NASA/TM–2015-218764

Overview of Design, Lifecycle, and Safety for Computer-Based Systems

Wilfredo Torres-Pomales
Langley Research Center, Hampton, Virginia

May 2015

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM–2015-218764

Overview of Design, Lifecycle, and Safety for Computer-Based Systems

Wilfredo Torres-Pomales
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

May 2015

Acknowledgment

The author would like to express his gratitude to Alwyn Goodloe, Paul Miner, and Kurt Woodham for their support, insight, and comments during the development of this document.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

This document describes the need and justification for the development of a design guide for safety-relevant computer-based systems. This document also makes a contribution toward the design guide by presenting an overview of computer-based systems design, lifecycle, and safety.

Table of Contents

Abbreviations

ConOps	Concept of Operations
COTS	Commercial Off The Shelf
DIMA	Distributed Integrated Modular Architecture
DMA	Direct Memory Access
DPS	Data Processing System
FAA	Federal Aviation Administration
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IEEE	Institute of Electrical and Electronics Engineers
IKIWISI	I'll-know-it-when-I-see-it
IMA	Integrated Modular Avionics
IoP	Index of Performance
LRU	Line Replaceable Unit
MIL-STD	Military Standard
NAS	National Airspace
NASA	National Aeronautics and Space Administration
SOI	System Of Interest
SS	Sub-System
TRL	Technology Readiness Level
V&V	Validation and Verification

1. Introduction

An aircraft consists of a collection of systems performing a wide variety of functions with different safety criticality levels. The aviation industry is experiencing an ongoing, decades-old trend of adopting increasingly sophisticated computer-based technology to implement aircraft functionality. Modern aircraft are highly complex, functionally integrated, network-centric systems of systems [1]. The design and analysis of distributed-computation systems like the ones used on aircraft are inherently complex activities. Ensuring that such systems are safe and comply with existing airworthiness regulations is costly and time-consuming as the level of rigor in the development process, especially the validation and verification activities, is determined by considerations of system complexity and safety criticality. A significant degree of care and deep insight into the operational principles of these systems are required to ensure adequate coverage of all design implications relevant to system safety.

1.1. Background

Aircraft have used digital and software-based electronics since the 1960s [2]. The first avionics architectures were in the form of custom-made line replaceable units (LRU) with each LRU performing a single function. These architectures allocated dedicated physical resources (computing processors, communication, input sensors and output effectors) to each function, and the LRUs were either completely independent of each other or exchanged limited amounts of data [3]. Later, so-called federated architectures retained the allocation of one function per LRU but showed a higher level of integration mostly by increasing the data exchange between functions through dedicated one-way data links or shared communication data buses [4]. Beginning in the 1990s, Integrated Modular Avionics (IMA) architectures were introduced to address market forces that drove the need to reduce the volume, weight, power and maintenance cost of avionics systems [5], [6], [7], [8], [9]. The fundamental characteristic of IMA architectures is the sharing of resources between functions on a distributed computational platform. The initial examples of the IMA concept consisted of centralized cabinets with multiple simple modules (or even just electronic cards) performing specific tasks like processing, networking, data storage and input-output from sensors and effectors [10]. These resources are shared among various airplane functions that run on the platform as time-sharing processes. These architectures used local backplane buses for communication between modules in the same cabinet and single-source data links or shared buses for communication cabinet-to-cabinet or between cabinets and separate LRUs. Thus, at a global level, the communication network is segmented and uses gateways for transferring data between segments. More recent IMA architectures connect the modules to a common switch-based data network that handles the routing of data messages to their intended destinations [6], [9], [11]. This architectural model can be characterized as a distributed IMA (DIMA) with a high degree of functional integration on a platform of physically distributed computation and input-output modules (or remote data concentrators), and may include federated LRUs on the same network [12].

The main drivers for the evolution of avionics architectures are the competition among airlines in the air travel market and the competition among airplane manufacturers to meet the demands from the airlines for more fuel-efficient and cost-effective airplanes that have more functionality and higher sophistication [5], [6], [7], [13]. Over time, functionality has been added to improve airplane flight performance and safety, as well as to improve maintenance and passenger comfort [11]. The greatest enabler to the evolution of avionics architectures has been advancements in electronics and computer technology, including microprocessors, operating systems, data networking, sensors, displays and design development tools [14]. As technology has improved, the cost of electronic hardware components has decreased, but so has their lifecycle duration. Simultaneously, the ever-increasing functional complexity is being implemented mostly

in software, prompting greater interest in ways to simplify software development. System cost and considerations of hardware part obsolescence and software reuse have driven system developers toward layered and modular designs with standardized interfaces and generic hardware and software commercial-off-the-shelf (COTS) components where practicable. As functionality has increased, software development and system integration have become the primary cost factors.

In addition to functional requirements, aircraft avionics systems must satisfy demanding quality requirements for performance, dependability and safety under stated operational and environmental conditions [15], [16], [17], [18], [19]. A system is safety-relevant if its failure can endanger human life, property or the environment [20], [21]. The main goal behind system safety requirements is to ensure an acceptable and rational inverse relation between the probability and severity of functional failures [22]. A system can experience internal perturbations due to logical faults (i.e., defects) from design errors introduced during system development, or physical faults introduced during system development or operation [23]. The basic approach for dealing with these threats to the quality of the services delivered by a system consists of fault prevention and removal, as well as the use of redundancy supported by fault and error containment techniques (e.g., isolation, separation, partitioning, dissimilarity, selection, and voting) to mitigate the propagation of fault effects [22], [23], [24], [25]. Older federated avionics systems had logical dependencies that were simple to manage and physical barriers that prevented the propagation of effects between architecture-level components, but modern functionally integrated architectures with more resource sharing and intricate data exchange patterns have a higher risk of unintended interactions between components. Most of the complexity in modern aircraft systems stems from these requirements for high functional quality while protecting against potential failures due to physical or logical defects, or misuse [1].

A system is said to be complex when its operation, failure modes or failure effects are difficult to comprehend without the aid of analytical methods [25]. The complexity of some systems is such that they cannot be analyzed and understood well enough to be managed effectively by any one individual or small groups. System complexity has many aspects, including task structure (i.e., the sequencing and timing of actions to achieve a goal), predictability (i.e., randomness in the effects of system actions), size (i.e., the number of components and functions), and algorithmic complexity (including space and time requirements of computations and cognitive complexity) [26], [27]. Distributed safety-critical systems require complex algorithms for achieving and preserving distributed coordination and consistency even when operating with faulty components [28]. In general, complex interactions between components (including both hardware and software components) have a higher potential for execution errors. This threat can be aggravated by coupling between components that allows the propagation of fault effects along paths of data and control information flow. With complex integrated systems, there is also the possibility of unintended coupling through shared resources and the logical and physical environment in which the system operates.

Uncertainty about the interactions and coupling between components in complex computer-based aircraft systems, especially under failure conditions, is a recognized point of concern for certification authorities because of the possible safety implications [29]. It is known that using testing and statistical techniques are not feasible approaches to quantify the reliability of software at the levels required for safety-critical applications [30]. Likewise, it may not be possible to develop a test suite for complex computer-based systems to demonstrate the absence of requirements and design errors [24]. Because of the present inability to ascertain the completeness and correctness of the requirements and design of complex systems, the current state of practice relies heavily on assessments of the process used to develop the system to achieve an adequate confidence that errors have been identified and corrected [25], [31], [32], [33]. However, industry, academia and certification authorities recognize that development process assurance

alone may not be sufficient to establish that safety objectives are adequately satisfied [30], [24]. Safety engineering and evidence-based approaches have been proposed and are being investigated as a way to increase confidence in the safety of complex computer-based systems [30], [34], [35].

The validation and verification (V&V) and the certification of complex computer-based systems, including safety-critical systems, are recognized problems of national significance [31], [36], [37]. The challenges in assuring the design and safety of systems need considerable attention and financial investment [38]. As the complexity of systems continues to increase, the V&V and certification costs and related programmatic risks can provide a basis against the development and implementation of new capabilities [39]. Such obstacles against innovation pose a threat to national competitiveness and can hinder the proposed operational improvements to the National Airspace System (NAS) that are intended to increase capacity and flexibility and reduce costs, but would also increase the complexity of airborne and ground aviation systems [40]. There are initiatives underway to produce methods, tools, and techniques that enable predictable, timely and cost-effective complex systems development [41]. NASA aims to identify risks and provide knowledge to safely manage the increasing complexity in the design and operation of vehicles in the air transportation system. For this, multidisciplinary tools and techniques are being developed to assess and ensure safety in complex aviation systems and enable needed improvements to the NAS.

1.2. Purpose

This document is an initial contribution to a design guide intended to provide insight into the system safety domain, present a general technical foundation for designers and evaluators of safety-relevant systems, and serve as a reference for designers to formulate well-reasoned safety-related claims and arguments and identify evidence that can substantiate the claims. That evidence forms a basis for demonstrating compliance with certification regulations, and its generation is a major objective of a system development process. This work is part of an effort to enable sound assurance of safety-related properties of computer-based aircraft systems by developing an effective capability to model and reason about the safety implications of the system requirements and design.

1.3. Approach

The approach selected for the design guide is to present a summary of major theoretical and practical considerations relevant to the design and evaluation of safety-relevant computer-based systems. The presentation leverages a unified abstract model of system safety applicable globally and locally at every level of a design hierarchy to ensure that the material is intellectually accessible to a broad audience with the wide range of knowledge and experience on system development and safety. This is accomplished, in part, by leveraging the general concept of a system throughout the presentation and using technical concepts and terminology from computer science and engineering only where needed to ensure clarity. The aim is to simplify the description of the domain and offer sufficient insight to enable the identification and definition of evidence needed to support explicit claims that the system safety risks are acceptably low.

This document provides basic foundational material for the design guide in the form of an overview of concepts in system design, lifecycle, and safety. This document is also a survey of major and influential research and engineering publications in these areas.

Future sections of the design guide will introduce a general safety-risk mitigation strategy based on the application of rigorous development processes to minimize the likelihood of design errors, and system architectures that can mitigate residual logical defects and physical faults. The concepts and safety

implications and considerations of functionally integrated and distributed systems will be reviewed, followed by an overview of system-safety risk assessment. The guide will also offer insight into various particularly complex and consequential aspects in the design and development of modern safety-relevant computer-based systems.

2. System Design, Lifecycle, and Safety

This section presents an introduction to basic concepts that provide foundation, background, and motivation for the rest of the document. It begins with an overview of basic aspects of the design of a system. This is followed by a review of important system lifecycle concepts, including topics such as concept development, requirements, and system architecture. The section ends with an overview of system safety.

2.1. System Design

An engineered **system** is an entity with a purpose that is achieved by the interactions (i.e., relations) of its internal components (i.e., sub-entities) with each other and with the external environment (i.e., surroundings). A system is defined by a **boundary** (i.e., border) which delimits the extent of the system and is where the system interacts with its environment.

The concept of system is circular and recursive. In general, both the environment and the components of a system are themselves systems. The **system of interest** (SOI) (or **system in focus)** is identified by its boundary. An SOI is always contained within a larger **super-system** (or **meta-system**) which includes the SOI and its environment. An SOI also contains **sub-systems** which are sub-sets of internal components and their interactions. This recursion is bounded above and below based on relevance to the problem or matter under consideration. This system hierarchy has a tree structure with the super-system composed of the SOI and its environment at the top, and each sub-system branch is decomposed into one or more layers of sub-systems until **atomic** components (or **items**), whose internal structure is irrelevant or unknown, are reached at the bottom.

The purpose of an engineered system is defined in terms of the desired effect on the environment. This effect is achieved by the flow of matter, energy, or information between the SOI and its environment. An SOI can have different kinds of sub-systems such as mechanical, hydraulic, pneumatic, thermal, electrical, electronics, human, and computer, as well as combinations of these, which interact to achieve the system purpose.

Henceforth, the focus of this guide will be on **digital computer systems** embedded in larger engineered systems. A computer system is a **data processing system** (DPS) whose primary inputs and outputs are data sequences. A computer sub-system interfaces with other kinds of sub-systems to achieve the desired overall purpose of the containing domain system. The environment of a computer consists of other sub-systems as well as the environment of the domain system.

2.1.1. Environment

The environment of a system of interest can be divided into three parts. The **external systems** are the set of external entities that directly interact with the SOI at its boundary [42]. The **context** is the set of external entities that have a significant influence on the SOI but are not themselves directly influenced by the SOI. Beyond the external systems and the context is the rest of the environment which has little or no influence on the SOI and is thus considered outside the scope of relevance to the purpose of the SOI.

2.1.2. Function, Behavior, and Service

The engineering **function** of a system is what the system is intended to do [23], and it is described in

terms of inputs, outputs, and the relations between these. In general, the system inputs and outputs are sequences (or flows) of data items. Each data item in a sequence is characterized by a value and time of occurrence. The data items of a system are carried by input, output, and internal **variables** (or **signals**).

The mathematical **function** of a system is the relation between the values of input and output variables of the system. The **behavior** of a system is the functional and temporal relation (i.e., value and time) between the inputs and outputs of the system.

There are three basic types of behaviors. A **transformation** behavior changes the values of input variables into new values at the output variables. A **storage** behavior buffers the values of input variables until some future point in time when they appear at the output variables. A **transfer** (or **distribution)** behavior simply transfers the values of input variables to output variables. In general, the behavior of a system is a combination of these basic behaviors intended to change the attributes (i.e., what, when, and where) of data items.

The **service** delivered by a system is the output flow as perceived by its users. A **user** is a system or entity that receives a service.

2.1.3. Structure

The structure of engineered systems is generally hierarchical consisting of a set of internal composite and atomic components, their containment relations, and interconnections. The structure of the system is what enables it to generate the intended behavior [23]. Every component has a particular behavior, and the hierarchical composition of these internal behaviors results in the intended top-level system behavior[1].

The internal structure of computer systems can be generically described as a hierarchy of computation nodes that perform data transformation and storage functions, and a communication capability that transfers data between nodes (see Figure 1). Non-atomic nodes are composite sub-systems that contain internal structures that can also be described with this generic pattern of computation and communication capabilities, as illustrated in Figure 1.

[1] Note that this relation between the external behavior of the system and its internal structure is especially significant for safety-relevant systems because as the complexity of the desired system functionality increases, the complexity of the behavior of the internal system components and their interrelations also increases.

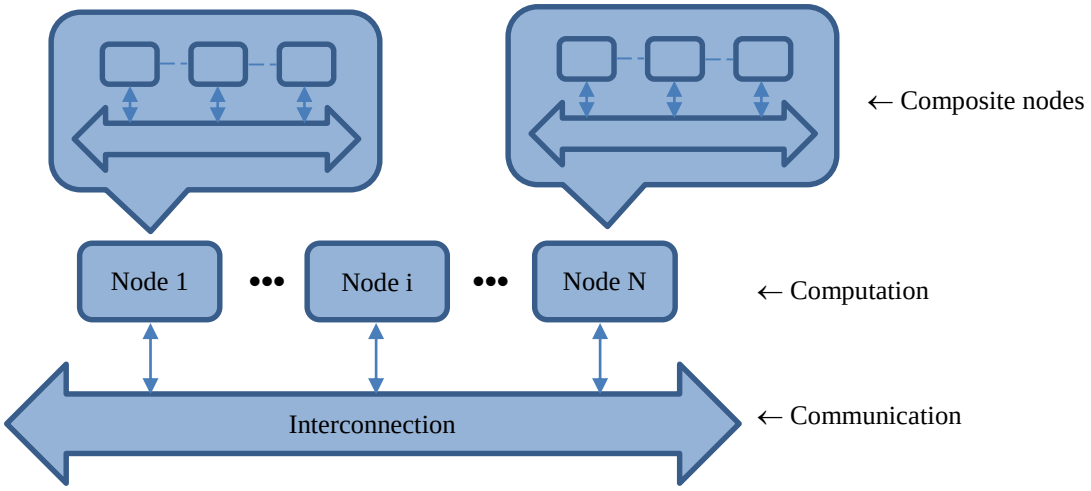

Figure 1: Generic System Structure

2.1.4. Informational, Logical, and Physical Layers

A computer system, whose purpose is to process data, can be viewed at three basic levels as illustrated in Figure 2. This is a simplified version of the models described by Avizienis in [43] and Parhami in [44]. In the **informational** layer, the system handles data which is processed to accomplish the purpose of the system. The function of the system is achieved by its internal logic, which performs combinations of the basic operations of transforming, storing, and transferring data. The **logical** layer is where the computation algorithms are described [45], [46], [47]. The informational and logical levels are abstract representations of a system in terms of operations performed on data. The **physical** layer is the substrate that realizes the system in a concrete physical sense enabled by equivalence relations (i.e., mappings) between the abstract data and functions at the informational and logical levels and conditions and processes at the physical level of the system [48], [49], [50]. The physical level of a system is where actual physical space (i.e., volume) is occupied, power is consumed, and heat is generated in the performance of the system functions.

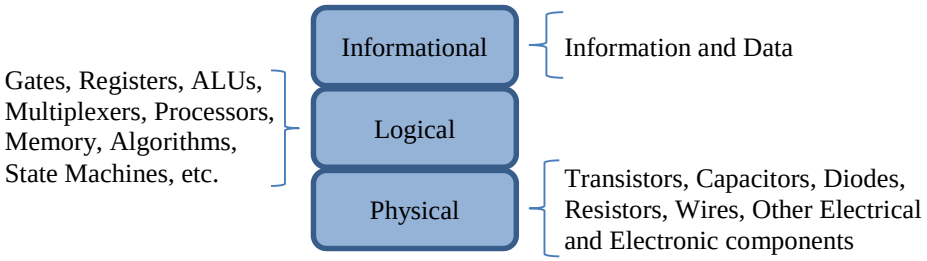

Figure 2: Basic System Layers

2.1.5. Hardware and Software Layers

A computer system, in the general sense of data processing system, consists of a set of interconnected logical components which together perform the intended function of the system. A typical computer

contains one or more physical devices (i.e., **hardware**) that perform functions such as input and output of analog, digital, and discrete data; data memory; data communication; application-specific data processing; and general-purpose data processing [51]. In general, many of these devices have degrees of programmability, performed either offline or online, that enable the specification of a desired functionality from a range of possibilities based on the particular needs of the application. This can include devices such as programmable read only memories (ROM), gate arrays, field programmable gate arrays (FPGA), semi-custom standard cell design, and processing units (e.g., microcontrollers and general-purpose and graphical microprocessors) [52]. These devices have particular means of programming. In the case of processors, they have instruction sets that specify the units of computation that a processor can perform [51]. **Software** is a sequence of instructions (i.e., a program) stored in memory to be executed by a processor to realize a desired function.

As illustrated in Figure 3, a computer system consists of a hardware layer and a software layer, and the logical layer of the system is divided between the software and the hardware layers. Because of the great flexibility and space (i.e., volume) efficiency afforded by programmable processors and software, most of the functionality in modern avionics systems is implemented with software running on processors [52]. Much of the complexity growth in modern systems is happening in the software. However, the processors and other hardware devices are also increasing in complexity, as predicted by Moore's Law[2] [52].

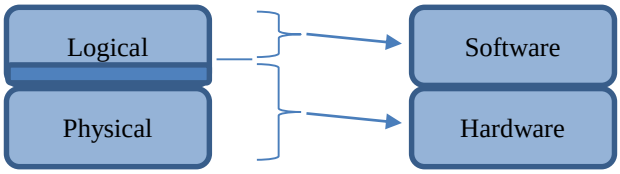

Figure 3: Mapping from Physical and Logical Layers to Hardware and Software Layers

2.1.6. Logical Processes

The behavior of a computer system is determined by the logical layer, which is itself a sub-system of the computer and is allocated between the hardware and the software. The concept of a **process**, defined here as a logical component that performs a particular function, has similar definitions in the domains of systems [42], hardware [53], and software [54]. Thus, this concept will be used henceforth to refer to logical system components. We will refer specifically to hardware and software processes only where the distinction is significant.

Logical processes operate on data flows, whose items have value and time attributes. Also, note that a logical process may be composed of multiple interrelated sub-processes whose actions must be coordinated in order to achieve a desired overall function. The following sub-sections address various aspects of the value and time dimensions of logical processes and the data they manipulate.

2.1.6.1. Data Flow

The data flow pattern of a system is the set of relations in the flow of data from the inputs, between internal processes, and to the outputs. The data flow relations in a properly executing system determine the

[2] Moore's Law is the observation that the processing power and number of transistors in microprocessors will double approximately every two years (i.e., exponential growth rate) [52].

data dependencies among processes such that the data flows among processes in a particular order and each process executes only when it has received the necessary data items. In effect, the data flow relations constrain the relative order of execution of the processes based on the availability of required data [55].

2.1.6.2. Control Flow

The control flow pattern of a system is the set of relations on the relative sequence and timing of process activation as determined by the algorithm being executed. The control flow relations are based on decisions regarding process activation sequences. The system control flow can include process activation patterns such as unconditional sequence, branching (i.e., selection), looping (i.e., repeat a predetermined number of times), concurrency (i.e., concurrent execution of multiple processes), and synchronization (i.e., multiple concurrent processes wait for each other at a point in their execution before continuing) [42]. Depending on the application, there may also be timing constraints such as relative delay in activation between processes, the rate of activation of the processes, or the duration of execution of a process [27]. These control flow relations between processes establish **control dependencies** which must be satisfied to ensure the proper behavior of the system.

Data and control dependencies in a system must be managed carefully as they are critical determinants of the safety-relevant characteristics of the system, including modes and probabilities of failure.

2.1.6.3. State and Event

The concept of the state of a system has various definitions in different fields of study. The common notion in all of them is that the **state** of a system at a point in time is the information (about the status or condition of the system) that, together with the input, determines the future behavior of the system [56], [57], [58], [42]. The **state** of a variable is its value [59]. An **event** is a change of state at a point in time [57].

2.1.6.4. Sequential Machines, Time and Triggers

A digital computer system, regardless of its design and mode of operation, is fundamentally a **sequential finite-state machine (FSM)** that contains internal memory elements and whose outputs are a function of present and past inputs as well as the initial state of the machine [51]. Structurally, a computer system can consist of multiple levels of interacting sequential sub-machines. The data flows in a system at the inputs, outputs, and between processes carry information about states and events in the system and its environment. Changes in the memory and output of a system are conditioned on the input and the state of the system, and triggered by events on the inputs or state of the system.

An **asynchronous** sequential machine is triggered (i.e., activated) and updates its state and outputs when input events occur. A **synchronous** sequential machine is triggered at points in time marked by the output of a clock oscillator which can be internal or external to the system [51]. Both asynchronous and synchronous sequential machines are triggered by events, and the distinction between the two is whether the triggering events are strictly time-based or not.

From a general perspective, logical processes are triggered by events which can be time-referenced or not [57], [58]. An **event-triggered** process is activated by particular non-time-based events. A **time-triggered** process is activated strictly by time-based events. These time-based events can be derived from logical-time clocks that keep track of the passage of time as indicated by clock oscillators [60], [61].

Process triggering is a consideration related to the control of system execution flow. The distinction between time and event triggering is important because these modes of triggering have different advantages and detriments in the problem of coordinating multiple system processes in safety-relevant systems, where it is critical that the system exhibit a high degree of predictability even under conditions of failure, disturbance, and degradation [57], [58], [62], [63], [64].

2.1.6.5. Event Timing

The timing of events is an important consideration in system design. The relative offset between two events is equal to the real time elapsed between the occurrence of the events. This can be characterized by lower and upper bounds on the offset.

One significant timing attribute of an event is its **periodicity** (or **recurrence**), which is the time ΔT elapsed between consecutive instances of the event (i.e., **inter-arrival time**) (see Figure 4). Table 1 shows the range of possibilities of event periodicity based on the upper and lower bounds of the inter-arrival time. The **jitter** of an event is the difference between the upper and lower bounds of the inter-arrival time (i.e., $\Delta T_{max} - \Delta T_{min}$) [58].

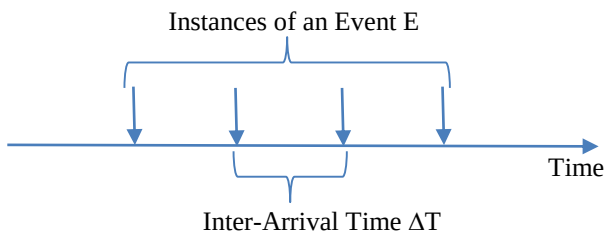

Figure 4: Inter-Arrival Time

Table 1: Event Periodicity Levels

Event Periodicity		**Inter-Arrival Time**	
		Lower Bound	**Upper Bound**
Periodic (i.e., fixed rate)		Nominal ΔT	Nominal ΔT
Sporadic	Rate Constrained	ΔT_{min}	ΔT_{max}
Sporadic	Minimum-Rate Constrained	Unbounded	ΔT_{max}
Sporadic	Maximum-Rate Constrained	ΔT_{min}	Unbounded
Aperiodic (i.e., unconstrained/unbounded rate)		Unbounded	Unbounded

Process **latency** (or **response time**) is the time duration between activation and completion of an instance of execution of a process. Latency is characterized by lower and upper bounds. The difference between these is the **latency jitter** of the process.

If two logical clocks are nominally periodic with the same period, we may be interested in whether they are synchronized or unsynchronized (i.e., asynchronous). The **relative skew** between two clocks is the real time elapsed from the instant one clock makes a particular state transition (i.e., the count reaches a particular value) until the other clock makes the same transition [61]. Two clocks are synchronized if the relative

skew has a known upper bound which is much smaller than the period of the clocks.

2.1.6.6. Value and Timing Uncertainty

In digital computer systems there can be various sources and conditions of uncertainty (i.e., unknown but possibly bounded) in the duration of operations, the timing of events, and the value and time of data items. From the perspective of processing and communication, the workload and latency of a process can vary over time, the latency of a logical process can vary for the same workload (e.g., due to data-dependent control flow), different processes performing the same computation can have different latencies, the communication delay between processes can vary [57], and the drift rate of clock oscillators can vary [61]. The execution timing of modern processors can be unpredictable due to advanced features such as cache memories, pipelining, branch prediction, out-of-order execution, dynamic scheduling, interrupts, multi-master arbitration of buses, and the use of multiple direct memory access (DMA) engines [65]. Reference [66] offers an overview of the problem of estimating worst-case execution time (WCET).

Digital computer systems are discrete-value and discrete-time systems [67], [52]. From this perspective, uncertainties are primarily due to the sampling and discretization of physical analog quantities, such as position, speed, angle, and time. The data item uncertainties in the system can be relative to the real physical world (i.e., **accuracy**) or relative between different data items representing the same quantity (i.e., **precision**) [58].

Another source of uncertainties, both in terms of accuracy and precision of data items, is finite-precision computer arithmetic. The problem of developing algorithms to perform precise computations using finite-precision arithmetic is studied in the field of numerical analysis [68]. One concern regarding finite-precision computer arithmetic is that the computation errors vary with the specific mathematical expressions used and the order in which they are evaluated. Thus, even if two processes compute the same function, their results can be different in ways that can be significant for particular applications [69], [70], [71]. Hall and Driscoll in [72] provide a list of numerical analysis consideration for safety-relevant systems.

This issue of value and timing uncertainty is an important aspect in the problem of achieving predictable interactions and coordination of multiple system processes in safety-relevant systems.

2.1.7. Interfaces

An interface is a link (or point of interaction) between entities. The purpose of an interface is to enable the exchange of information (i.e., communication) between entities. These interfacing entities can be systems or components, as a system consists of components and interfaces at the inputs, outputs, and between components.

We consider interfacing at the physical and logical levels. Hardware interfaces are a combination of physical and logical level interfaces. Software interfaces are logical interfaces. Hardware – software interfaces link the logical layer of the hardware with the software.

The scale and complexity of a communication problem is determined by the physical and logical attributes of the entities and what lies between them. A general communication system must deal with a complex set of problems such as physical media, signaling, formatting, synchronization, flow control, addressing, error management, and routing [73]. Most component communication problems are simpler instances of this general problem.

2.1.7.1. Physical Interfaces

Interfaces at the physical layer[3] deal with the mechanical and electrical characteristics of interacting components and the communication medium, including considerations such as power, volume, thermal properties, voltage and current levels and timing, signaling rates, signal propagation, electrical interference, and others related to the physical environment [73], [74]. These interfaces can range from interconnects on integrated circuits [49] and printed circuit boards [75] to cables and connectors for interfacing between boards, modules, and cabinets [76].

2.1.7.2. Logical Interfaces

Logical components interact to achieve the larger purpose of the system that contains them. These interactions are exchanges of functional and temporal information in the form of states and events related to the data and control flows of the system [57], [58]. In modern systems, the interactions can be highly complex with intricate patterns of data and control dependencies between components at all levels of the structural hierarchy. The logical interfaces are critical determinants of safety-related system properties as they can be sources of uncertainties and also serve as paths for the propagation of effects of value and timing uncertainties. The dependencies between components must be carefully managed to ensure system-level predictability even under conditions of failure, degradation, or disturbances.

Hall and Driscoll in [72] provide a list of considerations for system interfaces and environment in safety-relevant systems.

2.1.8. Functional Modes

A **mode** of a system is a distinct operational capability [42]. Modes are high-level states that determine lower level states and functionality (i.e., algorithms), including state transitions. An aircraft system may have different available operational modes for different operational mission phases and conditions. The components of a system may also have their own operational modes. Statecharts are an effective means to model the mode transition logic of a system [77].

Figure 5 illustrates the basic system modes and transitions. A system's functionality may not be required at a particular point in time (i.e., there may not be a **demand** for it). When there is a demand for it, the system must first go through an initialization and startup process to reach a state of operational readiness. When required, the system transitions to the operational mode where it may have a number of sub-modes and mode transition logic. When the functionality is no longer needed, the system executes an orderly shutdown process.

In general, this basic mode logic applies to a system as well as its internal components. An important system design problem is the relation between the modes of the system and the modes of the components and how to properly coordinate mode transitions of system components. This system design problem is critically important in safety-relevant systems, which must achieve predictability at the system level even under conditions of uncertainty about the status of the components due to failures, disturbances, or degradations.

---

[3] Note that the concept of physical layer here is a subset of the one used in standard computer communications theory.

Hall and Driscoll provide a number of considerations for initialization and startup in safety-relevant systems [72].

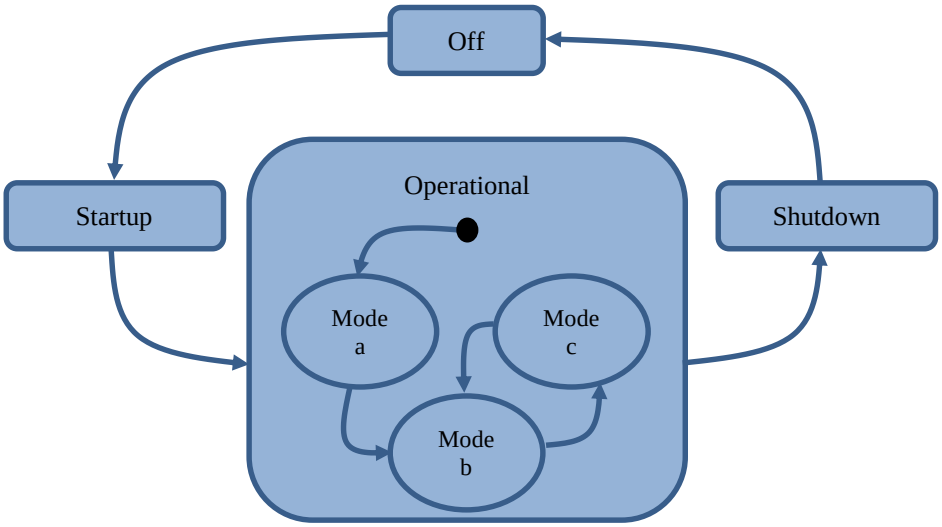

Figure 5: Basic System Modes

2.1.9. Composition

The purpose (i.e., the goal) of a system is to have a desired (i.e., intended) effect on its environment by interacting with it. In general, this interaction sets up an interdependence (i.e., mutual dependence) relation between the system and the environment such that the intended environmental effect is achieved if the output of the system satisfies certain assumptions, but the system can only guarantee properties about its output if the environment satisfies certain assumptions (see Figure 6). Thus, the goal of the system is achieved if the guarantees by the system are a superset of the assumptions about the system and the guarantees by the environment are a superset of the assumptions about the environment. These assumptions and guarantees cover both the physical and logical layers of the system and the environment. Proper operation requires that the system and the environment restrain their physical and logical actions at their interfaces to remain within assumed (i.e., hypothesized) bounds. Physical layer assumptions and guarantees concern physical interaction properties (e.g., mechanical and electrical). Logical layer assumptions and guarantees are functional and temporal (i.e., value and time) properties of data flows.

The goal of a system design is to satisfy required service guarantees. These can be functional and non-functional service goals. Functional goals concern the input-output response and achieving and maintaining certain output conditions. Non-functional goals pertain to the quality of the service in terms of criteria such as reliability, availability, integrity, and performance [42], [59].

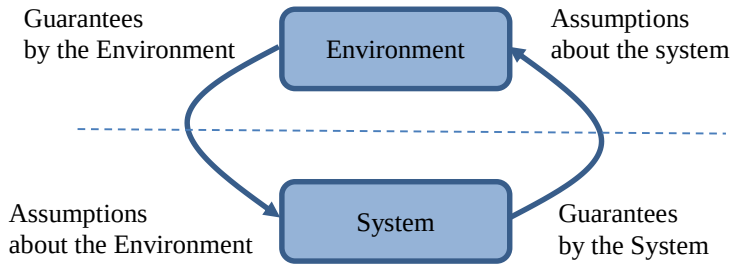

Figure 6: Assume-Guarantee Model of Interaction between System and Environment

The design of a system consists of the specification of the components and the interaction protocols. The assume-guarantee model of interaction also applies at this level. Thus, the system service properties are a result of the properties of the components and the properties of the interactions between them and with the environment (see Figure 7). The delivery of system service guarantees requires that the actions of the components and the interaction protocols remain within assumed bounds.

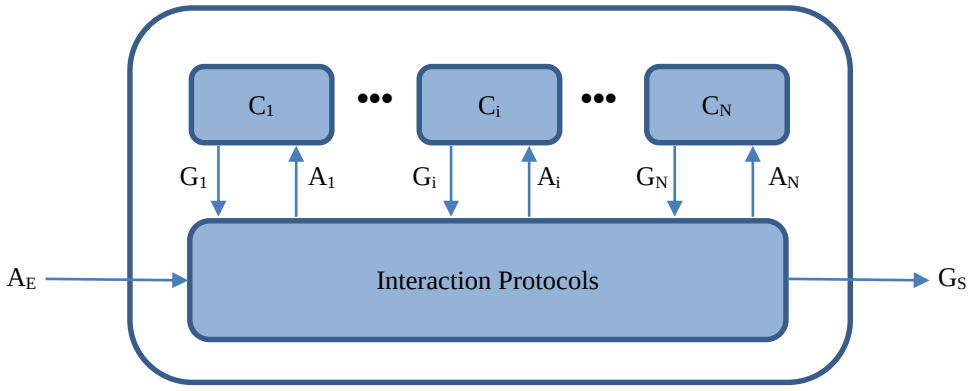

Figure 7: Assume-Guarantee Relations in System Composition

Composition analysis is central to achieving predictable (i.e., guaranteed or ensured) system behavior in safety-relevant systems. Violation of system-level guarantees is the result of the violation of physical or logical assumptions about the environment, the components, or the interaction protocols. Such violations cause unintended (or undesired) interactions whose effects can propagate throughout the system and out to the external interface. Predictable system behavior under conditions of failure, disturbances or degradation requires that the system design be able to tolerate a bounded degree of uncertainty about the actions of individual elements and the number of elements whose actions are uncertain.

Hall and Driscoll have provided a number of considerations for system temporal composition and determinism [72]. Additional information about assume-guarantee reasoning and the related topic of contract-based-design is available from Bate et al. [78] and Benveniste et al. [79].

2.2. System Lifecycle

This section goes over some of the considerations in the system lifecycle, the high-level design of a system, the development lifecycle, and management of the evolution of the system after entry into operation. The system development process is the set of interrelated activities performed to produce a system that meets the needs and goals of the stakeholders. This process should include explicit activities to identify the full set of stakeholders, as their needs and goals are the basis for the system requirements. The system architecture is the strategy to satisfy the requirements and it is a critical factor in the effectiveness of a system throughout its lifecycle.

2.2.1. System Lifecycle and Stakeholders

It is important to take a broad perspective in multiple dimensions when developing a system. One aspect of this is considering the full lifecycle of the system, which covers all the phases in the life of the system beginning with the identification of the operational need and ending with disposal after the system is no longer needed. Figure 8 illustrates the phases and periods in the lifecycle of a system [42], [80], [81]. In the **development** period, the operational needs and goals of the stakeholders are identified, system concept studies are performed, a conceptual design is selected, necessary technologies are developed, and the system is designed. The **pre-initial operational capability** period overlaps with the development period and includes the production and manufacturing phase of the system, deployment to the operational field, and training of operators and maintainers of the system. In the **operational use and refinement** period, the system sees operational use while production and deployment of the system continues and the system is refined based on needs identified during operation. In the **retirement** period, the system is no longer refined or produced, but operational use continues while instances or parts of it are decommissioned and disposed of.

Another area where taking a broad perspective is important is in regards to the stakeholders. A system **stakeholder** is an individual or group who has an interest in some aspect of the system lifecycle because they are affected by or accountable for the outcome. Stakeholders can be internal or external to the organization(s) directly involved in the lifecycle of the system. Examples of stakeholders include owners, developers, manufacturers, trainers, operators, maintainers, victims (if there is an accident), and government regulators. Every stakeholder has a different perspective on the system lifecycle and different needs and goals. Stakeholder identification, analysis, and management is crucially important because their needs, goals, values, capabilities, and constraints influence every aspect of the system lifecycle [81], [82]. The system is intended to satisfy the needs of the stakeholders. The success or failure of the system will be decided by them.

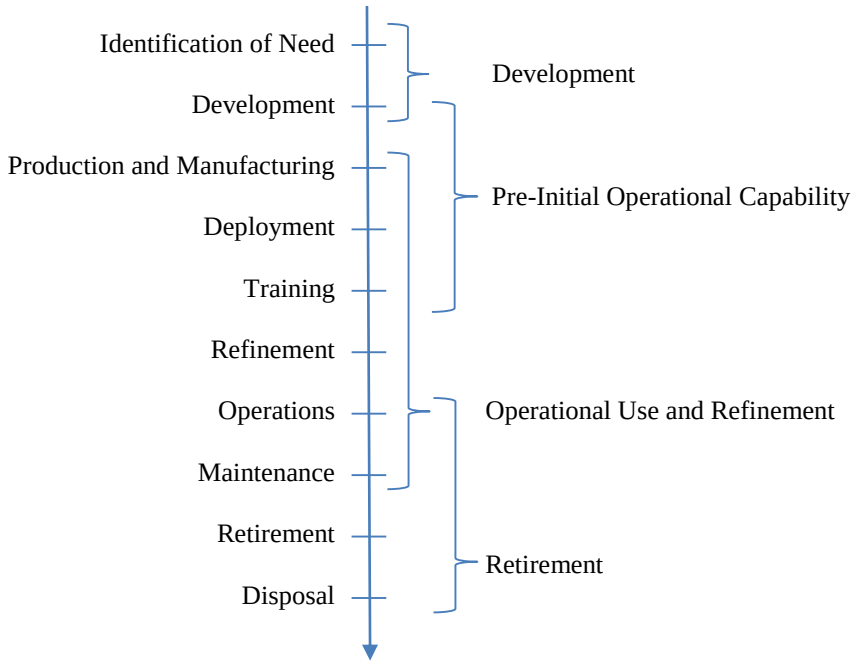

Figure 8: System Life Cycle Phases and Periods

2.2.2. Operational Needs Analysis and Concept Development

The origin of an engineered system is a perceived need (or opportunity), either current or projected, in the operational domain of an organization. The purpose of an operational needs analysis is to identify and characterize operational capability gaps (i.e., deficiencies) relative to the goals and values of the stakeholders. Operational needs can arise for various reasons such as change in goals, conditions (e.g., product obsolescence, change in the operational environment), or regulations, and availability of new or more effective capabilities enabled by new technology. Note that the broader notion of system, as a set of interacting elements with a purpose, applies to organizational operations. Thus we will refer to this as the **operations system**, which may have technological (e.g., machines and tools) and human elements with machine-machine, human-machine, and human-human interactions within the system and between the system and its environment.

A number of tools can be used for the needs analysis including operational tests, experiments, and modeling and simulation. Qualitatively and quantitatively **indices** (or **measures**) **of performance** (IoP) relevant to the goals and values of the stakeholders can be used to measure operational effectiveness and efficiency. Examples of these include operational cost, delays, and processing capacity, as well as quality measures such as safety, availability, reliability, and flexibility. Systems-based approaches can be leveraged in assessments of operations systems. Examples of these approaches include hard systems thinking, system dynamics, and soft systems methodology [83]. Gibson's systems analysis approach is another means of gaining insight into operational problems, as well as developing candidate solutions [84]. Systems Engineering handbooks by NASA and MITRE offer additional approaches and guidelines for the assessment of operational needs [81], [85].

The next part of the operational needs analysis is determining whether a technically feasible and cost effective solution exists. This involves the exploration of the problem and solution spaces to identify and develop a plausible **concept of operations** (ConOps), which is a description of an envisioned system and operations that satisfy the needs of the stakeholders. Kossiakoff et al. [86] describe a structured approach for concept development. IEEE Standard 1362-1998 is a guide for documenting ConOPs [87]. A fully developed ConOps considers the needs of all the stakeholder groups and covers the full lifecycle of a system from development to retirement and disposition [42]. A ConOps describes operational scenarios in the environment of a system of interest (SOI) and serves as the basis for defining the boundary and required major capabilities and attributes of the SOI. Domain knowledge, insight into the stakeholder needs, awareness of technology and capability options, and creativity are critical factors in the development of an effective ConOps. A concept of operations may also be referred to as a *concept of execution*.

2.2.3. System Requirements

The system requirements capture the needs and goals of the stakeholders for the SOI. As the purpose of a system is to achieve a desired effect on its environment and an overall performance level in the operations system, the SOI requirements are primarily statements about phenomena in the environment in the form of monitored inputs and controlled outputs by the SOI. Part of the system development effort and operations system analysis is to determine the SOI inputs and outputs and their relations such that the desired overall operational goals are achieved. The effectiveness of the requirements in realizing the operational goals is dependent on known properties and assumptions (i.e., hypotheses) about environmental processes.

There are a number of ways to structure the system requirements. The three basic categories of requirements are **functional** (i.e., behavior in interaction with the environment), **non-functional** (i.e., attributes, qualities), and **design constraints** (i.e., limits on the design of SOI) [88]. Examples of non-functional system requirements include reliability, maintainability, usability, safety, availability, flexibility, producibility, testability, disposability, and performance (e.g., throughput, reaction time, size, speed, weight).

Buede describes a thorough and insightful category structure for requirements in which, for each system lifecycle phase, there are input/output, technology and system-wide, trade-off, and qualification requirements [42]. **Input/output** requirements include descriptions of valid inputs and outputs, functions (i.e., input-output relations), and external interface constraints. The **technology and system-wide** requirements are constraints and performance thresholds on physical system resources, including technologies to be used (or not used), qualities (e.g., reliability, availability, safety, maintainability), and lifecycle cost and schedule. The **trade-off** category of requirements are decision algorithms used for comparing and selecting among design alternatives relative to performance and cost. Trade space exploration and decision making use the trade-off requirements combined with an objectives hierarchy, which is a hierarchical structure of performance and cost objectives expressed in terms of qualitative and quantitative figures of merit (FoM) with relative weights, valid ranges, and utility (i.e., value) curves as determined by the stakeholders. The **qualification** requirements specify the plans for validation, verification, and acceptance of the system by the stakeholders and the data to be gathered in support of assessments of quality for the developed system.

Requirements engineering is the discipline concerned with the elicitation, evaluation, specification, and quality assurance of requirements [59], [85]. The system requirements are generated in an iterative process of interaction with stakeholders. As illustrated in Figure 9, requirements engineering operates in a

cyclic process of exploration and discovery in which additional requirements are identified and generated based on new domain knowledge, followed by a quality assessment activity to ensure the effectiveness of the revised requirements set. Assurance of requirements quality is about detecting and correcting defects by means of inspections, reviews, simulations, and formal analysis. Requirements **validation** activities check the requirements set against desired extrinsic properties aimed at ensuring completeness and adequacy, while requirement **verification** activities ensure desired intrinsic properties such as consistency, unambiguity, traceability, feasibility, and verifiability (i.e., be able to check that the developed SOI meets the requirements). Requirements engineering is also responsible for monitoring the stakeholder needs to identify changes over the lifecycle of the system and generate suitable requirements as the basis for refinement and evolution of the system.

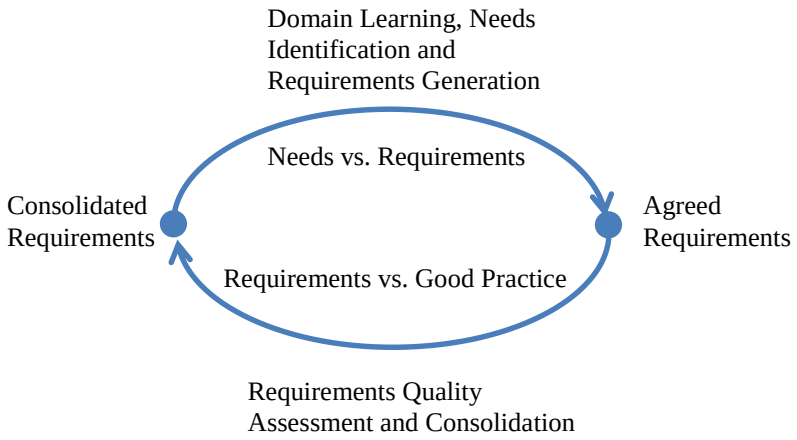

Figure 9: Basic Model of Requirements Engineering Process

Additional information on requirements engineering is available from many other sources. Hall and Driscoll have a list of questions to ask regarding system requirements [72]. The FAA has published a Requirements Engineering Management Handbook [89]. The following sources also have useful information on requirements engineering: [59], [81], [85], [86], [90], [91], and [92].

2.2.4. System Architecture

The system requirements are the goals to be achieved by the system, and the system architecture is the high-level strategy to achieve those goals. The IEEE defines **architecture** as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [93]. As a set of interacting components whose properties are integrated through the interactions, the architecture provides the link between the system assumptions and the required system-level properties (i.e., emergent holistic guarantees). The architecture describes not only how the functional requirements are satisfied, but also the non-functional quality requirements [94]. The architecture must account for the full system lifecycle, from development to disposal [42]. The architectural description should include the structural relationships as well as behavioral aspects of the components and their interactions to ensure that the system will work as intended [95].

The architecture also serves as a means to reduce the complexity of a system design problem. A design

process consists of four basic activities: **abstraction** (i.e., generalization and omission of irrelevant details); **decomposition** (i.e., hierarchy; partitioning; reduction of an object into smaller and simpler parts); **elaboration** (i.e., adding detail, either structural or behavioral (functional or temporal)); and **decision making** (i.e., selection among alternatives) [96]. Decomposition can be **horizontal** (i.e., breakdown into multiple modules at the same level of abstraction) or **vertical** (i.e., breakdown into multiple modules at sequentially lower levels of abstraction). With vertical decomposition (also referred to as **layering**), every layer receives a cohesive set of services from the lower substrate layers and provides more abstract services to the layers above [97], [98]. In addition to abstraction and decomposition, other classical complexity reduction techniques include **regularity** (i.e., decomposition into regular structures with similar components), **modularity** (i.e., decomposition into components with particular well-defined functions and interfaces), and **locality** (i.e., information hiding and use of mainly local variables) [49]. Reusable architectural patterns can also be used to solve recurring design problems and manage the complexity of a system [99], [100].

One approach to develop a system architecture is to decompose the problem into the development of a functional architecture and a physical architecture which are combined as an allocated architecture [42]. The **functional** (or **logical**) **architecture** is an arrangement of system functions, their decompositions, and interfaces, and the definition of control and data flows at internal and external functional interfaces. The **physical architecture** is an arrangement of physical resources for computation and communication, their decompositions, and internal and external interfaces and their physical constraints. The **allocated architecture** is a complete system architecture with the mapping of functions to physical resources that implement them. In the case of a computer system, the functional architecture could be an application-dependent software architecture and the physical architecture could be a computing platform consisting of hardware resources and an operating system. The advantage of this approach is that the design problem is decomposed into major manageable parts whose development can proceed concurrently, though not completely independent as there are relations between the functional and physical architectures which must be actively managed during the development process.

Architecture design can be viewed as a decision-making process [101], [91], [102], [103]. These decisions are broad in scope and concern system-level properties captured in the requirements. Architectural decisions also pertain to guiding principles for development and evolution to ensure system-wide conceptual integrity over time [103], [102]. At each level of architecture development, decisions are made about how requirements will be satisfied, and these decisions result in particular mappings of requirements to the next level in the hierarchical decomposition. These architectural decisions are often made under uncertainty about operational conditions and about the implications of the decisions as the system development advances. Experienced engineering judgment is required to mitigate the risks to the development effort and the operational effectiveness of the system.

Bass, Clemens, and Kazman list a number of advantages of architectures, particularly software architectures, including: predicting system qualities, enhancing communication with stakeholders, defining constraints on implementation, allowing incorporation of independently developed components, enabling reasoning about and managing change, and influencing organizational structure [94].

Additional information and insight into system and software architectures is available from Maier and Rechtin [104]; Bass, Clemens, and Kazman [94]; Buede [42]; NASA Systems Engineering Handbook [81]; and the Guide to the Systems Engineering Body of Knowledge [105]. Cleland-Huang et al. describe the relation between software requirements and architecture [106]. Koopman has developed a taxonomy of decomposition strategies [107].

2.2.5. Development Lifecycle

The need to develop a system has its origin in identified operational deficiencies and/or awareness on the part of the stakeholders about technological opportunities to enhance operational performance with the introduction of new or more sophisticated capabilities. As described above, the path to realizing this potential operational improvement begins with an analysis of operational needs and the development of an operational concept. After this, the boundaries for the system of interest (SOI) are defined and the system requirements are generated describing the interaction between the SOI and its environment to achieve the desired operational outcomes.

The system requirements are, in effect, the definition of a technical engineering design problem to be solved and stated in terms of functional, performance, and quality goals and constraints. The complexity of a technical problem can vary depending on the number and diversity of stakeholders and their needs. The solution to a technical problem can range from a trivial one-possible-solution exercise to highly complex and multi-dimensional problems with multiple possible solutions and requiring active engagement by the stakeholders throughout the design process to ensure a satisfactory solution. This later case is the most general and most relevant to modern complex systems.

Solving complex system design problems involves a combination of rational analytic approaches (i.e., science) and experience-based intuition (i.e., judgment, design "art") [58], [104]. Gibson's Systems Analysis methodology offers insight and guidance into general goal-centered analysis and synthesis for complex systems problems [84]. This requires the definition of performance criteria, generation of alternative solutions, and ranking and selection of alternatives based on their performance. In general, arriving at a complete system solution is fundamentally an iterative and recursive decision-rich analysis process of decomposition that generates a hierarchical tree structure of simpler but interrelated sub-problems and continues along each branch until sub-problems are reached which can be solved whole, at which point the lower-level solutions are composed along each branch until the top-level problem is solved. Iteration is required in complex-problem solving to correct decisions that lead to inadequate results or unintended consequences. From a structural system perspective, this problem decomposition corresponds to components, interfaces, and interactions at different levels of abstraction. Thus, the design problem solution is a hierarchical structure of decisions ranging in scope from global system-level decisions to low-level decisions about components and interfaces.

Although it is known that the final complex-system solution will have hierarchical structure, there is no deterministic process by which to arrive at the final satisfactory solution. There are at least two reasons for this. First, the development of requirements is an iterative process of discovering and refining the stakeholder needs. In general, the basic needs can be derived from the operational concept, but the generation of a comprehensive and accurate set of requirements is complicated by the fact that the priorities of the stakeholders change over time and the IKIWISI (I'll-know-it-when-I-see-it) syndrome, which means that the stakeholders must interact with prototypes or early versions of the system in order to discover their requirements [108]. The second reason is that complex, interdependent, and potentially conflicting system requirements can lead to significant uncertainty about the implications of high-level decisions. This requires an iterative process to refine the requirements and discover a satisfactory solution. The design of complex systems is a *wicked problem* as it exhibits the characteristics that the problem is uniquely dependent on the specific characteristics of the environment, the problem cannot be defined independently of the solution, and there are no clear rules for when a final solution has been reached [58]. The complex-system solution must seek a balance between conflicting requirements, especially performance and quality requirements. Thus, there is significant degree of subjectivity and uncertainty in solving complex system

problems. Also, usually there is no single satisfactory solution, there are multiple decision paths (i.e., sequences) to arrive at a solution, and most decisions have the potential for unintended consequences. Complex systems typically fail because of unintended and unanticipated consequences of requirements and design decisions.

The decision to develop a system is based on cost and benefit considerations. The costs are the funds and time allocated to develop the system. The benefits are related to the degree to which the technical performance of the acquired system meets the needs and expectations of the stakeholders, while complying with applicable standards, guidance, regulations, and policies. These cost, schedule, and technical performance considerations are assessed based on estimates (i.e., models) of the acquisition processes. As with all models, these estimates are based on assumptions about required resources and the effectiveness and efficiency of the development process. Thus, risks (i.e., uncertainties and related potential undesired consequences) on attainment of cost, schedule, and performance goals must also be considered in development feasibility analyses.

A structured and managed development process is needed to ensure a successful development project. Figure 10 illustrates a simple high-level model of a system development project in which cost and time are "consumed" to transform the needs of the stakeholders into a system with the required level of technical performance. For this, a Development System of people, technology, and processes (e.g., methodologies, techniques, and tools) with certain level of capability and maturity [109] is engaged to develop the SOI. The relation between project cost, schedule, and technical performance is determined, in part, by the complexity of the technical problem and the capability and maturity of the development system. The management of a development project includes activities of planning, coordination, measuring, monitoring, controlling, and status reporting to stakeholders [91]. The project management function can be divided into two major sub-functions: project control and systems engineering [81]. Project control is responsible for activities such as planning and management of the schedule, system configuration, resources, and acquisition. Systems engineering is responsible for system design, product realization, and technical management, including planning, control, assessment, and decision analysis.

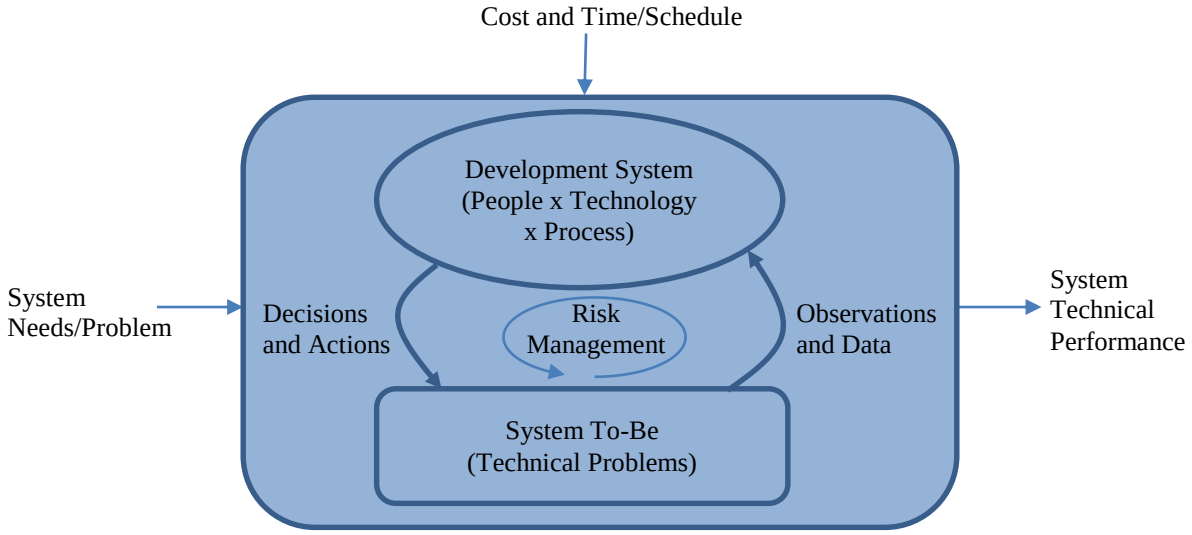

Figure 10: Top Level System Development Model

Risk management is a project management process responsible for the identification, assessment, and generation of mitigation actions against cost, schedule, or technical performance risks. The solution of unexpected problems during the development process typically requires tradeoffs among cost, schedule, and technical performance. The overall system development process should be managed to ensure that risks diminish and are minimized as the development advances toward completion. For safety-relevant systems, it is critically important that the technical performance of the developed system satisfy the requirements with high confidence (i.e., low uncertainty). Two important risk considerations in the development of a system are the readiness (i.e., maturity) level of the technologies introduced in the system (i.e., Technology Readiness Level, TRL) and risks associated with software development, which are caused by factors such as inadequate understanding of operational requirements and system interfaces, and lack of sufficient qualified personnel [92].

Feiler has identified two main points of concern in the development of embedded systems [110]. One of these problems is the potential for multiple truths in the results of system analyses. Loose coupling among system development teams and between development and analysis activities can lead to inconsistencies between models of different aspects of a system and also between the system being developed and the one captured in analyses. The second main problem of concern in the development is the introduction of errors early in the development process and their discovery much later during the integration of components or system-level testing. This is a concern because, in general, the cost of correcting errors increases exponentially with the distance in the development process between the point where they are introduced and the point where they are identified [110]. This is because both the breadth and depth of implications of design decisions increases as the development process advances. One significant source of errors and inconsistencies are mismatched assumptions about different aspects of a system. Assumptions of different sorts are leveraged to simplify and bound the design and analyses efforts, but for complex systems, successful development critically depends on the use of a consistent set of assumptions by everyone involved. Another source of development errors is the inability to identify and to understand all the implications of design assumptions and decisions. The difficulty in doing a thorough and precise examination increases with the complexity of the system.

Additional information and insight into the system development lifecycle, including risk management, is available from multiple sources [92], [81], [86], [111], [91], [105], [112], [113]. Information on readiness levels for technology, integration, and systems is available from [92], [114], [115], [116]. Information on formal contracts for systems design is available from [79]. Information on decision analysis is available from [42] and [81]. Hall and Driscoll have checklists for configuration management and organizational factors [72]. Goossen and Buster recently completed a study that identifies issues and makes recommendations to enhance the state-of-the-practice in regulatory compliance for avionics development in multi-tier supplier networks [117].

2.2.6. Refinement and Evolution

The requirements and design of a system is based on assumptions, expectations, and best understanding of stakeholder needs and the current and future environment in which the system will operate. For a complex system with complex requirements and design, perfection is unattainable and there are no means to show with complete certainty that a fixed final design will meet the stakeholder needs over the full system lifecycle. Instead, the decision to transition to production and deployment is based on an assessment of whether a satisfactory level of technical performance and expected lifecycle costs has been achieved and the operational impacts of any remaining shortcomings can be accepted as-is or adequately mitigated.

One way to manage initial development deficiencies or limitations and future changes in stakeholder needs is with refinement and evolution of the requirements and design over the system lifecycle. The changes to a system can be anticipated and planned for at the time of initial development, or the changes can be due to conditions that were not anticipated during initial development. System changes can be reactive or proactive to correct defects or to enhance technical performance. The system requirements and configuration management processes are critical enablers of post-development design changes by ensuring a thorough assessment of impact, feasibility, and implementation.

Additional information on product refinement and evolution (i.e., upgrades and modernization) is available from multiple sources. The Guide to the Software Engineering Body of Knowledge has extensive guidance on requirements change management, software maintenance, and configuration management [91]. Lamsweerde has insightful information on requirements change control and traceability management [59]. Other sources include [81], [92], [105], [86], [111], [113], [80], and [118].

2.3. System Safety

There are various definitions of safety and related concepts. Avizienis et al. define *safety* as "absence of catastrophic consequences on the user(s) and the (natural) environment" [23]. NASA uses the definition in MIL-STD-882 where *safety* is defined as "freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the (natural) environment" [119], [120]. Leveson defines *safety* as "freedom from accidents or losses" [34]. Ericson offers several definitions of *safety*, including "the state of being safe", where *safe* is defined as "the condition of being protected from danger, mishaps, or other undesirable consequences" [121]. Definitions of an *accident* include "an undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss" [34], and "an unplanned event, or events, that results in an outcome culminating in death, injury, damage, harm, and/or loss" [121]. MIL-STD-882 defines a *mishap* as "an event or series of events resulting in unintentional death, injury, occupational illness, damage to or loss of equipment or property, or damage to the (natural) environment" [120]. So, MIL-STD-882 equates mishap with accident. The FAA uses slightly different definitions in the context of aviation safety, where *mishap* is "a source of irritation, annoyance, grievance, nuisance, vexation, mortification … a minor accident"; *incident* is "an unplanned event that could have resulted in an accident, or did result in minor damage"; and *accident* is defined as "an unplanned fortuitous event that results in harm, i.e., loss, fatality, injury, system loss" [122]. Based on these definitions, *safety* is concerned with events or conditions with various degrees of severity in terms of loss or damage to people, property, or the natural environment. Henceforth, we will refer to these safety-relevant events or conditions as **mishaps**.

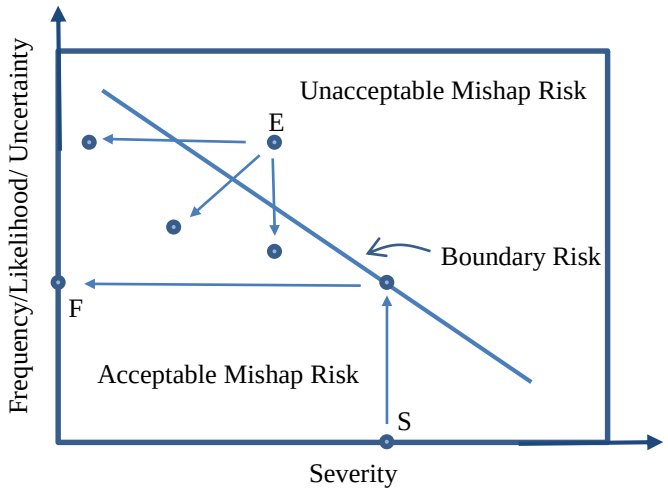

Figure 11: Safety Risk Space

It is generally impossible in the real world to guarantee the absence of mishaps. Thus, **safety** is better defined as "the state in which risk is acceptable", where risk is defined as "the combination of the frequency (or likelihood) of an occurrence and its associated level of severity" [25]. This risk of mishaps is known as **safety risk**. Figure 11 illustrates the relation between the risks of events and the **boundary risk**, which is the upper limit of acceptable risk. From the perspective of designing a system, the boundary risk is the maximum allowed **residual** risk when the system is in operation. The acceptable risk level varies with the needs and values of individuals and societies, as well as the practicalities and cost of achieving a particular level of risk (i.e., it is the result of implicit or explicit decisions based on cost-benefit assessment) [123]. In general, the boundary risk defines an inverse relation between the likelihood (or frequency) and severity of mishaps. Note that the *uncertainty* about future occurrences must be commensurate with the required upper bound on the likelihood (or probability) of an event. The lower the required probability of an event, the higher the certainty (i.e., *confidence*) must be that the event will not occur.

The concepts of safety and safety risk are assessed based on a model of a system, defined previously as a set of interacting components with a purpose. As safety pertains to physical damage, the system in question is a *physical operations system* with human and/or technological components (e.g., machines and tools) interacting with each other and with a physical environment. Mishaps are the result of a chain of causality that extends from root (i.e., initiating) causes to mishaps. A **hazard** is "a condition, event, or circumstance that could lead to or contribute to an unplanned or undesired event" [122], where the relevant unplanned or undesired event here is a mishap. Ericson defines a hazard as "an existing system state that is dormant, but which has the potential to result in a mishap when the inactive hazard state components are actualized" [121]. In general, other conditions in the system or the environment are necessary for the **activation** of a hazard (i.e., the propagation of effects in the form of a chain of events) and the occurrence of a mishap. Hazards are defined with respect to a physical operational system and its environment. Hazards are not necessarily root causes but can be intermediate conditions in a causal chain. The identification of hazards is the result of decisions based on multiple factors, some of which may be subjective, including the ability to describe the hazards clearly and concisely, allocate responsibility for the hazards in the context of a system development or sustainment process, and effectively manage safety

engineering activities [124]. The selection and definition of hazards is also based on considerations of observability and controllability of safety-relevant events or conditions. The **severity** of a hazard is the worst-case possible mishap what could result when other conditions are at their most unfavorable [34]. The **safety risk of a hazard** is the product of the hazard severity, likelihood of occurring, exposure (i.e., duration), and likelihood that the hazard leads to a mishap. The exposure condition is a critical risk factor as the longer a hazard remains present, the more likely that other required conditions will happen (i.e., **time coincidence**) and trigger a mishap.

The risk of a particular mishap can be mitigated by manipulating its causal chain to reduce the frequency of the mishap, its severity, or both (see event E in Figure 11). The mishap model can be reduced to a sequence of three elements: **hazards → mechanisms → mishap**, where mechanisms represent the means of propagation of hazards effects. Based on such a mishap model, there are four categories of risk mitigation actions [34]. The most effective risk mitigation action is **hazard elimination**, which results in an inherently safer system. This is also known as *design for minimum risk* [122]. The next risk mitigation alternative is **hazard reduction**, which reduces the likelihood of a hazard by increasing controllability of the system state, introducing barriers to prevent the coincidence of reinforcing hazardous conditions, and making a system more robust to prevent individual hazards from occurring. **Hazard control** can be achieved by reducing exposure and by containment of effects by passive means or by reactive means involving detection and activation of suitable mechanisms to limit the propagation of effects. Hazard reduction and control involve the incorporation of passive or automatic technological devices to enhance safety by and providing warnings in case of unsafe conditions. If this is not possible or practical, the next option is to introduce personnel procedures and training to prevent and manage hazards [122]. The last risk mitigation alternative is **damage minimization** by ensuring that potential victims of a mishap are protected from its effects (e.g., by keeping non-involved workers or the public away, or by operating in remote environments such as a desert).

The safety **threats** are the set of physical processes or phenomena that are causes of system hazards. **Endogenous** hazards are caused by factors inherent to the system, such as design defects or operational procedures [34]. **Exogenous** hazards are caused by phenomena external to the system, such as lightning. The set of relevant threats is determined based on knowledge of the domain, past experience, standards, or as required per regulations or policies.

The assessment of safety risk in the physical operations system is based on models of the system and assumptions about operating conditions. The system models must take into account **epistemic uncertainties** due to modeling abstractions and lack of knowledge about the system and its environment, and also **aleatoric uncertainties** due to inherent randomness or variation in the physical system and the environment. In general, the uncertainties in a computer-based system are primarily due to its complexity, which is a function of the number and variety of components and the intricacy of their interactions.

There are two basic types of safety analyses. **Inductive** analyses begin with the hazards and proceed forward along the chains of causality toward the mishaps. **Deductive** analyses begin with mishaps and proceed backward toward the hazards. These complementary analyses can be used to develop insight into the dependency structure and the hazard-effects propagation mechanisms of a system.

Safety analysis can be **qualitative** (i.e., subjective and non-numerical) and **quantitative** (i.e., applying mathematical methods), with qualitative analyses performed early on in the development process when the system exists mostly as a concept or high level abstraction, and quantitative analyses performed later when the design is sufficiently refined to use reasonably accurate quantitative measures. A safety assessment

must include an examination of confidence in the results (i.e., the claims) to ensure their validity based on the methods, tools, and techniques used, the supporting arguments, and the available data (i.e., evidence), which itself can be a mix of qualitative or quantitative items.

The safety goal in the design of an operations system is to ensure that the safety risk does not exceed the acceptable boundary risk. As illustrated in Figure 11, given the severity **S** of a possible mishap, the boundary risk determines the maximum acceptable frequency (or likelihood) **F**. Using deductive analyses, it is then possible to allocate risk (i.e., allowed events and frequency) to the various elements in the operations system.

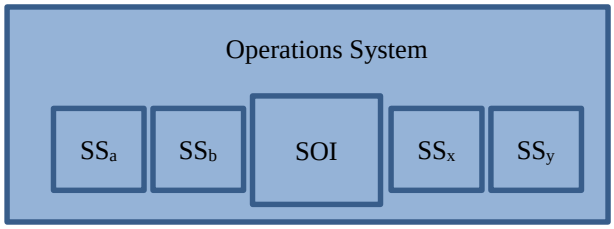

Figure 12: System of Interest SOI as Part of the Operations System

As illustrated in Figure 12, the operations system, where accidents can happen and operational safety risk is assessed, consists of a number of subsystems (SS) including our computer-based system of interest (SOI). The SOI requirements, including functional, non-functional, and design constraints, are defined taking into consideration the contribution of the SOI to safety risk in the operations system. Henceforth it is assumed that the *intended* (though not necessarily the *actual*) SOI functional and performance requirements are safe in the sense that the SOI will not cause a mishap if its delivered service is in compliance with the intended requirements. SOI safety then is about (service) **failure** events in which the SOI fails to meet its **intended** service and fails to deliver service or delivers an **unintended** service (i.e., the delivered service deviates from the intended one). An SOI failure is a hazard in the operations system. The non-functional safety attributes (or qualities) of the SOI capture the safety risk requirements covering the failure modes (i.e., the hazards) and maximum frequency of occurrence for each. As illustrated in Figure 13, the system safety design problem is how to design the SOI to deliver the required functions with the specified safety-related quality attributes under stated threat conditions. The threats are the causes of SOI failures and can be endogenous or exogenous to the SOI. As stated previously, the system architecture is the high-level strategy to achieve the goals of the system (i.e., the requirements). These three concepts of system architectures, safety-relevant qualities, and safety-relevant threats will be examined further in the future sections of the design guide.

Additional information on safety, system safety, risk, and safety-relevant system properties for systems and software is available from the sources referenced above as well as the following documents: [28], [125], [126], [127], [128], [129], [130], and [131].

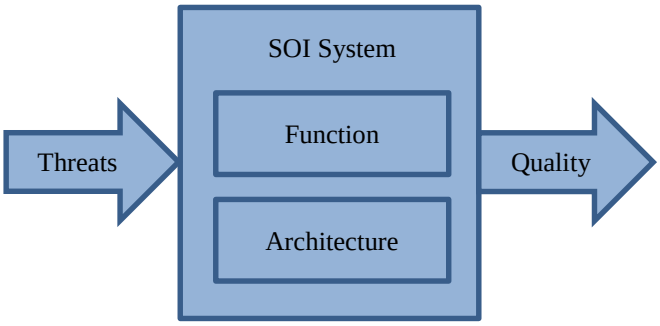

Figure 13: Elements of the System Safety Design Problem

3. Final Remarks

This document presented an overview of system design, lifecycle, and safety. The document is intended to be part of larger design guide for safety-relevant computer-based systems. The next report in this series will address the topic of system-safety threats.

References

[1] S. C. Beland, "Assuring a Complex Safety-Critical Systems of Systems," *AeroTech Congress & Exhibition, SAE Technical Paper 2007-01-3872,* September 2007.

[2] P. De Montalk, "Computer Software in Civil Aircraft," in *Sixth Annual Conference on Computer Assurance (COMPASS'91)*, Gaithersburg, MD, 1991.

[3] J. Lewis and L. Rierson, "Certification Concerns With Integrated Modular Avionics (IMA) Projects," in *The 22nd Digital Avionics Systems Conference (DASC'03)*, 2003.

[4] C. M. Fuchs, "The Evolution of Avionics Networks From ARINC 429 to AFDX," in *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), and Aerospace Networks (AN), Summer Semester 2012*, Munich, Germany, 2012.

[5] R. Garside and F. J. Pighetti, Jr., "Integrating Modular Avionics: A New Role Emerges," in *IEEE/AIAA 26th Digital Avionics Systems Conference*, Dallas, Texas, 2007.

[6] P. Bieber, F. Boniol, M. Boyer, E. Noulard and C. Pagetti, "New Challenges for Future Avionic Architectures," *Aerospace Lab Journal,* no. 4, pp. AL04-11 1 - 10, May 2012.

[7] C. B. Watkins and R. Walter, "Transitioning From Federated Avionics Architectures to Integrated Modular Avionics," in *IEEE/AIAA 26th Digital Avionics Systems Conference (DASC'07)*, Dallas, Texas, 2007.

[8] C. B. Watkins, "Integrated Modular Avionics: Managing the Allocation of Shared Intersystem Resources," in *IEEE/AIAA 25th Digital Avionics Systems Conference (DASC'07)*, Portland, Oregon, 2006.

[9] H. Butz, "The Airbus Approach to Open Integrated Modular Avionics (IMA): Technology, Methods, Processes and Future Road Map," in *Proceedings of the 1st International Workshop on Aircraft Systems Technologies (AST 2007)*, Hamburg, Germany, 2007.

[10] B. Witwer, "System Integration of the 777 Airplane Information Management System," *IEEE Aerospace and Electronic Systems Magazine,* vol. 11, no. 4, pp. 17 - 21, April 1996.

[11] J. B. Itier, "A380 Integrated Modular Avionics," ARTIST Meeting on Integrated Modular Avionics, Rome, Italy, 2007.

[12] R. Wolfig and M. Jakovljevic, "Distributed IMA and DO-297: Architectural, Communication and Certification Attributes," in *IEEE/AIAA 27th Digital Avionics Systems Conference (DASC'08)*, St. Paul, MN, 2008.

[13] C. S. Tang, J. D. Zimmerman and J. I. Nelson, "Managing New Product Development and Supply Chain Risks: The Boeing 787 Case," *Supply Chain Forum: An International Journal,* Vols. 10, no. 2, no. 19, pp. 74 - 86, 2009.

[14] J. Moore, "Advanced Distributed Architectures," in *Avionics: Development and Implementation*, C. R. Spitzer, Ed., CRC Press, 2007.

[15] R. J. Bleeg, "Commercial Jet Transport Fly-By-Wire Architecture Considerations," in *9th AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 1988.

[16] J. Hasson and D. Crotty, "Boeing's Safety Assessment Processes for Commercial Airplane Designs," in *16th Digital Avionics Systems Conference (DASC)*, 1997.

[17] Y. C. Yeh, "Triple-Triple Redundant 777 Primary Flight Computer," in *Proceedings of the 1996 Aerospace Applications Conference*, 1996.

[18] Y. C. Yeh, "Design Considerations in Boeing 777 Fly-By-Wire Computers," in *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium*, 1998.

[19] RTCA, Inc., *Environmental Conditions and Test Procedures for Airborne Equipment (RTCA DO-160G)*, 2010.

[20] N. Storey, Safety-Critical Computer Systems, Addison Wesley, 1996.

[21] J. C. Knight, "Safety Critical Systems: Challenges and Directions," in *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*, 2002.

[22] Federal Aviation Administration, *Advisory Circular: System Design and Analysis (AC No. 25.1309-1A),* 1988.

[23] A. Avizienis, J. C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing,* Vols. 1, No. 1, pp. 11 - 33, January - March 2004.

[24] Federal Aviation Admistration, Certification Authorities Software Team (CAST), *Reliance on Development Assurance Alone when Performing a Complex and Full-Time Critical Function. Position Paper CAST-24,* 2006.

[25] SAE International, *Aerospace Recommended Practice: Guidelines for the Development of Civil Aircraft and Systems (ARP-4754), Revision A,* 2010.

[26] A. Ranganathan and R. H. Campbell, *What is the Complexity of a Distributed System?,* Vols. 12, Number 6, Complexity, 2007, pp. 37 - 45.

[27] J. J. Chilenski and J. L. Kurtz, *Object-Oriented Technology Verification Phase 2 Report - Data Coupling and Control Coupling,* Federal Aviation Administration, Report DOT/FAA/AR-07/52, 2007.

[28] J. Rushby, "Critical System Properties: Survey and Taxonomy," SRI International, Technical Report SRI-CSL-93-1, 1993.

[29] G. Bartley and B. Lingberg, "Certification Concerns of Integrated Modular Avionics (IMA) Systems," in *27th Digital Avionics Systems Conference*, 2008.

[30] R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering,* vol. 19, no. 1, pp. 3 - 12, January 1993.

[31] D. Jackson, M. Thomas and L. I. Millett, Eds., *Software for Dependable Systems: Sufficient Evidence?,* Committee on Certifiably Dependable Software Systems; National Research Council, 2007.

[32] RTCA, Inc., *Software Considerations in Airborne Systems and Equipment Certification (RTCA/DO-178C),* 2012.

[33] RTCA, Inc., *Design Assurance Guidance for Airborne Electronic Hardware (RTCA/DO-254),* 2000.

[34] N. G. Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995.

[35] N. G. Leveson, Engineering a Safer World: Systems Thinking Applied to Safety, MIT Press, 2011.

[36] National Research Council, Steering Committee for the Decadal Survey of Civil Aeronautics, *Decadal Survey of Civil Aeronautics: Foundation for the Future,* 2006.

[37] *National Aeronautics Research and Development Plan,* Office of Science & Technology Policy, 2010.

[38] D. C. Winter, *Statement before a hearing on Networking and Information Technology Research and Development (NITRD) Program,* Committee on Science and Technology, U.S. House of Representatives, 2008.

[39] A. Mozdzanowska and R. J. Hansman, *System Transition: Dynamics of Change in the US Air Transportation System,* International Center for Air Transportation, Massachusetts Institute of Technology, Report Number ICAT-2008-3, 2008.

[40] Joint Planning and Development Office (JPDO), *NextGen Integrated Development Plan, Version 1.0,* 2008.

[41] Aerospace Vehicle Systems Institute (AVSI), *AFE #58 Summary Final Report,* System Architecture Virtual Integration (SAVI) Program, Document SAVI-58-00-01, version 1, 2009.

[42] D. M. Buede, The Engineering Design of Systems: Models and Methods, John Wiley & Sons, Inc., 2009.

[43] A. Avizienis, "The Four-Universe Information System Model for the Study of Fault Tolerance," in *Proceedings of the 12th Annual International Symposium on Fault-Tolerant Computing,* Santa Monica, California, 1982.

[44] B. Parhami, "A Multi-Level View of Dependable Computing," *Computers and Electrical Engineering,* vol. 20, no. 4, pp. 347 - 368, July 1994.

[45] Z. Kohavi, Switching and Finite Automata Theory, 2nd ed., McGraw-Hill Book Company, 1978.

[46] T. A. Sudkamp, Languages and Machines: An Introduction to the Theory of Computer Science, Addison-Wesley, 1997.

[47] T. A. Standish, Data Structures, Algorithms, and Software Principles in C, Addison-Wesley Publishing Company, 1995.

[48] W. I. Fletcher, An Engineering Approach to Digital Design, Prentice-Hall, Inc., 1980.

[49] N. H. E. Weste and K. Eshraghian, Principles of CMOS VLSI Design: A Systems Perspective, Addison-Wesley Publishing Company, 1993.

[50] C. H. Chen, Ed., Computer Engineering Handbook, McGraw-Hill, Inc., 1992.

[51] D. D. Gajski, Principles of Digital Design, Prentice-Hall, Inc., 1997.

[52] I. Moir, A. Seabridge and M. Jukes, Civil Avionics Systems, 2nd ed., John Wiley & Sons, Ltd., 2013.

[53] J. R. Armstrong and F. G. Gray, Structured Logic Design with VHDL, Prentice-Hall, Inc., 1993.

[54] G. Nutt, Operating Systems: A Modern Perspective, Addison Wesley, 1997.

[55] D. D. Gajski, F. Vahid, S. Narayan and J. Gong, Specification and Design of Embedded Systems, Prentice Hall, 1994.

[56] C.-T. Chen, Linear System Theory and Design, Holt, Rinehart and Winston, 1984.

[57] R. Obermaisser, Event-Triggered and Time-Triggered Control Paradigms, Springer Science + Business Media, Inc., 2005.

[58] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, 2nd ed., Springer Science + Business Media, 2011.

[59] A. v. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, John Wiley & Sons Ltd, 2009.

[60] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558 - 565, 1978.

[61] W. Torres-Pomales, M. R. Malekpour and P. S. Miner, "ROBUS-2: A Fault-Tolerant Broadcast Communication System," NASA Langley Research Center, Techical Report NASA/TM-2005-213540 , 2005.

[62] H. Kopetz, "Should Responsive Systems be Event-Triggered or Time-Triggered?," *Institute of Electronics, Information, and Communications Engineers Transactions on Information and Systems,* Vols. E76-D, no. 11, pp. 1325-1332, 1993.

[63] J. J. Scarlett and R. W. Brennan, "Re-evaluating Event-Triggered and Time-Triggered Systems," in *2006 IEEE Conference on Emerging Technologies and Factory Automation (ETFA '06)*, Prague, Czech Republic, 2006.

[64] A. Albert, "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems," in *Embedded World 2004*, 2004.

[65] R. N. Mahapatra and S. Ahmad, "Microprocessor Evaluations for Applications: Authority for Expenditures no. 43 Phase 1 Report," Federal Aviation Administration; DOT/FAA/AR-06/34, 2006.

[66] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, T. Mitra, F. Mueller, I. Puat-Irisa, P. Puschner, J. Staschulat and P. Stenstrom, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 7, no. 3, pp. 36:1 - 36:53, 2008.

[67] J. G. Proakis and D. G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, 2nd ed., Macmillan Publishing Company, 1992.

[68] R. L. Burden and J. D. Faires, Numerical Analysis, Third ed., PWS Publishers, 1985.

[69] S. Brilliant, J. C. Knight and N. G. Leveson, "The Consistent Comparison Problem in N-Version Software," *IEEE Transactions on Software Engineering,* vol. 15, no. 11, pp. 1481 - 1485, 1989.

[70] S. Poledna, Fault-Tolerant Real-Time Systems: The Problem of Replica Determinisn, Kluwer Academic Publishers, 1996.

[71] L. L. Pullum, Software Fault Tolerance Techniques and Implementation, Artech House, Inc., 2001.

[72] B. Hall and K. Driscoll, "Distributed System Design Checklist," NASA Langley Research Center, NASA/CR-2014-218504, 2014.

[73] W. Stallings, Data and Computer Communications, 4th ed., Macmillan Publishing Company, 1994.

[74] H. V. Bertine, "Physical Level Protocols," *IEEE Transactions on Communications,* Vols. COM-28, no. 4, pp. 433 - 444, 1980.

[75] C. F. Coombs, Jr., Printed Circuits Handbook, 4th ed., The McGraw-Hill Companies, Inc., 1996.

[76] Aeronautical Radio, Inc., ARINC Specification 650: Integrated Modular Avionics Packaging and Interfaces, 1994.

[77] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming,* vol. 8, no. 3, pp. 231 - 274 , 1987.

[78] I. Bate, R. Hawkins and J. McDermid, "A Contract-based Approach to Designing Safe Systems," in *8th Australian Workshop on Safety Critical Systems and Software (SCS'03)*, Canberra, Australia, 2003.

[79] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger and K. Larsen, "Constract for Systems Design," Inria, 2012.

[80] B. S. Blanchard, System Engineering Management, John Wiley & Sons, Inc., 2008.

[81] National Aeronautics and Space Administration (NASA), "NASA Systems Engineering Handbook (NASA/SP-2007-6105, Revision 1)," 2007.

[82] P. T. Hester, J. M. Bradley and K. M. Adams, "Stakeholders in systems problems," *International Journal of System of Systems Engineering,* vol. 3, no. 3/4, pp. 225 - 232, 2012.

[83] M. C. Jackson, Systems Thinking: Creative Holism for Managers, John Wiley & Sons, 2003.

[84] J. E. Gibson, W. T. Scherer and W. F. Gibson, How to Do Systems Analysis, John Wiley & Sons, Inc., 2007.

[85] The MITRE Corporation, "Systems Engineering Guide," 2014.

[86] A. Kossiakoff, W. N. Sweet, W. N. Seymour, S. J. Seymour and S. M. Biemer, Systems Engineering Principles and Practice, 2nd ed., John Wiley & Sons, Inc., 2011.

[87] Institute of Electrical and Electronics Engineers, *IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document (IEEE Std 1362-1998 R2007),* 1998.

[88] B. Langer and M. Tautschnig, "Navigating the Requirements Jungle," in *Leveraging Applications of Formal Methods, Verification and Validation*, T. Margaria and B. Steffen, Eds., Springer, 2008, pp. 354 - 368.

[89] Federal Aviation Administration, "Requirements Engineering Management Handbook (DOT/FAA/AR-08/32)," U.S. Department of Transportation (DOT), Federal Aviation Administration (FAA), 2009.

[90] R. K. Kandt, "Software Requirements Engineering: Practices and Techniques (JPL Document D-24994))," Jet Propulsion Laboratory (JPL), California Institute of Technology, 2003.

[91] IEEE Computer Society, "Guide to the Software Engineering Body of Knowledge (SWEBOK V3.0)," 2014.

[92] Department of Defense, Systems Management College, "Systems Engineering Fundamentals," 2001.

[93] Institute of Electrical and Electronics Engineering (IEEE), *IEEE 1471-2000: Systems and software engineering - Recommended practice for architectural description of software-intensive systems,* 2000.

[94] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, 3rd ed., Addison-Wesley, 2013.

[95] F. Bachmann, L. Bass, P. Clemens, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford, "Documenting Software Architecture: Documenting Behavior," Software Engineering Institute, 2002.

[96] N. Storey, "Design for Safety," in *Towards System Safety: Proc. 7th Safety-Critical Systems Symposium*, Huntingon, UK, 1999.

[97] D. Garlan and M. Shaw, "An Introduction to Software Architecture," Carnegie Mellon University (CMU), Software Engineering Institute (SEI), 1994.

[98] D. Bertsekas and R. Galager, Data Networks, Prentice-Hall, Inc., 1987.

[99] D. Cofer, "Complexity-Reducing Design Patterns for Cyber-Physical Systems," Department of Defense, 2011.

[100] R. S. Hanmer, Patters for Fault Tolerant Software, John Wiley & Sons Ltd, 2007.

[101] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, 2005.

[102] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, pp. 19 - 26, March/April 2005.

[103] R. Malan and D. Bredemeyer, "Less is More with Minimalistic Architecture," *IT Professional*, pp. 48, 46-47, September/October 2002.

[104] M. W. Maier and E. Rechtin, The Art of Systems Architecting, Second ed., CRC Press, 2000.

[105] The Trustees of the Stevens Institute of Technology, "Guide to the Systems Engineering Body of Knowledge (SEBoK), version 1.1," Hoboken, NJ, 2013.

[106] J. Cleland-Huang, R. S. Hanmer, S. Supakkul and M. Mirakhorli, "The Twin Peaks of Requirements and Architecture," *IEEE Software*, vol. 30, no. 2, pp. 24 - 29, March-April 2013.

[107] P. J. Koopman, Jr., "A Taxonomy of Decomposition Strategies Based on Structures, Behaviors, and Goals," in *1995 Design Theory and Methodology Conference*, Boston, MA, 1995.

[108] B. Boehm, "Some Future Trends and Implications for Systems and Software Engineering Processes," *Systems Engineering*, vol. 9, no. 1, pp. 1 - 19, 2006.

[109] R. Bate, D. Kuhn, C. Wells, J. Armitage, G. Clark, K. Cusick, S. Garcia, M. Hanna, R. Jones, P. Malpass, I. Minnich, H. Pierson, T. Powell and A. Reichner, "A Systems Engineering Capability Maturity Model, Version 1.1," Carnegie Mellon University, Software Engineering Institute, 1995.

[110] P. H. Feiler, "Challenges in Validating Safety-Critical Embedded Systems," *SAE International Journal of Aerospace*, vol. 3, no. 1, pp. 109 - 116, 2009.

[111] International Council of Systems Engineering (INCOSE), Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 2003.

[112] G.-S. Chang, H.-L. Perng and J.-N. Juang, "A Review of Systems Engineering Standards and Processes," *Journal of Biomechatronics Engineering*, vol. 1, no. 1, pp. 71 - 85, 2008.

[113] "EIA-632: Processes for Engineering a System," SAE International, 2003.

[114] B. J. Sauser, E. Forbes, M. Long and S. E. McGrory, "Defining an Integration Readiness Level for Defense Acquisition," in *International Symposium of the International Council on Systems Engineering (INCOSE)*, Singapore, 2009.

[115] B. Sauser, J. E. Ramirez-Marquez, R. Magnaye and W. Tan, "A Systems Approach to Expanding the Technology Readiness Level within Defense Acquisition," *International Journal of Defense Acquisition Management*, vol. 1, pp. 39 - 58, 2008.

[116] B. Sauser, Ph.D., J. Ramirez-Marquez, Ph.D., D. Verma, Ph.D. and R. Gove, "From TRL to SRL: The Concept of Systems Readiness Levels," in *Conference on Systems Engineering Research*, Los Angeles, CA, 2006.

[117] E. R. Goossen and D. A. Buster, "Regulatory Compliance in Multi-Tier Supplier Networks (NASA/CR-2014-218550)," NASA Langley Research Center, 2014.

[118] Department of Defense, "Military Handbook: Configuration Management Guidance (MIL-HDBK-61A(SE))," 1997.

[119] National Aeronautics and Space Administration (NASA), "NASA System Safety Handbook: Volume 1, System Safety Framework and Concepts for Implementation (NASA/SP-2010-580, Version 1.0)," 2011.

[120] Department of Defense, "System Safety (MIL-STD-882E)," 2012.

[121] C. A. I. Ericson, Concise Encyclopedia of System Safety: Definition of Terms and Concepts, John Wiley & Sons, Inc., 2011.

[122] Federal Aviation Administration (FAA), "FAA System Safety Handbook," 2000.

[123] E. Lloyd and W. Tye, Systematic Safety: Safety Assessment of Aircraft Systems, London, England: Civil Aviation Authority, 1982.

[124] D. J. Pumfrey, "The Principled Design of Computer System Safety Analyses (Ph.D. Dissertation)," University of York, 1999.

[125] W. R. Dunn, "Designing Safety-Critical Computer Systems," *Computer*, pp. 40 - 46, November 2003.

[126] N. Moller and S. O. Hansson, "Principles of Engineering Safety: Risk and Uncertainty Reduction," *Reliability Engineering and System Safety*, vol. 93, no. 6, pp. 798 - 805, 2008.

[127] E. Hollnagel, "Risk + Barriers = Safety?," *Safety Science*, vol. 46, no. 2, pp. 221 - 229, February 2008.

[128] Health & Safety Executive, Health & Safety Commission: Study Group on the Safety of Operational Computer Systems, "The Use of Computers in Safety-Critical Application," United Kingdom, 1998.

[129] National Aeronautics and Space Administration (NASA), "NASA Risk Management Handbook (NASA/SP-2011-3422, Version 1.0)," 2011.

[130] Federal Aviation Administration (FAA), "Risk Management Handbook (FAA-H-8083-2)," 2009.

[131] National Aeronautics and Space Administration (NASA), "NASA Software Safety Guidebook (NASA-GB-8719.13)," 2004.

[132] A. S. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigms, 2nd ed., Pearson Prentice Hall, 2006.

[133] Aeronautical Radio, Inc., "ARINC Specification 653 P1-3: Avionics Application Software Software Standard Interface Part 1 - Required Services," 2010.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
01- 05 - 2015

2. REPORT TYPE
Technical Memorandum

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
Overview of Design, Lifecycle, and Safety for Computer-Based Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Torres-Pomales, Wilfredo

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
999182.02.50.07.02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-20552

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA-TM-2015-218764

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 62
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This document describes the need and justification for the development of a design guide for safety-relevant computer-based systems. This document also makes a contribution toward the design guide by presenting an overview of computer-based systems design, lifecycle, and safety.

15. SUBJECT TERMS
Architecture; Computers; Lifecycle; Requirement; Safety; Systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	42	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18