes and Effects Analysis

Coding & Unit Testing

Implement Safety Controls

Integration & Systems Testing

Safety Testing of Interfaces
Testing of Software that is Safety-Critical
Testing of Non-Safety-Critical Code
That Can Compromise Safety Controls

Software Safety

Safety Requirements and Analysis

The first step to any safety analyses is asking the right questions:

- What could go wrong?
- Why won't it?
- What if it did?

Everyone involved in each activity throughout the product lifecycle should think about all the ways the system or software can fail, how to prevent the failure from occurring and how to prevent or mitigate an accident if the failure occurred

Safety Requirements and Analysis - 2

In general there are two types of safety requirements:

- Imposed by standards and regulations
- Technically specific to the project and its operating environments

The most commonly used assessment tool used during this beginning activity is the Preliminary Hazard Analysis (PHA) as illustrated in the Software Safety Lifecycle Overlay

Preliminary Hazard Analysis (PHA)

Before any system with software can be analyzed or developed for use in hazardous operations or environments, a system PHA must be performed

The PHA is the first source of “specific” system safety requirements which may go to the level of pointing out the specific software safety requirements

When a system is decomposed into subsystems, how the software relates to each component must be considered

Software Subsystem Hazard Analysis

Software Subsystem Hazard Analysis determines which of the hazards including hazard causes, hazard controls, hazard mitigations, or hazard verifications have software as a component

Software may impact a hazard in a variety of ways:

- Software failure may lead to a hazard occurring (Hazard Cause)
  - A failure in a data conversation function could incorrectly report a temperature valve resulting in personal damage
- Failure of a Software Hazard May Allow a Hazard to Occur
  - Failure that monitors pressure and opens a valve when it reaches a threshold may be a hazard control
  - Failure of that software would allow an over-pressurization of the vessel and a potential for a rupture or other hazard

Software Subsystem Hazard Analysis - 2

- Software Safing Mode May Fail
  - Failure of the software to detect or shut down a runaway electromechanical device is an example of such an impact
- Software Used to Mitigate the Consequences of An Accident May Fail
  - Software controlling the purging of toxic gas may fail, allowing the gases to remain in the chamber or be vented inappropriately to the outside air
- Software Used to Verify Hazard Hardware or Software Hazard Controls May Fail
  - Failure in this situation would be due to invalid results
  - False positives may allow a potentially hazardous system to be put into operation

Software Safety Planning

If the Preliminary Hazard Analysis (PHA) reveals that any software is safety-critical, a software safety process must be initiated

Standard Participant's in a Successful Safety Process include:

- Project Manager
- Systems Engineers
- System Safety Engineers
- Software Safety Engineers
- Software Developers
- Software Quality Assurance
- Independent Verification & Validation

Standard Participants in a Successful Safety Process

Project Manager

- The Project Manager maintains a high-level overview of the whole process, works with the team to include the extra software development and analysis tasks in the schedule and budget, and can be called upon to help in negotiations between team members
- Ultimately, the Project Manager carries the responsibility for determining the amount and types of risk they are willing to accept for their project

Standard Participants in a Successful Safety Process - 2

Systems Engineers

- Systems Engineers are responsible for designing the system and partitioning the elements between hardware and software.
- Systems Engineers own the “big picture”
- Systems developers can make sure that safety-critical elements are not overlooked in the initial analysis
- Systems Engineers makes sure that “criticality” of the design component is transferred to software later in the design process

Standard Participants in a Successful Safety Process - 3

System Safety Engineers

- System Safety Engineers determine what components of the systems are hazardous
- They look at the system as a whole entity, but also try to understand how the different elements function or fail to function

Software Safety Engineers

- Software Safety Engineers build on the work of the system safety engineers
- They look closely at the requirements and development process to verify that software safety requirements are present and being designed in
- They perform analyses, and tests to verify the safety of the software

Standard Participants in a Successful Safety Process - 4

Software Developers

- Software Developers are the “in the trenches” engineers who design, code, and often test the system
- Software developers can have a great impact on the safety of the software by the design chosen and by implementing defensive programming techniques

Software Quality Assurance

- Software Quality Assurance engineers work with the software developers and software safety engineers to assure that safe software is created
- Software Quality Assurance monitors development processes, assures the traceability of all requirements, performs or reviews analyses, witnesses or performs tests, and provides an “outside” view of both the software process and products

Standard Participants in a Successful Safety Process - 5

Independent Verification & Validation

- For all safety critical projects, the Project Manager is required to perform a self assessment of the project software and report the findings
- IV&V may then perform their own review and present the Project Manager with their estimate for additional analyses
- IV&V is, in addition to Software Quality Assurance and Software Safety and not a replacement for those roles

Identifying Safety-Critical Software

Identifying Safety-Critical Software

Software that can cause a hazard if it malfunctions is considered safety-critical software

- Software which is required to recognize hazardous conditions
- Software which implements automatic safety control actions
- Software which provides a safety-critical service
- Software which inhibits a hazardous event

Complexity

Greater complexity of the software system increases the chances of errors

- Software complexity is increased as the number of safety related software requirements for hazards control increases
- Rough measures of complexity include the number of subsystems controlled by the software and the number of interfaces between:
  - Software to Software
  - Software to User
  - Software to Software Subsystems

Time Criticality

The speed at which a system with software hazard control must react is a major factor

- Does control need to be exercised faster than a human or mechanical system can react?

System and Software Concept Phase

Initial Planning of the Software Development Effort

During the System Concept phase, the software team should involved in the initial planning of the software development effort

Plans typically developed in the Concept Phase include:

- Software Project Plan / Software Management Plan
- Software Development Plan
- Software Quality Assurance Plan
- Software Configuration Management Plan
- Software Safety Plan
- Software Verification and Validation Plan
- Software Acquisition Plan

System Concept Phase Tasks

Software Engineering Tasks	System and Software Safety Tasks
Provide input to project concept and software concept documents	Create the Software Safety Plan, including planning and tailoring the safety effort.
Provide input to Software Safety Plan	Conduct Preliminary Hazard Analysis (PHA)
Plan the software management and development processes	Set up hazards tracking and problem resolution process
Plan the Configuration Management System	Prepare hazard verification matrix
Plan the verification and validation process	Review PHA for safety-critical software
Participate in “make or buy” decisions for software Review software acquisition	Participate in “make or buy” decisions for software Review software acquisition Develop safety-critical software tracking process Conduct Software Subsystem Hazard Analysis

Project / Software Conception Documentation

Document	Software Safety Section
Software Safety Plan	Include software as a subsystem. Identify tasks and personnel
Software Concepts Document	Identify safety-critical processes
Software Management Plan	Discuss coordination with systems safety tasks, flow-down incorporation of safety requirements and applicability to safety-critical software
Software Security Plan	Determine security of safety-critical software
Risk Management Plan	Identify software risks, especially those related to safety and reliability
Software Configuration Management Plan	Identification and handling of safety-critical components
Software Verification and Validation Plan	Discuss verification and validation of safety-critical components
Software Quality Assurance Plan	Identify quality assurance support to software safety function, verification of software safety requirements, and safety participation in software reviews and inspections

Project Management

Project Management

Project Management can be thought of as the establishment and maintenance of an environment that enables work to be performed efficiently and effectively

To effectively manage and control a project, the project leader must be able to:

- Identify the customer(s)
- Define and manage the requirements
- Understand the system that must be built
- Establish the necessary project roles
- Understand the project factors that must be managed
- Establish the project management lifecycle
- Choose a product lifecycle

Project Management - 2

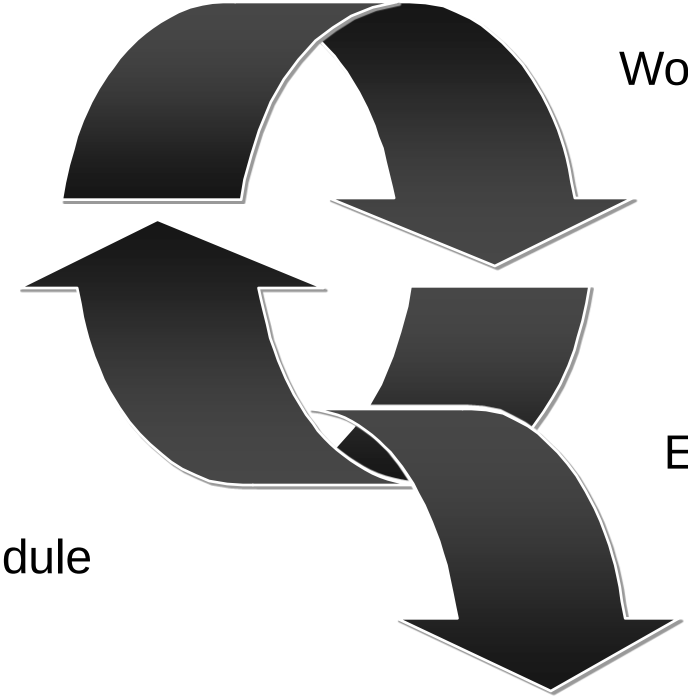
The purpose of project planning is to establish and maintain plans that define project activities and activities of other relevant stakeholders that affect the project and includes:

- Estimating
- Developing the project plan
- Interacting with the relevant stakeholders
- Resolving conflicts

The project plan will need to be revised as the project progresses

- During the project's lifecycle it will become necessary to address changes in requirements, commitment changes, inaccurate estimates and assumptions, necessary corrective actions and process improvements

Iterative Nature of Planning



Goal Statement

Budget

Work

Resources

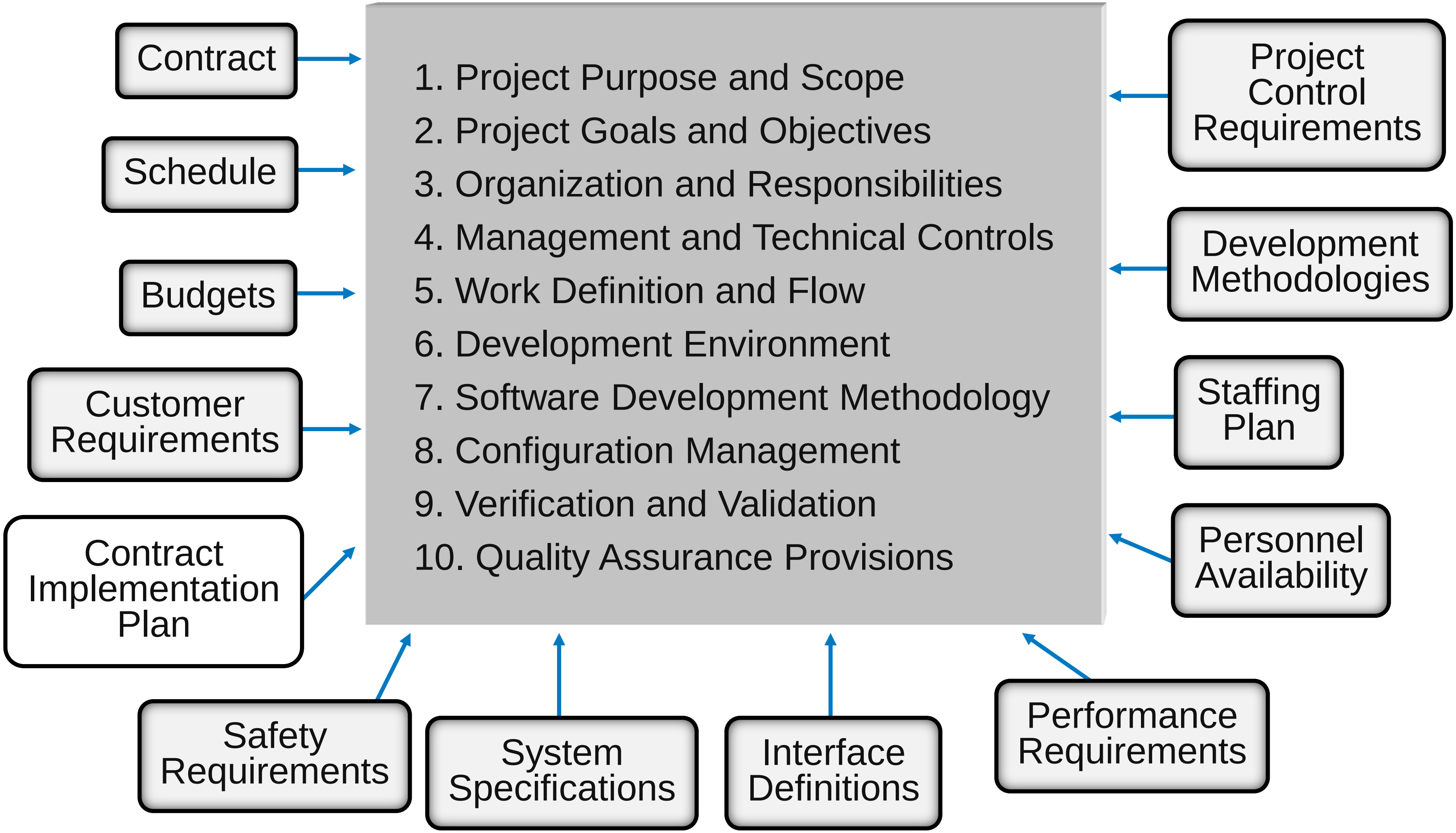
Risks

Estimates

Schedule

Project Baseline

The Project Plan



1. Project Purpose and Scope
2. Project Goals and Objectives
3. Organization and Responsibilities
4. Management and Technical Controls
5. Work Definition and Flow
6. Development Environment
7. Software Development Methodology
8. Configuration Management
9. Verification and Validation
10. Quality Assurance Provisions

Contract

Schedule

Budgets

Customer Requirements

Contract Implementation Plan

Safety Requirements

System Specifications

Interface Definitions

Performance Requirements

Personnel Availability

Staffing Plan

Development Methodologies

Project Control Requirements

Plans That Affect the Project

All plans that affect the project should be reviewed. Candidate plans that affect project success include:

- Quality Assurance plans
- Configuration Management plans
- Risk Management strategy
- Integration strategy
- Verification strategy
- Validation strategy
- Product integration plans
- Software Safety plans
- Documentation plans
- Stakeholder involvement
- Knowledge and skills development

Risk Management

Risk Management

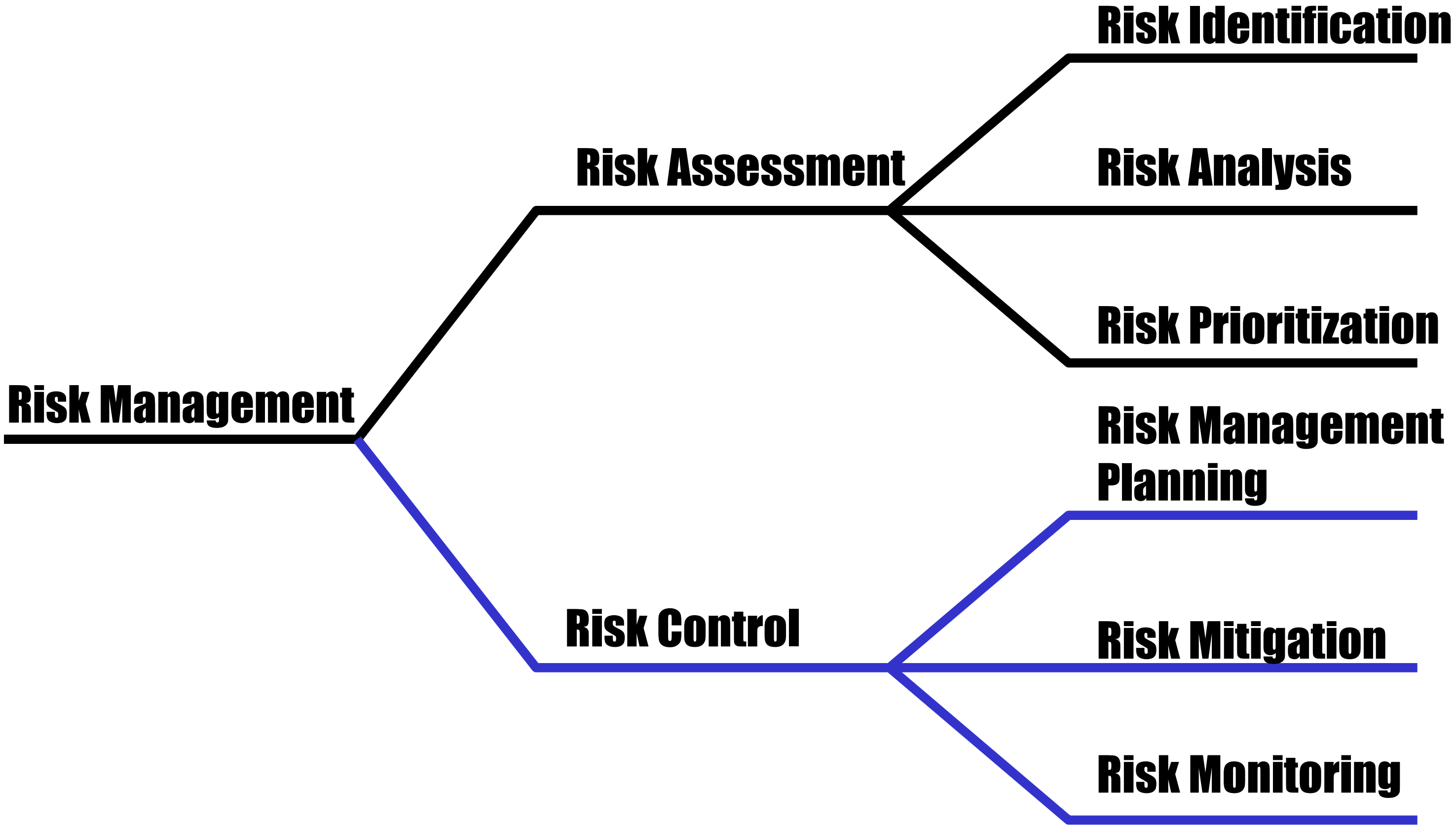
Risk management is a continuous, forward-looking process that is an important part of project management

Risk management should address issues that could endanger achievement of critical objectives

A continuous risk management approach effectively anticipates and mitigates risks that may have a critical impact on a project

- Enables project participants to share a future view of the project and business
- Adds to project members individually and collectively owning the project

Elements of Risk Management



Risk Management

Risk Assessment

Risk Identification

Risk Analysis

Risk Prioritization

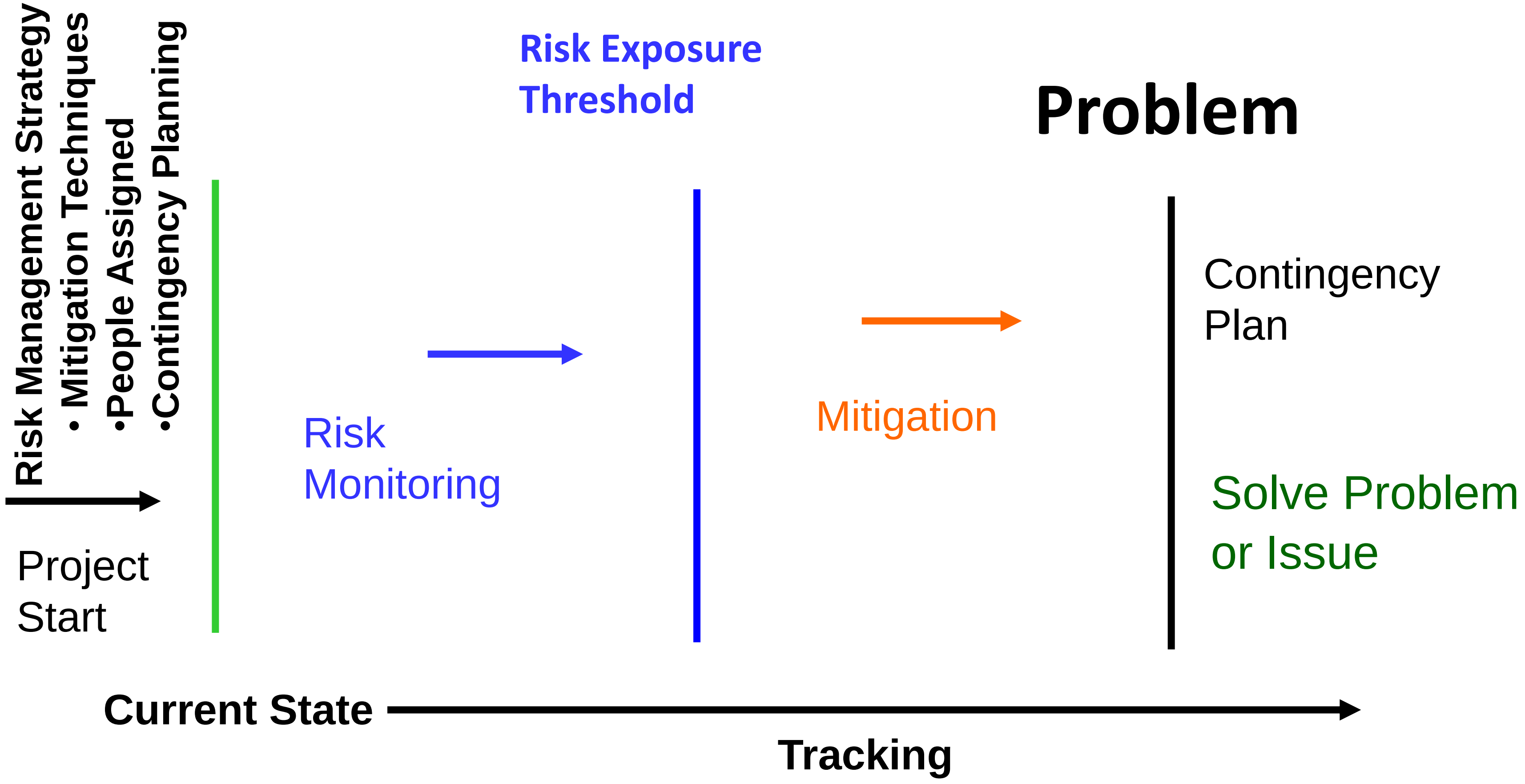
Risk Control

Risk Management Planning

Risk Mitigation

Risk Monitoring

Establishing Risk Thresholds



Hazards and Risks

Hazard analyses such as the PHA are not primarily concerned with whether the hazard is likely to occur or not

All Hazards are bad!

However without unlimited time and money the hazards must be prioritized!

Risk is the combination of:

- The probability that a program will experience an undesired event such as a safety mishap, compromise of security or system component failure
- The consequences, impact, or severity of the undesired event were it to occur

Hazards and Risks - 2

Each project or program needs to define a set of "hazard severity" levels

"Relative risk terms" may be used as long as they are accompanied by explanations or ranges of values that contribute to a common vocabulary among project and program members when discussing hazard severity levels

Combining these two concepts (severity and likelihood) leads to a single risk index value for the hazard

- ***Risk index for hazards is equivalent to risk priority for product development***

Hazard Severity Definitions

	Catastrophic Loss of human life or permanent disability; loss of entire systems of ground facility; severe environmental damage	**Critical** Severe injury or temporary disability; major system or environmental damage
	Moderate Minor injury; minor system damage	**Negligible** No injury or minor injury; some system stress, but no system damage

Likelihood of Occurrence Definitions

Likelihood of Occurrence Definitions	**Likely** The event will occur frequently, such as greater than 1 out of 10 times	**Probable** The event will occur several times in the life of an item
	Possible Likely to occur sometime in the life of an item	**Unlikely** Remote possibly of occurrence during the life of an item
	Improbable Very rare, possibility is like winning the lottery	

Hazard Prioritization

Severity Levels	Likelihood of Occurrence				
	Likely	**Probable**	**Possible**	**Unlikely**	**Improbable**
Catastrophic	1	1	2	3	4
Critical	1	2	3	4	5
Moderate	2	3	4	5	6
Negligible	3	4	5	6	7

System Risk Index

System Risk Index	Class of Safety Activities Recommended
1	Not Applicable as is (Prohibited)
2	Full
3	Moderate
4,5	Minimum
6,7	None (Optional)

Requirements Engineering

Requirements

Requirements elicitation, analysis, review, traceability and management are key elements of a successful and safe software development process

Many of the costly and critical system failures that are attributed to software can ultimately be traced back to missing, incorrect, misunderstood, or incompatible requirements

Requirements - 2

Software requirements flow down from many sources including:

- System Requirements
- Safety and Security Standards
- Hazard and Risk Analyses
- System Constraints
- Customer Input
- Software Safety “Best Practices”

Requirements - 3

The software system must be evaluated to determine if any of it is safety-critical or has safety-critical characteristics.

- Top-down analyses such as Software Fault Tree Analysis are often used to identify safety-critical software
- Any safety-critical characteristics found during the requirements analysis should be written into the software and system requirements

Requirements - 4

Software Safety requirements are derived from the system and subsystem safety requirements which were developed to mitigate hazards identified in the Preliminary Hazard Analysis (PHA)

Software Safety requirements should be included in:

- Software Requirements Specification
- Software Interface Specification or Interface Control Document
- Requirements Traceability Matrix [See Diagrams on Traceability Below]

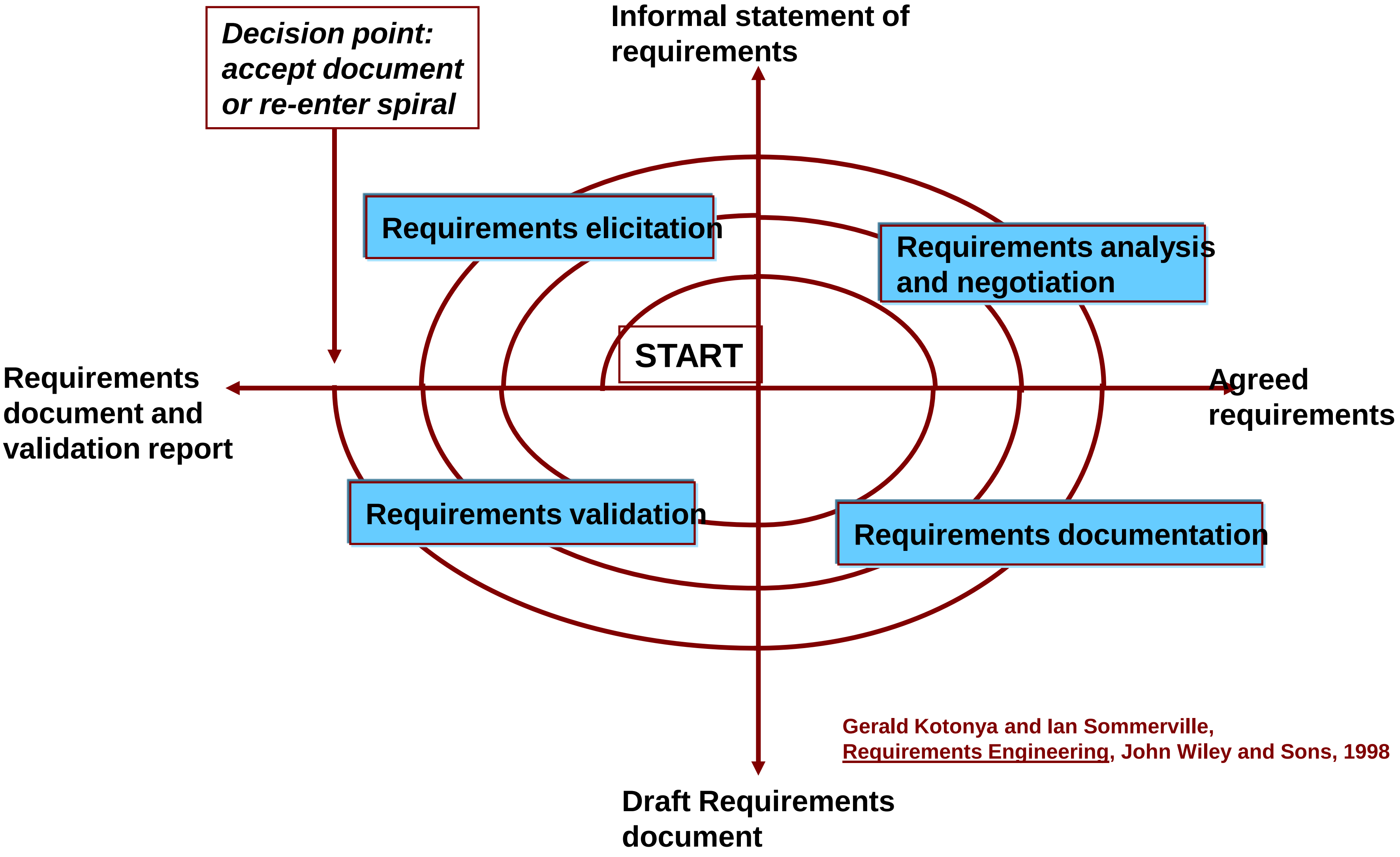
Early Phase Requirements Validation

Customer requirements should be **validated** early in the development schedule to **gain confidence** that the customer requirements are capable of guiding a development that results in the customer's operational needs being met

- Simulations
- Prototypes,
- Analyses
- Scenarios
- Storyboards

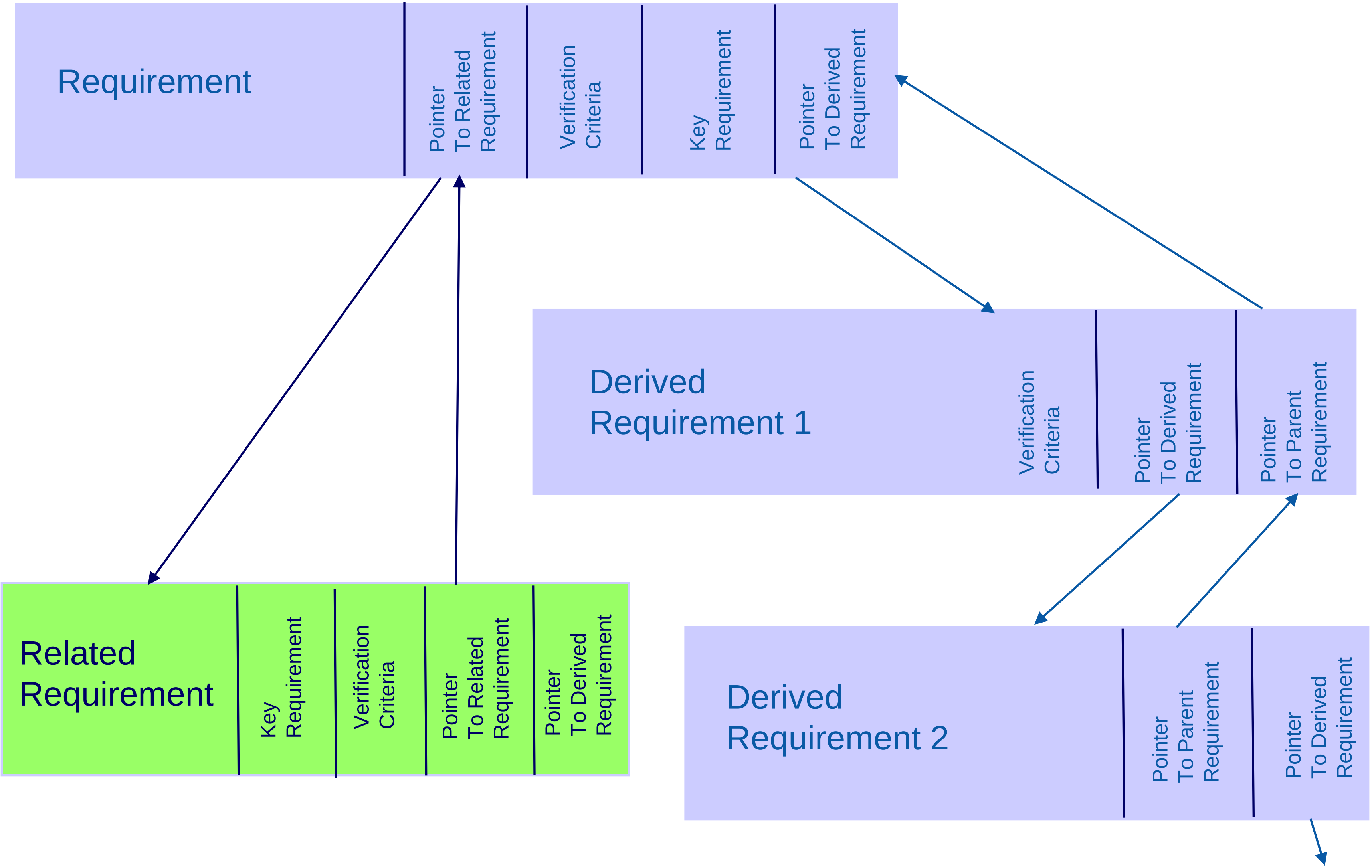
Validation of requirements acts as elicitation of requirements for the "next round"

Spiral Model of the Product Requirements Engineering Process


Decision point: accept document or re-enter spiral
Informal statement of requirements
Requirements elicitation
Requirements analysis and negotiation
START
Requirements document and validation report
Agreed requirements
Requirements validation
Requirements documentation
Draft Requirements document

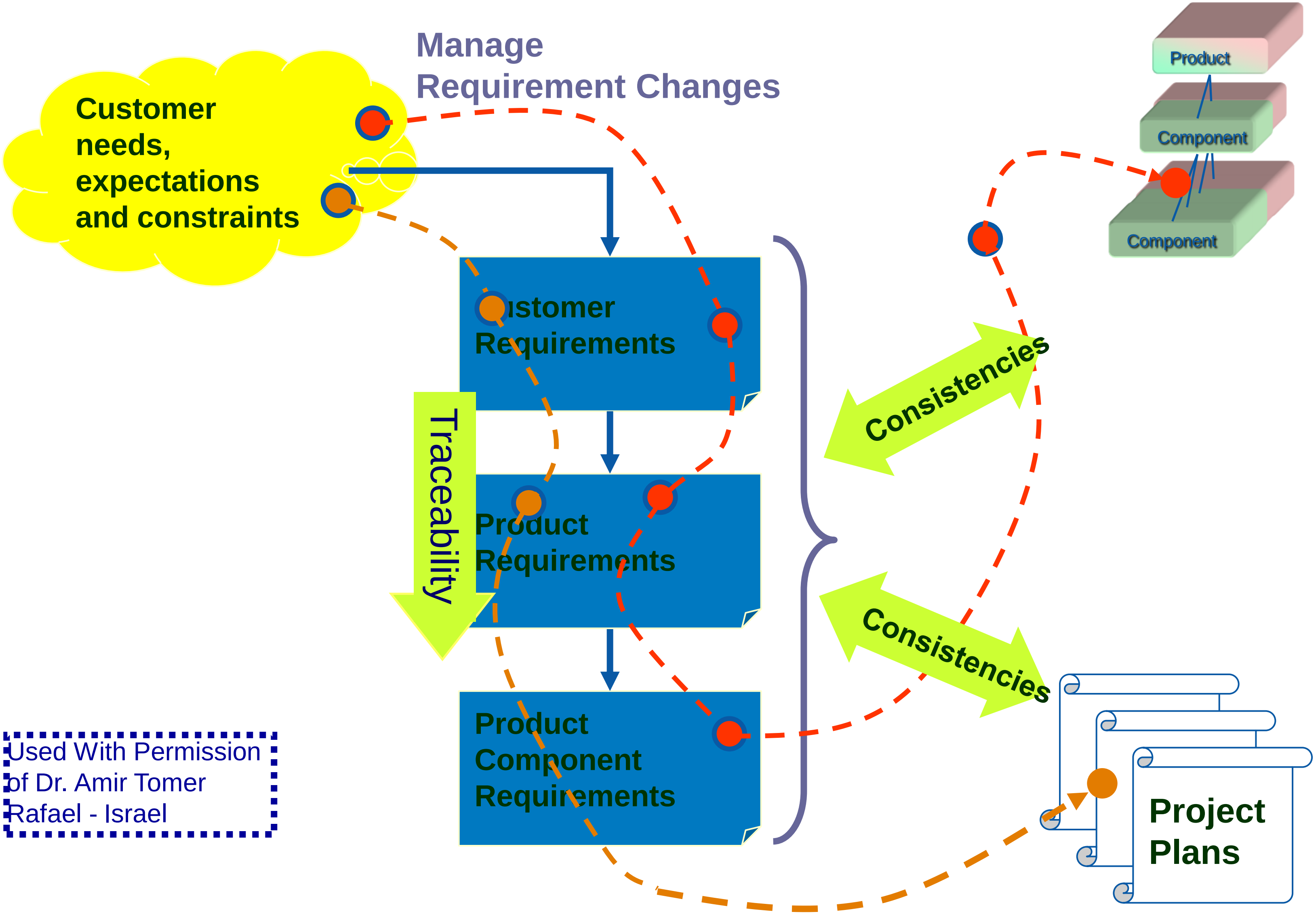
Gerald Kotonya and Ian Sommerville, Requirements Engineering, John Wiley and Sons, 1998

Bi-Directional Requirements Traceability



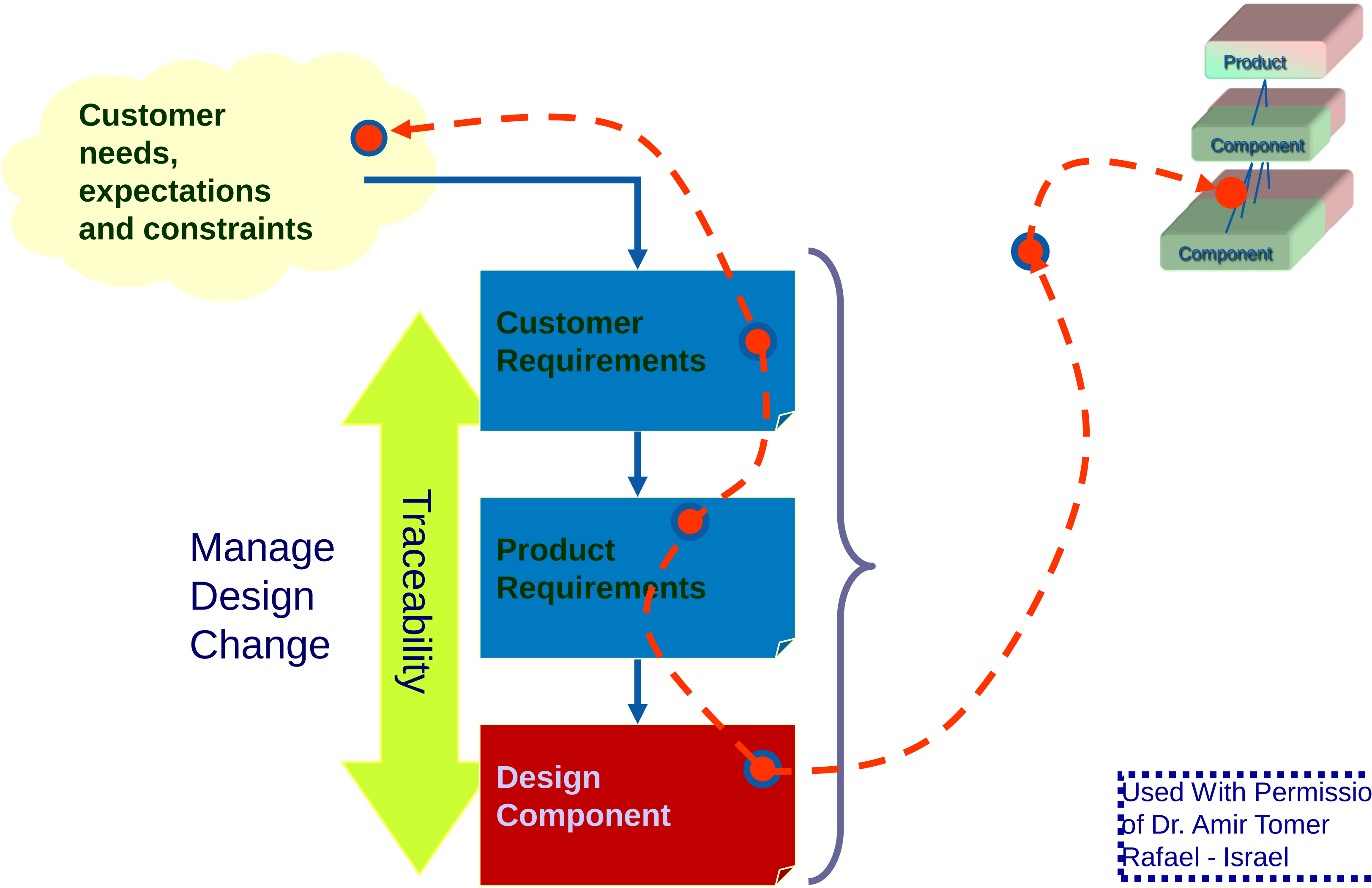
Requirement
Pointer To Related Requirement
Verification Criteria
Key Requirement
Pointer To Derived Requirement
Derived Requirement 1
Verification Criteria
Pointer To Derived Requirement
Pointer To Parent Requirement
Related Requirement
Key Requirement
Verification Criteria
Pointer To Related Requirement
Pointer To Derived Requirement
Derived Requirement 2
Pointer To Parent Requirement
Pointer To Derived Requirement

Requirement Traceability



Used With Permission
of Dr. Amir Tomer
Rafael - Israel

Bi-Directional Traceability



Customer needs, expectations and constraints

Customer Requirements

Product Requirements

Design Component

Traceability

Manage Design Change

Product

Component

Component

Used With Permission
of Dr. Amir Tomer
Rafael - Israel

Development of Software Safety Requirements

Development of Software Safety Requirements

Software safety requirements are obtained from various sources and are usually classified as generic or specific

Generic Software Safety Requirements – Generic software safety requirements are derived from sets of requirements and best practices used in different programs and environments to solve common software safety problems

- Capture lessons learned and provide a valuable resource for developers
- Prevent costly duplication of effort by taking advantage of existing proven techniques and lessons learned rather than reinventing techniques or repeating mistakes

Development of Software Safety Requirements - 2

Specific Software Safety Requirements – Specific software safety requirements are system-unique functional capabilities or constraints that are identified in the following ways:

- Method 1 – Through top-down analysis of system design requirements and specifications
- Method 2 – From the Preliminary Hazard Analysis (PHA)
- Method 3 – Through bottom-up analysis of design data (e.g., flow diagrams, Failure Mode Effects and Criticality Analysis (FMECA)

Fault and Failure Tolerance

A fault is an error that does not affect the functionality of the system

- Bad data
- Unknown command
- Command or data coming at an unknown time

A failure is the inability of a system or system component to perform its required functionality within specified performance requirements

Consequences of Failures and Faults

- A fault may or may not lead to a failure
- One or more faults can become a failure
- All failures are the result of one or more faults

Fault Tolerance - 2

Fault Tolerance (system failure) is the ability of the system to withstand an unwanted event and maintain a safe and operational condition

- It is determined by the number of faults that can occur in a system or subsystem without the occurrence of a failure

Software fault tolerance is usually concerned with detecting and recovering from small defects before they can become larger failures

- EXAMPLE: Error detection and handling is one example of fault-tolerant software coding practices

Hazardous Commands

A “Hazardous” command is one whose execution could lead to an identified critical or catastrophic hazard or a command whose execution can lead to reduced control of a hazard

Commands can be internal to a software set

- From one component to another
- External crossing an interface to/from hardware or a human operator
- Longer command paths increase the probability of an undesired or incorrect command response due to noise on the communication channel, link outages, equipment malfunctions or human error

Timing, Sizing, and Throughput Considerations

System design should consider real-world parameters and constraints including human operator and control system response times, and flow these down to software

- Adequate margins of capacity should be provided for all of these critical resources

Guidance for developers in specifying software requirements to meet safety objectives are provided here in overview fashion and discussed later in Software Safety Requirements Analysis

Timing, Sizing, and Throughput Considerations - 2

Time to Criticality – Safety-critical systems have a characteristic “time to criticality” which is the time interval between a fault occurring and the system reaching an unsafe state

- This interval represents a time window in which automatic or manual recovery and/or safing actions can be performed either by software, hardware, or by a human operator

Automatic Safing – Required if the time to criticality is shorter than the realistic human operator response time, or if there is no human in the loop

Timing, Sizing, and Throughput Considerations - 3

Control System Design – Control system design defines system timing requirements

- – Computerized control systems use sampled data (versus continuous data)

Sampling Rates – Sampling rates should be selected with consideration for noise levels and expected variations of control system and physical parameters

- – For measuring signals that are not critical, the sample rate should be at least twice the maximum expected signal frequency
- – For critical signals and parameters used for a closed loop control, the sampling rate is normally much higher

Timing, Sizing, and Throughput Considerations - 4

Dynamic Memory Allocation – Whether a dynamic allocation will fail or succeed depends on

- How much actual memory is available
- Whether virtual memory is to be used
- How much memory the software programs and the operating system use statically
- How much memory is intended to be dynamically allocated

How the software will deal with failed dynamic allocation should be specified

- Allowing a default like “abort, retry, fail” is a very bad idea for safety-critical software

Formal Inspections of Software Requirements

An Inspection is a formal verification technique in which life-cycle work products are examined in detail by a group of peers for the explicit purpose of detecting and identifying defects

The process is led by a Moderator or Facilitator or Inspection Leader who is not the author and is impartial to the life-cycle work products under review

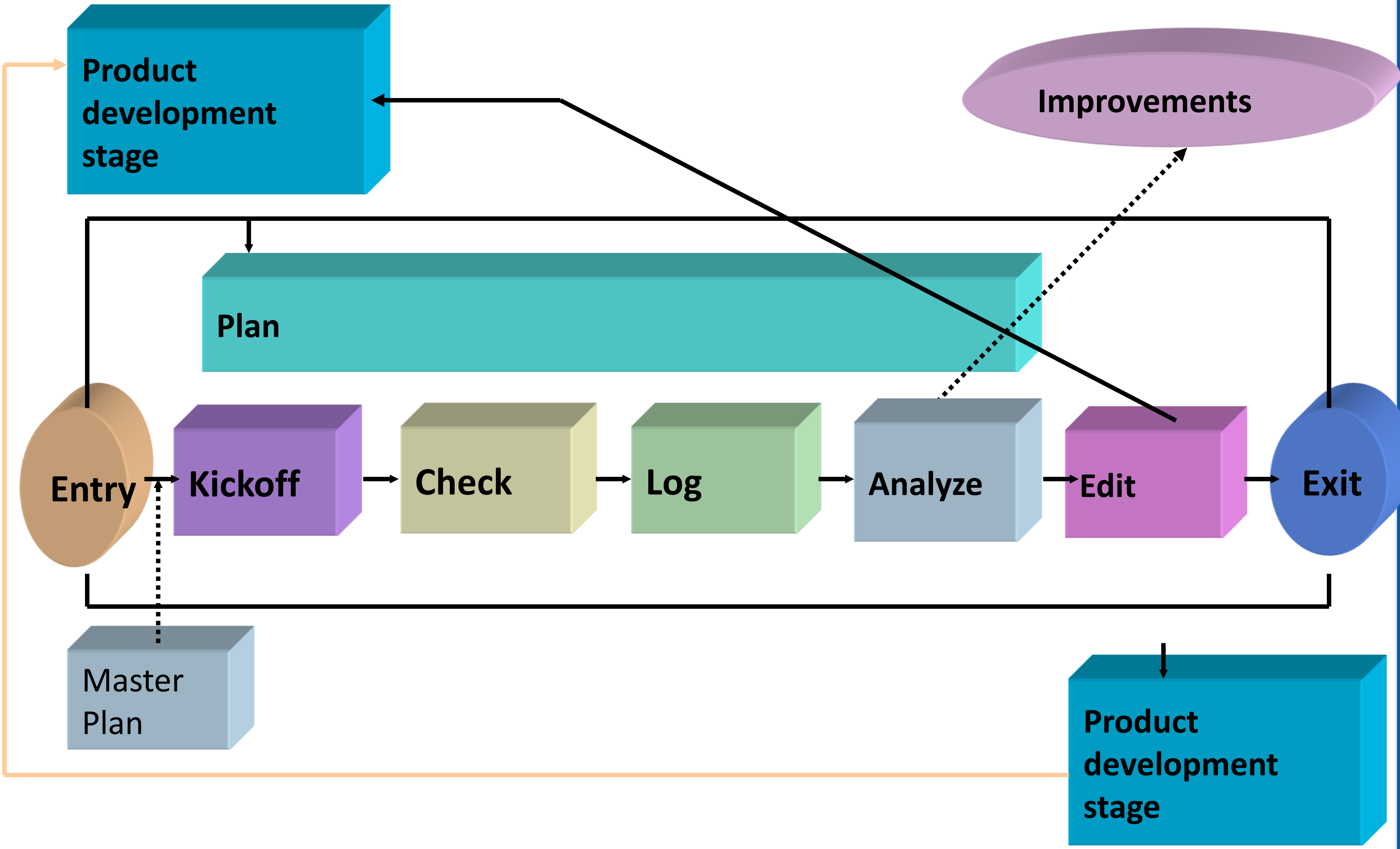
The author is not allowed to act as the Moderator

Defect resolution is mandatory and rework is formally verified

Defect data is systematically collected and stored in an inspection database

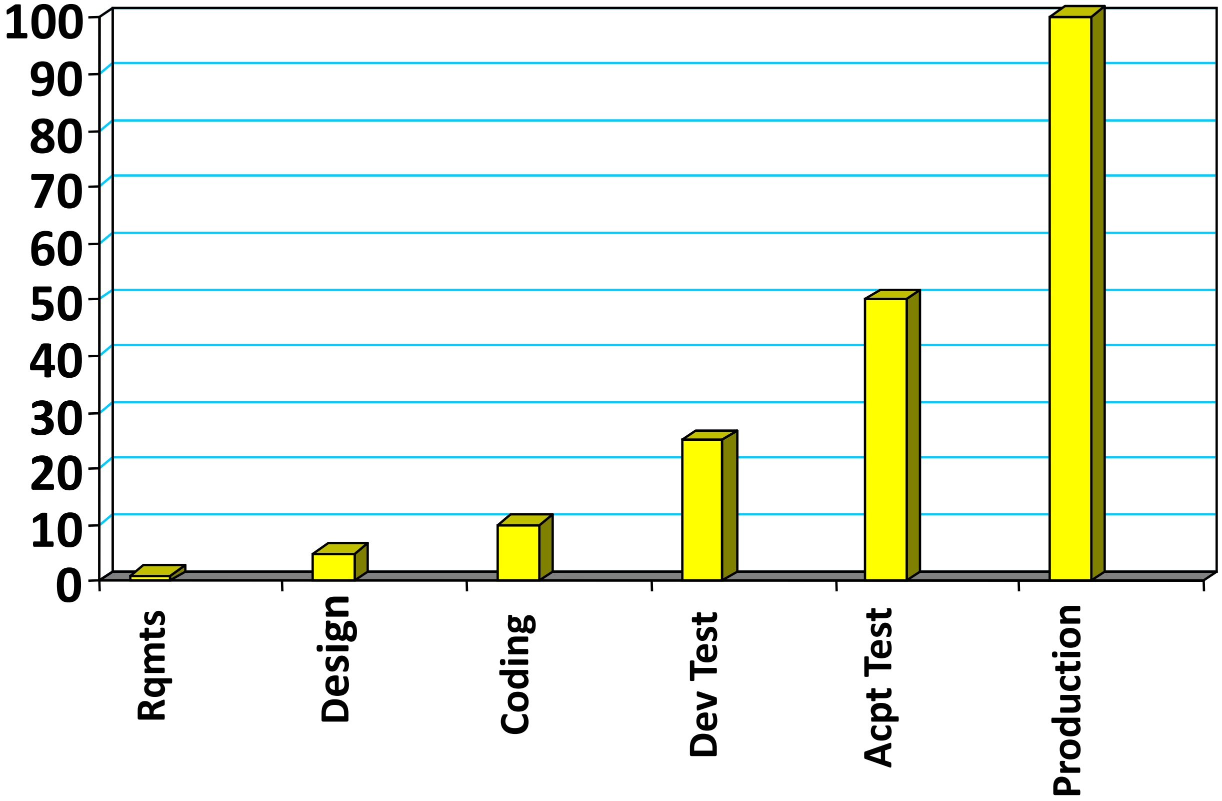
Data is analyzed to improve the product, the process and the effectiveness of the inspection

Steps in the Software Inspection Process



Why We Should Conduct Peer Reviews?

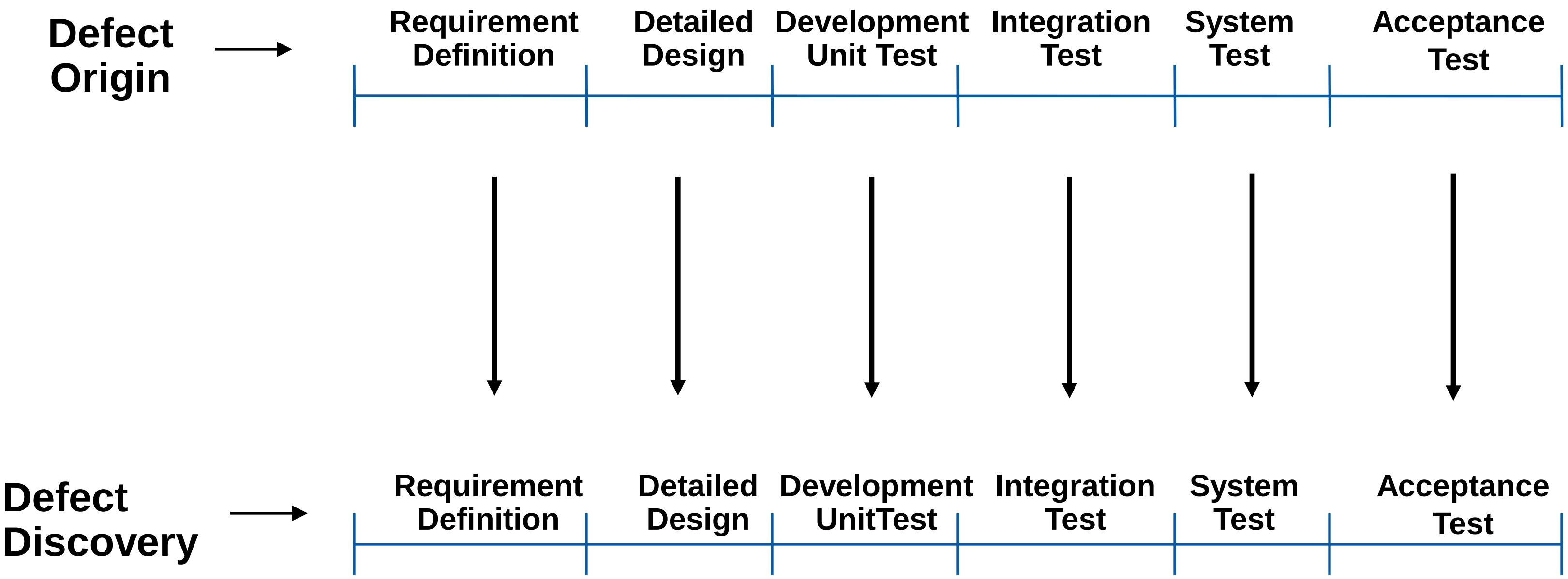
The sooner a defect is found, the cheaper it is to fix



1 **3-6X** **10X** **15-40X** **30-70X** **40-1000X**

Source: Boehm

What Is the Ultimate Objective?



Peer Reviews aim to minimize the number of defects being passed along from one segment to the next, by finding and fixing the defects in the segment in which they were created.

Test Planning

Safety tests of the system should also be designed at the time the systems tests are defined.

- System tests are designed to verify the functional aspects of the software under nominal conditions as well as performance, load, stress and other tests that verify acceptable behavior in non-standard situations
- Non-functional requirements such as the quality factors of reliability, maintainability, safety, security and usability should also be included in Systems Testing
- Safety tests should demonstrate how the software and system meets the safety requirements in the Software Requirements Document

Software Safety Requirements Analysis

Software Safety Requirements Analysis

The Requirements Analysis activities verify that safety requirements:

- Were properly flowed down from the system safety requirements
- Are correct, consistent, and complete
- Have covered all hazards, even new hazards such as software functions that can impact hazard controls and ways the software can behave that are unexpected
- These analysis activities are primarily top down analyses

Software Safety Requirements Analysis - 2

Bottom-up analysis of software requirements such as Requirements Criticality Analysis are performed to identify possible hazardous conditions

- Results in another iteration of the PHA or Software Subsystem Hazard Analysis
- May generate new software requirements

Software Safety Requirements Analysis - 3

Analyses related to the Software Requirements include:

- Software Safety Requirements Flow-down Analysis
- Requirements Criticality Analysis
- Specification Analysis
- Formal Methods
- Timing, Throughput and Sizing Analysis
- Preliminary Software Fault Tree Analysis
- Preliminary Software Failure Modes and Effects Analysis

Software Safety Requirements Flow-down Analysis

Generic safety requirements should be established “a priori” and placed into the system specification and overall project design specifications

- They then should be flowed into the subsystem specifications, such as the software subsystem requirements
- Top-down analysis

Other safety requirements are derived from bottom-up analysis

- They are flowed up from subsystems and components back to the system level requirements
- These new system level requirements are then flowed back down across all affected subsystems

Problems in the flow-down process can be caused by:

- Incomplete analysis
- Inconsistent analysis of highly complex systems
- Use of ad hoc techniques by inexperienced analysts

Requirements Criticality Analysis

Criticality analysis identifies program requirements that have safety implications

- Analyze the hazards of the software / hardware system and identify those that could present catastrophic or critical hazards
  - This evaluates each program requirement in terms of safety objectives derived for the software component
- For those requirements that have safety implications, designate the requirement as “safety-critical”
- Place the safety-critical requirement into a tracking system to ensure traceability of the software safety requirements throughout the software development cycle
  - From the highest level specification all the way to the code and test documentation

Requirements Criticality Analysis - 2

The System Safety organization should coordinate with the Project System Engineering organization to review and agree on the criticality designations

- Software Safety Engineers and Software Development Engineers should be included in the discussion
- The “software” viewpoint must be part of any systems engineering activity
- Requirements may be consolidated to reduce the number of critical requirements or flagged for special attention during design to reduce the criticality level
- It is always better to remove or reduce requirements later than to try to add them in when they did not exist before

Requirements Criticality Analysis - 3

Requirements Criticality Analysis Steps:

- Analyze all software requirements to identify additional potential system hazards that the system PHA did not reveal
- Use a checklist to identify PHA-designated hazards that are not reflected in the software requirements and new hazards missed by the PHA
- Look for areas where system requirements were not correctly flowed to the software
- Flow potential hazards added to the system requirements down to the subsystems (hardware, software and operations)
- Review the system requirements to identify hardware or software functions that receive, pass, or initiate critical signals or hazardous commands

Requirements Criticality Analysis - 4

- Review the software requirements to verify that the functions from the system requirements are included
- Look for any new software functions or objects that receive/pass/initiate critical signals or hazardous commands
- Look through the software requirements for conditions that may lead to unsafe situations
  - Out of sequence
  - Wrong event
  - Inappropriate magnitude
  - Incorrect polarity
  - Inadvertent command
  - Adverse environment
  - Deadlocking
  - Failure-to-command modes

Requirements Criticality Analysis - 5

Software safety analysis must also examine the characteristics of the software system

- All characteristics of safety-critical software must be evaluated to determine if they are safety-critical [See list of characteristics – next slide]
- Safety-critical characteristics should be controlled by requirements that receive rigorous quality control in conjunction with rigorous analysis and test

Requirements Criticality Analysis - 6

Safety-critical characteristics to be considered include:

- Specific limit ranges
- Out of sequence event protection requirements
- Timing
- Relationship to logic for limits (EXAMPLE: Temperature may be more constrained during an experiment run than when the system is idle)
- Voting logic
- Hazardous command processing requirements
- Fault response
- Fault detection, isolation, and recovery
- Redundancy management / switchover logic
  - What components to switch, and under what circumstances

Requirements Criticality Analysis - 7

Resources that serve as input to Requirements Criticality Analysis

- Software Development Plan
- Software Quality Assurance Plan
- Software Configuration Management Plan
- Risk Management Plan
- System Requirements
- System Design Document
- Software Requirements Specification
- External Interface Specification
- Functional Flow Diagrams
- Information from the system PHA on hazardous sources that may be controlled directly or indirectly by software

Specification Analysis

Specification analysis evaluates the completeness, correctness, consistency, and testability of software requirements.

- Well-defined requirements are strong standards by which to evaluate a software component

Specification analysis should evaluate requirements individually and as an integrated set.

Techniques to perform specification analysis include:

- Reading Analysis
- Traceability Analysis
- Control-flow Analysis
- Information-flow Analysis
- Functional Simulation

Specification Analysis - 2

The Safety Organization should ensure the software requirements appropriately influence the software design and the development of the operator, user, and diagnostic manuals.

The Safety Representative normally reviews:

- System Specification
- Software Requirements Specification
- Interface Requirements Specification
- Functional Flow Diagrams
- Storage allocation and program structure documents
- Information concerning system energy and other hazardous event sources that may be controlled directly or indirectly by software
- Software Development Plan, Software Configuration Management Plan, Software Quality Assurance Plan and Historical Data

Formal Methods

Formal methods represents the process of writing the requirements (specification) in a formal, mathematical language

- Even if Formal Verification is not used to verify the specifications, the act of creating a Formal Specification often catches many errors

Formal Specification removes ambiguity and uncertainty

- It allows errors or omission to be discovered including undocumented assumptions and inadequate off-nominal behavior
- Conflicting requirements and logic errors are also uncovered
- Defects found and corrected early in the lifecycle are much less costly to fix

Formal Methods - 2

“Model Checking” is a form of Formal Methods that verifies finite-state systems

- Model Checking is now used by many projects as an analysis and verification technique
- Model Checking is especially oriented at the verification of reactive, embedded systems or systems that are in constant interaction with the environment
- Model Checking can be easily applied at any stage of the existing software process without causing major disruptions

Timing, Throughput and Sizing Analysis

Timing, throughput, and sizing analysis for safety-critical functions evaluates software requirements that relate to execution time, I/O data rates and memory/storage allocations

- The analysis focuses on program constraints

Typical constraints include:

- Maximum execution time
- Maximum memory usage
- Maximum storage size for programs
- I/O data rates the program must support

The Safety Organization should evaluate the adequacy and feasibility of safety-critical timing, throughput, and sizing requirements

- I/O Channels may be overloaded by many error messages, preventing safety-critical features from operating

Timing, Throughput and Sizing Analysis - 2

Factors to consider for quantifying timing/sizing resource requirements:

- Memory Usage Versus Availability
  - Dynamic Memory Allocation – Can lead to problems from not freeing allocated memory, freeing memory twice or buffer overruns that overwrite code or other data areas
  - When data structures are dynamically allocated, they often cannot be statically analyzed to verify that arrays, strings, etc., do not go past the physical end of the structure

Timing, Throughput and Sizing Analysis - 3

- I/O Channel Usage (Load) Versus Capacity and Availability
  - Look at the amount of input data (science data, housekeeping data, control sensors) and the amount of output data (communications) generated
  - I/O Channel should include:
    - Internal hardware (sensors),
    - Interprocess communications (messages)
    - External communications (data output, command, and telemetry interfaces)

Timing, Throughput and Sizing Analysis - 4

- Execution Times Versus CPU Load and Availability
  - Investigate the time variations of CPU load and determine the circumstances that generate peak load
  - Is the execution under high load conditions acceptable?
  - Consider the timing effects from multitasking
    - Message passing delays
    - Inability to access a needed resources because another task has it
    - Excessive multitasking can result in a system instability leading to “crashes”

Timing, Throughput and Sizing Analysis - 5

- Sampling Rates Versus Rates of Change of Physical Parameters
  - Analysis should address the validity of the system performance models used, together with simulation and test data

- Program Storage Space Versus Executable Code Size
  - Estimate the size of the executable software in the device it is stored in
  - The program size includes the operating system as well as the application software

Timing, Throughput and Sizing Analysis - 6

- Amount of Data to Store Versus Available Capacity
  - Consider how much science, housekeeping, or other data will be generated and the amount of storage space available
  - Under some conditions, being unable to save data or overwriting previous data could be a safety related problem

Software Fault Tree Analysis

It is possible for a system to meet requirements for a correct state and still be unsafe

- Complex systems increase the chance that unsafe modes will not be caught until the system is in the field
- Fault Tree Analysis (FTA) is one method that focuses on how errors can lead to hazards
- Software Fault Tree Analysis (SFTA) is an extension of the hardware FTA into the software arena
- The requirements phase is the time to perform a preliminary SFTA
  - The Preliminary Hazard Analysis or Software Subsystem Hazard Analysis is the primary source for hazards along with the Requirements Criticality Analysis
- The result of a fault tree analysis is a list of contributing causes (e.g., states or events) that can lead to a hazard

Software Fault Tree Analysis - 2

Software Fault Tree Analysis (SFTA) works well in combination with the Software Failure Modes and Effects Analysis (SFMEA)

- The SFTA is top-down working from the hazard to possible causes
- The SFMEA starts with individual components and works from the bottom (component) up to a hazard
- When used together, these two analyses are very good at finding all of the possible failure modes or areas of concern

Software Failure Modes and Effects Analysis

Failure Modes and Effects Analysis is a bottom-up technique that looks at:

- How each component could fail
- How the failure propagates through the system
- Whether it can lead to a hazard or not
- A detailed design is required for an effective FMEA to be conducted

Software Failure Modes and Effects Analysis (SFMEA) uses the methods of a standard (hardware) FMEA, substituting software components for hardware components in each case

Software Failure Modes and Effects Analysis - 2

There are several types of FMEAs:

- System – focuses on global system functions
- Design – focuses on components and subsystems
- Process – focuses on manufacturing and assembly processes
- Service – focuses on service functions
- Software – focuses on software functions

Software Failure Modes and Effects Analysis - 3

Benefits of FMEA

- Provide the engineer with a tool that can assist in providing reliable, safe, and customer pleasing products and processes
- Improve product/process reliability and quality
- Increase customer satisfaction
- Early identification and elimination of potential product/process failure modes
- Prioritize product/process deficiencies
- Capture engineering/organization knowledge
- Emphasizes problem prevention
- Documents risk and actions taken to reduce risk
- Provide focus for improved testing and development
- Minimizes late changes and associated cost
- Catalyst for teamwork and idea exchange between functions

Software Failure Modes and Effects Analysis - 4

FMEA Procedure – Overview

- 1. Describe the product/process and its function - It is important to consider both intentional and unintentional uses since product failure often ends in litigation, which can be costly and time consuming.
- 2. Create a Block Diagram of the product or process - The diagram shows the logical relationships of components and establishes a structure around which the FMEA can be developed
- 3. Complete the header on the FMEA Form worksheet. (See Example)
- 4. Use the worksheet to begin listing items or functions

EXAMPLE FMEA - Spreadsheet

Part/Product Name and No.:
Process Name:
Other Areas Involved:

Part/Product Owner:
Process Owner:

Prepared By:
FMEA Date (Orig.): (Rev.):

Item/ Function	Potential Failure Mode	Potential Failure Effects	SEV	Potential Causes	OCC	Current Design Controls	DET	RPN	Recommended Actions	Actions or Plans	pS	pO	pD	pRPN

Software Failure Modes and Effects Analysis - 5

- 5. Identify Failure Modes – A failure mode is defined as the manner in which a component, subsystem, system, process, etc. could potentially fail to meet the design intent. Examples of potential failures modes include:
  - Corrosion
  - Hydrogen embrittlement
  - Electrical Short or Open
  - Torque Fatigue
  - Deformation
  - Cracking
- 6. List failure modes for function of each component or process step – a failure mode in one component can serve as the cause of a failure mode in another component

Software Failure Modes and Effects Analysis - 6

- 7. Describe the effects of those failure modes – A failure effect is defined as the result of a failure mode on the function of the product/process as perceived by the customer. Examples of failure effects include:
  - Injury to the user
  - Inoperability of the product or process
  - Improper appearance of the product or process
  - Odors
  - Degraded performance
  - Noise

Software Failure Modes and Effects Analysis - 7

- 8. Identify the causes for each failure mode – A failure cause is defined as a design weakness that may result in a failure. Examples of potential causes include:
  - Improper torque applied
  - Improper operating conditions
  - Contamination
  - Erroneous algorithms
  - Improper alignment
  - Excessive loading
  - Excessive voltage

Software Failure Modes and Effects Analysis - 8

- 9. Enter the Probability Factor – A numerical weight should be assigned to each cause that indicates how likely that cause is to occur

- 10. Identify Current Controls (design or process) – Current controls (design or process) are the mechanisms that prevent the cause of the failure mode from occurring or which detect the failure before it reaches the customer.
  - Controls should be assessed to determine how well it is expected to identify or detect failure modes
  - Previously undetected or unidentified failure modes should be used to update the FMEA

Software Failure Modes and Effects Analysis - 9

- 11. Determine the likelihood of Detection – Detection is an assessment of the likelihood that the Current Controls will detect the Cause of the Failure Mode or the Failure Mode itself thus preventing it from reaching the Customer

- 12. Review Risk Priority Numbers – The Risk Priority Number is a mathematical product – RPN = (Severity) x (Probability) x (Detection)

Software Failure Modes and Effects Analysis - 10

- 13. Determine Recommended Actions to address potential failures that have a high RPN – These actions could include:
  - Specific inspection, testing, or quality procedures
  - Selection of different components or materials
  - Limiting environmental stresses or operating range
  - Re-design of the item to avoid the failure mode
  - Performing preventative maintenance
  - Inclusion of back-up systems or redundancy

- 14. Assign Responsibility and a Target Completion Date for these actions – Make responsibility clear cut!

Software Failure Modes and Effects Analysis - 11

- 15. Indicate Actions Taken – After the actions have been taken, re-assess the severity, probability and detection and review the revised RPNs
  - Are further actions needed?

- 16. Update the FMEA as the design or process changes

Software Requirements Tasks

Software Engineering Tasks	System and Software Safety Tasks
Software Requirements Development	Development of software safety-critical requirements
Requirements Management	Safety Requirements Flow-down Analysis
Formal Methods for Specification	Requirements Criticality Analysis
Formal Inspection of Software Requirements	Specification Analysis of Safety-Critical Requirements
System Test Planning	Software Fault Tree Analysis
Timing, Throughput and Sizing Considerations	Software Failure Modes and Effects Analysis Formal Inspections of Software Requirements

Software Requirements Documentation

Document	Software Safety Section
Software Requirements Document	Identification of all safety-critical software requirements
Software Interface Specification	Identification of any interfaces that are part of safety-critical elements
Formal Inspection of Software Requirements	Identification of any safety-critical requirements defects that are considered major (must be fixed)
Requirements Traceability Matrix	Special identification given to safety-critical requirements
Analysis Reports	Identification of any safety-related aspects or safety concerns
Acceptance Test Plan	This is the customer acceptance test. Includes all safety testing necessary to assure the customers that the system is safe
Systems Test Plan	Includes stress, load, disaster, stability, and other tests including functional testing. Verifies that the system cannot go into an unsafe mode under adverse conditions

Software Configuration Management and Software Quality Assurance

Software Configuration Management

Software Configuration Management (SCM) is a critical Project Management support function

- One cannot product “safe” software without software configuration management
- The system or system component cannot be declared “quality” if there is no control over the pieces that have been developed
- SCM is a process to maintain and monitor the software development process

Software Configuration Management - 2

Software Configuration Management includes:

- Identification
  - Identifying the structure and kinds of components
  - Making them unique and accessible in some form by giving each component a name, version identification and configuration identification
- Control
  - Controlling the release of a product and changes to it throughout the project/product lifecycle by having controls in place that ensure consistent software via the creation of a baseline product

Software Configuration Management - 3

- Status Accounting
  - Recording and reporting the status of components and change requests
  - Gathering vital statistics about components in the product
- Configuration Auditing
  - Validating the completeness of a product
  - Maintaining consistency among the components by ensuring that the components are reviewed for possible change when the requirements they are associated with are changed

Software Configuration Management - 4

- ***Change Control***
  - Change control is an important part of developing safe software
  - Arbitrary changes should be avoided
  - Once a piece of software has reached a level of maturity, any subsequent change request should go through a formal change control process
  - Changes to the lifecycle work products, especially the code should be accompanied by comments that indicate
    - Why the change occurred
    - What was changed
    - When it was changed
    - Who changed it

Software Configuration Management - 5

- ***Defect Tracking***
  - Record all the defects for future reference
  - Having defect information from previous projects can be a big plus when debugging the next project
  - Recording the defects allows metrics to be determined
    - One of the easiest ways to judge whether a program is ready for serious safety testing is to measure its defect density, the number of defects per line of code

Software Quality Assurance

Software Quality Assurance should begin in the early phases of a project to establish plans, processes, standards, and procedures that will:

- Add value to the project
- Satisfy the requirements of the project and organizational policies

SQA Representatives should participate in the establishment of those plans, processes, standards, and procedures to ensure they will fit or can be tailored to fit the project's needs and that they will be usable for performing quality assurance evaluations

SQA Representatives provides project staff and managers at all levels with appropriate visibility into, and feedback on, processes and associated work products throughout the life of the project

Software Quality Control vs. Software Quality Assurance

Quality Control evaluates the products

- Checks to see if the requirements for the product or product components are satisfied (Verification)
- Checks the quality of the product(s)
  - Is the product within tolerance?
  - Is the product (or life-cycle work product) of an acceptable quality?
- Tools and Techniques
  - reviews
  - tests
  - inspections

Software Quality Control vs. Software Quality Assurance - 2

Quality Assurance evaluates the process

- Checks that the process is working
  - Is the process being followed?
  - Are the QC checks being applied?
  - Are the QC checks efficient?
  - Is the process causing quality problems?
  - Is the process working for the organization?
- Tools and Techniques
  - Audits / Objective Evaluations
  - Assessments / Appraisals

Software Quality Reports to Management

Software Quality Reporting should include comments about the processes, standards, etc., to the following questions:

- Are they efficient?
- Are they effective?
- Does following these processes result in the necessary product quality?
- If Software Safety is required, the quality report must indicate if the processes are being followed and are adequate to produce “safe software”

Quality reports are sent to Higher-level Management for their analysis and action

Safety as a Quality Factor

Concepts Taken from “Common Concepts Underlying Safety, Security, and Survivability Engineering”

Donald G. Firesmith

December 2003

CMU/SEI-2003-TN-033

Analysis of Software Intensive Systems

Software-intensive systems are commonplace, and society relies heavily upon them

- Software is found in automobiles, airplanes, chemical factories, power stations and other mission critical systems
- Software-intensive systems are neither perfect nor invulnerable
  - They fail due to software defects, hardware breakdowns, accidental misuse and deliberate abuse

Analysis of Software Intensive Systems - 2

Safety, security, and survivability engineering are three very close related disciplines

- Analysis of all three is inherently difficult – these requirements specify what is to be prevented such as attacks and safety hazards rather than a desired capability
- They deal with assets that must be protected and with the risks of harm to these assets that must be managed
- There is an inherent level of uncertainty because what these requirements seek to prevent may or may not ever happen!
- This is especially true of safety requirements!
- Hazards and threats associated with software-intensive systems are also constantly changing, making the risks very difficult to quantify

Analysis of Software Intensive Systems - 3

Quality means much more than merely meeting functional requirements

- An application that meets all of its required features and fulfills each of its use cases can still be totally unacceptable
  - Inadequate availability
  - Capacity too low
  - Performance too slow
  - Not interoperable with other systems
  - Not safe to use
  - Numerous security vulnerabilities
  - Not considered to be user friendly

It is not adequate to specify required quality by stating that an application shall have high capacity and be safe, secure and usable

Analysis of Software Intensive Systems - 4

Quality Model

- A quality model decomposes quality into its component quality factors
  - Attributes
  - Subfactors
- It provides specific quality criteria (descriptions)
- It provides metrics or means of measurement
  - Turns general high-level quality factors into detailed and specific measureable descriptions
- It describes the capabilities of a systems beyond its functionality

Analysis of Software Intensive Systems - 5

Quality Model – a hierarchical model for formalizing the concept of quality in terms of its component quality factors and subfactors

Quality Factor (Quality Attribute) – a high-level characteristic or attribute of something that captures an aspect of its quality

- Quality has to do with the degree to which something possesses a combination of characteristics, attributes, aspects, or traits that are desirable to its stakeholders
- Quality Factors include availability, expandability, maintainability, reliability, reusability, safety, security and usability

Analysis of Software Intensive Systems - 6

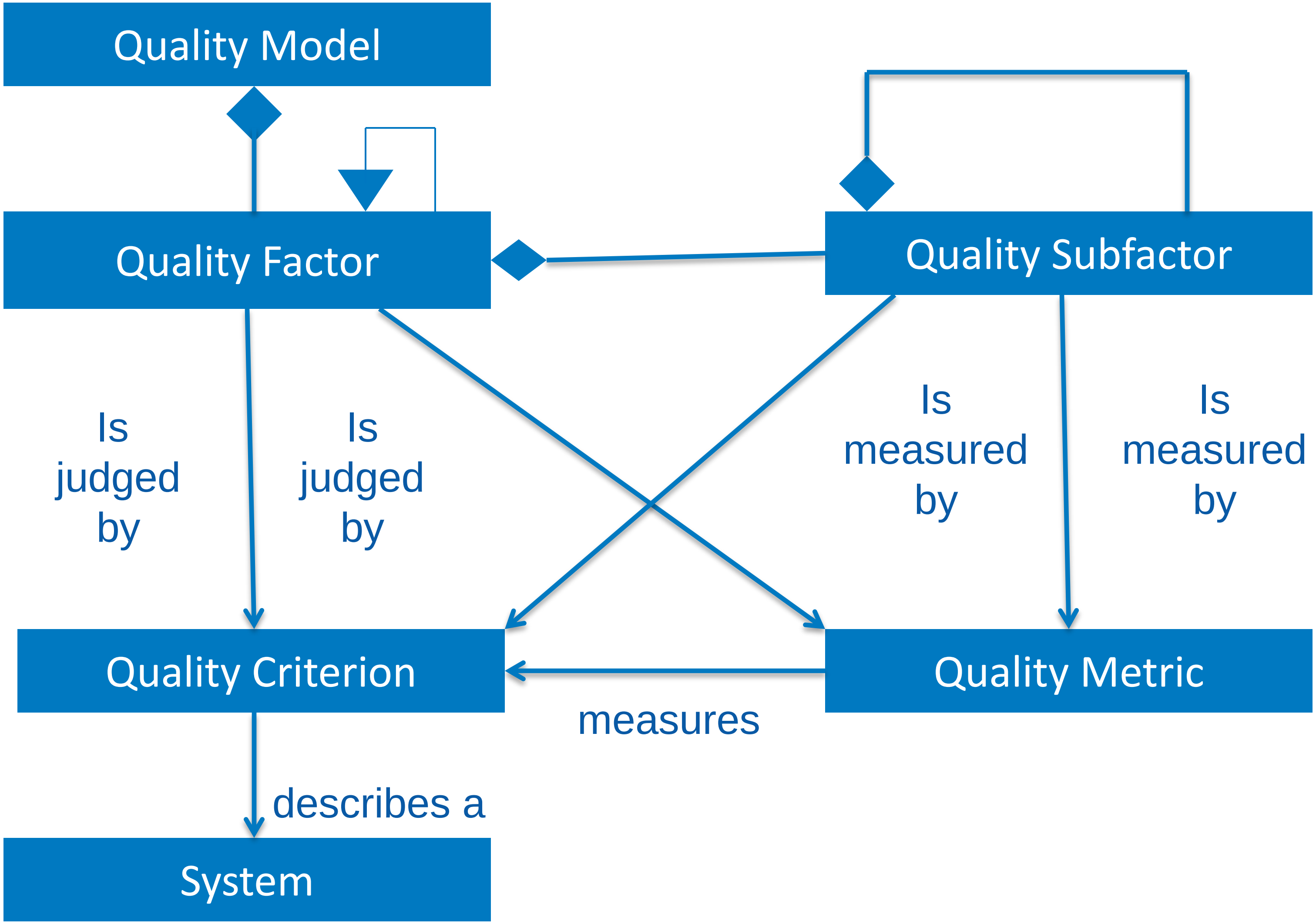
Quality Subfactor – a major component (aggregation) of a quality factor or another quality subfactor

Quality Criterion – a specific description of something that provides evidence either for or against the existence of a specific quality factor or subfactor

Quality Metric – a metric that quantifies a quality criterion and thus makes it measurable, objective, and unambiguous

System – an integrated collection of data components, hardware components, software components, human-role components and document components that collaborate to provide some cohesive set of functionality with specific levels of quality

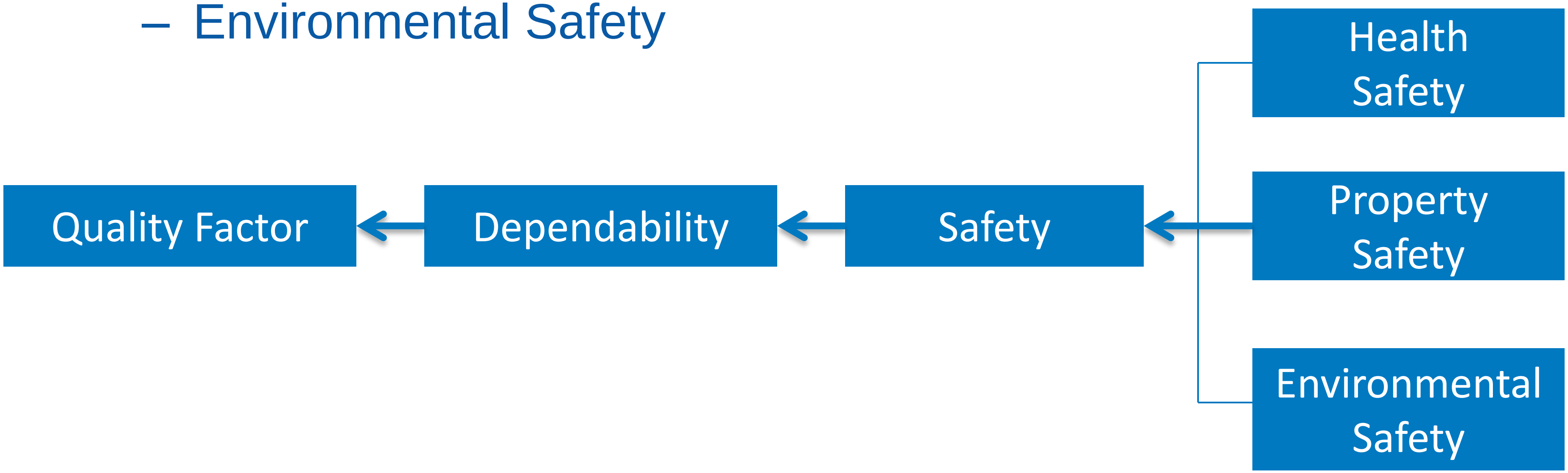
Model for a Quality Model



Safety as a Quality Factor

Safety – as a Quality Factor, safety can be classified into subclasses:

- Health Safety
- Property Safety
- Environmental Safety



Safety as a Quality Factor - 2

Safety – the degree to which accidental harm is prevented, detected, and properly reacted to

- People are not considered safe unless they are safe from both accidental and malicious harm

Health Safety – the degree to which illness, injury, and death are prevented, detected, and properly reacted to

- Health safety involves all people who may reasonably be expected to be harmed by the system during an accident

Property Safety – the degree to which accidental damage and destruction or property is prevented, detected, and properly reacted to

Environmental Safety – the degree to which accidental damage to and destruction of parts of the environment is prevented, detected, and properly reacted to

Tailoring the Software Safety Effort

Once the scope of the software safety effort has been determined, the project or program processes can be tailored

The safety activities should be sufficient to match the software development effort and yet ensure that the overall system will be safe!

Tailoring the Software Safety Effort - 2

Development	The use of safety features such as firewalls depends on where it is best applied and needed. The degree to which each of these activities is performed is related to the software risk. Software Safety features should be reflected in the requirements
Analysis	There are many types of analyses that can be completed during software development. The analyses can range from Requirements Criticality Analysis to Software Fault Tree Analysis of the design to Formal Methods

Tailoring the Software Safety Effort - 3

Inspections	Inspections can take place in a number of settings and with varying products (requirements to test plans). The number of inspections and products is dependent on the risk related to the system. Inspections is a form of Peer Reviews that can include Structured Walkthroughs, Walkthroughs, Circulation Reviews, and Buddy Checks. Technical Reviews that involves a group of complementary disciplines to seek alternative solutions is a Review but not normally considered a Peer Review.
Reviews	The number of formal reviews and the setting up of delta or follow-up reviews can be used to give the organization more places to look at the products as they are being developed.

Tailoring the Software Safety Effort - 4

Verification Activities	Verification includes verification of the product and intermediate work products against selected requirements, including customer, product, and product component requirements. Verification is inherently an incremental process. It begins with verification of the requirements and progresses through the verification of the evolving work products. Verification culminates in the verification of the completed product normally in Systems Test although sometimes Acceptance Testing serves as the final Verification activity. Verification methods should address re-verification to ensure that rework performed on work products do not cause unintended defects. Methods of verification include but are not limited to: • Inspections and Structured Walkthroughs • Audits • Path coverage testing • Simulations • Modeling • Functional testing • Decision table-based testing • Load, stress, and performance testing • Operational scenario testing • Observations and demonstrations Verification and Validation environments may be built, reused, purchased, or shared or some combination of these.

Tailoring the Software Safety Effort - 5

Validation	Validation demonstrates that a product or product component fulfills its intended use when placed in its intended environment (operational environment) by the intended end users who must use the system in that operational environment. Validation methods are selected based on their ability to demonstrate that user needs are satisfied. To support validation, the requirements must be defined for a realistic environment including: • Functionality (implementation) • Maintenance • Sustainment • Reuse • Installation • Training • Support • Performance • Quality Attributes • Disposal as appropriate

Verification and Validation Tailoring

Verification and Validation activities may be tailored by determining the types of testing based on the software risk, complexity and criticality of the software product or product component

Unit Testing may be performed by the software developer, by a peer developer or by an outside supplier depending on those factors, the mission and the expectations of the customer as well as the end users and the operational environment the product must operate in

Maintenance and expansion constraints or requirements, support, training, usability and reliability requirements all may have an effect on the tailoring of the Verification and Validation activities

"Full" Software Safety Effort

Systems and subsystems that have severe hazards which can escalate to major failures in a very short period of time require the greatest level of software safety effort

- These systems may require a formal, rigorous program of quality and safety assurance to ensure complete coverage and analysis of all requirements, design, code, and tests
- Safety analyses, software development analyses, safety design features, and Software Quality Assurance oversight are highly recommended

"Moderate" Software Safety Effort

Systems and subsystems which fall into this category typically have either

- 1)A limited hazard potential
- 2)The response time for initiating hazard controls to prevent failures is long enough to allow for human operators to respond to the hazardous situation

These systems require a rigorous program for safety assurance of software identified as safety-critical

- Some analyses are required to assure there are no "undiscovered" safety-critical areas that may need software safety features

“Minimum” Software Safety Effort

For systems in this category, either the inherent hazard potential of a system is very low or control of the hazard is accomplished by non-software means

- Software development in these types of systems must be monitored on a regular basis to ensure that safety is not inadvertently compromised or that features and functions are added which make the software safety-critical

Projects With Mixed Levels of Effort

Not all projects fall into neat categories for classification

- Some projects may be large and complex, but with a small portion of safety-critical, low risk software
- Other projects may be small, but a significant portion of software is safety-critical and high risk

Too often smaller projects argue that they have no safety-critical software out of concern that the label will lead to massive amounts of work

- Partitioning of the safety-critical software from code that is not safety-critical allows the safety effort to be applied to the appropriate safety-critical portion

Summary

Creating a safe system is a team effort and safety is everyone’s responsibility

Software is a vital part of most systems

Software, by itself cannot injure you, but software normally operates in an electronic system and often controls other hardware

Software is hazardous if it can directly lead to a hazard or is used to control a hazard

- Hazardous software includes all software that is a hazard cause
- Is a hazard control
- Provides information upon which safety-critical decisions are made
- Is used as a means of failure/ fault detection

Tim Kasse – Contact Information

Tim Kasse

QMS&A
Program Manager – Software Quality Assurance Program
National Renewable Energy Laboratory (NREL)
1617 Cole Blvd.
Golden, Colorado 80401
(303) 275 - 3285