DOT/FAA/AR-08/32

Air Traffic Organization
NextGen & Operations Planning
Office of Research and
Technology Development
Washington, DC 20591

Requirements Engineering Management Handbook

June 2009

Final Report

U.S. Department of Transportation
Federal Aviation Administration

NOTICE

This report is available at the Federal Aviation Administration William J. Hughes Technical Center’s Full-Text Technical Reports page: actlibrary.tc.faa.gov in Adobe Acrobat portable document format (PDF).

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/AR-08/32		

4. Title and Subtitle	5. Report Date
REQUIREMENTS ENGINEERING MANAGEMENT HANDBOOK	June 2009
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
David L. Lempia and Steven P. Miller	

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
Rockwell Collins, Inc. 400 Collins Road NE Cedar Rapids, Iowa 52245	11. Contract or Grant No. DTFACT-05-C-00004

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
U.S. Department of Transportation Federal Aviation Administration Air Traffic Organization NextGen & Operations Planning Office of Research and Technology Development Washington, DC 20591	Final Report
	14. Sponsoring Agency Code AIR-120

15. Supplementary Notes

The Federal Aviation Administration Airport and Aircraft Safety R&D Division COTR was Charles Kilgore.

16. Abstract

This Handbook presents a set of recommended practices on how to collect, write, validate, and organize requirements. It attempts to bring together the best ideas from several approaches, organize them into a coherent whole, and illustrate them with concrete examples that make their benefits clear.

The Handbook is targeted to the domain of real-time, embedded systems and specifically to the avionics industry. It describes a set of recommended practices in which basic concepts can be practiced in isolation, but reinforce each other when practiced as a whole. These practices allow developers to progress from an initial, high-level overview of a system to a detailed description of its behavioral and performance requirements. Due to the growing importance of software in avionics systems, these practices emphasize techniques to ease the transition from system to software requirements.

Concrete examples are used throughout the Handbook to make the concepts clear, but there are many other formats that could be used to obtain the same objectives. It is expected that most organizations wanting to use these practices will want to modify them, perhaps significantly, to integrate them with their existing processes and tools.

17. Key Words	18. Distribution Statement
Requirements, Engineering, Avionics, Systems, Software	This document is available to the U.S. public through the National Technical Information Service (NTIS) Springfield, Virginia 22161.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	146	

TABLE OF CONTENTS

		Page
EXECUTIVE SUMMARY		xi
1.	INTRODUCTION	1
1.1	Purpose	1
1.2	Background	2
2.	RECOMMENDED PRACTICES	3
2.1	Develop the System Overview	4
2.1.1	Develop System Overview Early	5
2.1.2	Provide System Synopsis	6
2.1.3	Identify System Contexts	6
2.1.4	Use Context Diagrams	7
2.1.5	Describe External Entities	7
2.1.6	Capture Preliminary System Goals	7
2.1.7	Maintain System Goal Information	8
2.2	Identify the System Boundary	9
2.2.1	Identify the System Boundary Early	10
2.2.2	Choose Environmental Variables	11
2.2.3	Choose Controlled Variables	12
2.2.4	Choose Monitored Variables	12
2.2.5	Ensure Environmental Variables are Sufficiently Abstract	12
2.2.6	Avoid Presentation Details in Environmental Variables	12
2.2.7	Define All Physical Interfaces	13
2.3	Develop the Operational Concepts	14
2.3.1	Document Sunny Day System Behavior	16
2.3.2	Include How the System is Used in its Operating Environment	17
2.3.3	Employ the Use Case Goal as its Title	18
2.3.4	Trace Each Use Case to System Goals	18
2.3.5	Identify Primary Actor, Preconditions, and Postconditions	18
2.3.6	Ensure Each Use Case Describes a Dialogue	18

2.3.7 Link Use Case Steps to System Functions 19

2.3.8 Consolidate Repeated Actions Into a Single Use Case 19

2.3.9 Describe Exceptional Situations as Exception Cases 19

2.3.10 Describe Alternate Ways to Satisfy Postconditions as Alternate Courses 19

2.3.11 Use Names of External Entities or Environmental Variables 20

2.3.12 Avoid Operator Interface Details 20

2.3.13 Update the System Boundary 20

2.3.14 Assemble a Preliminary Set of System Functions 21

2.4 Identify the Environmental Assumptions 22

2.4.1 Define the Type, Range, Precision, and Units 23
2.4.2 Provide Rationale for the Assumptions 24
2.4.3 Organize Assumptions Constraining a Single Entity 24
2.4.4 Organize Assumptions Constraining Several Entities 25
2.4.5 Define a Status Attribute for Each Monitored Variable 26
2.4.6 Summary 27

2.5 Develop the Functional Architecture 27

2.5.1 Organize System Functions Into Related Groups 28
2.5.2 Use Data Flow Diagrams to Depict System Functions 29
2.5.3 Minimize Dependencies Between Functions 30
2.5.4 Define Internal Variables 31
2.5.5 Nest Functions and Data Dependencies for Large Specifications 31
2.5.6 Provide High-Level Requirements That are Really High Level 32
2.5.7 Do Not Incorporate Rationale Into the Requirements 33

2.6 Revise the Architecture to Meet Implementation Constraints 33

2.6.1 Modify the Architecture to Meet Implementation Constraints 34

2.6.2 Keep Final System Architecture Close to Ideal Functional Architecture 35

2.6.3 Revise the System Overview 35

2.6.4 Revise the Operational Concepts 39

2.6.5 Develop Exception Cases 39

2.6.6 Link Exception Cases to Use Cases 40

2.6.7 Revise the System Boundary 40

2.6.8 Document Changes to Environmental Assumptions 40

2.6.9 Revise Dependency Diagrams 40

2.6.10 Revise High-Level Requirements 42

2.7 Identify the System Modes 42

2.7.1 Identify Major System Modes 44
2.7.2 Define How System Transitions Between Modes 44
2.7.3 Introduce Modes for Externally Visible Discontinuities 45

2.8 Develop the Detailed Behavior and Performance Requirements 45

2.8.1 Specify the Behavior of Each Controlled Variable 47
2.8.2 Specify the Requirement as a Condition and an Assigned Value 47
2.8.3 Ensure That Detailed Requirements are Complete 47
2.8.4 Ensure That Detailed Requirements are Consistent 49
2.8.5 Ensure That Detailed Requirements are not Duplicated 49
2.8.6 Organize the Requirements 49
2.8.7 Define Acceptable Latency for Each Controlled Variable 49
2.8.8 Define Acceptable Tolerance for Each Controlled Variable 50
2.8.9 Do Not Define Latency and Tolerance for Internal Variables 50
2.8.10 Alternative Ways to Specify Requirements 51

2.9 Define the Software Requirements 52

2.9.1 Specify the Input Variables 56
2.9.2 Specify the Accuracy of Each Input Variable 57
2.9.3 Specify the Latency of Each Input Variable 57
2.9.4 Specify IN' for Each Monitored Variable 57
2.9.5 Specify the Status of Each Monitored Variable 58
2.9.6 Flag Design Decisions as Derived Requirements 59
2.9.7 Specify the Output Variables 59
2.9.8 Specify the Latency of Each Output Variable 60
2.9.9 Specify the Accuracy of Each Output Variable 60
2.9.10 Specify OUT' for Each Controlled Variable 61
2.9.11 Confirm Overall Latency and Accuracy 61

2.10 Allocate System Requirements to Subsystems 63

2.10.1 Identify Subsystem Functions 65

2.10.2 Duplicate Overlapping System to Subsystem Functions 67

2.10.3 Develop a System Overview for Each Subsystem 69

2.10.4 Identify the Subsystem Monitored and Controlled Variables 69

2.10.5 Create New Monitored and Controlled Variables 69

2.10.6 Specify the Subsystem Operational Concepts 70

2.10.7 Identify Subsystem Environmental Assumptions Shared With Parent System 70

2.10.8 Identify Environmental Assumptions of the New Monitored and Controlled Variables 70

2.10.9 Complete the Subsystem Requirements Specification 71

2.10.10 Ensure Latencies and Tolerances are Consistent 71

2.11 Provide Rationale 72

2.11.1 Provide Rationale to Explain why a Requirement Exists 73
2.11.2 Avoid Specifying Requirements in the Rationale 73
2.11.3 Provide Rationale When the Reason a Requirement is not Obvious 74
2.11.4 Provide Rationale for Environmental Assumptions 74
2.11.5 Provide Rationale for Values and Ranges 75
2.11.6 Keep Rationale Short and Relevant 75
2.11.7 Capture Rationale as Soon as Possible 75

3. SUMMARY 76

4. REFERENCES 77

APPENDICES

A—Isolette Thermostat Example
B—Flight Control System Example
C—Flight Guidance System Example
D—Autopilot Example

LIST OF FIGURES

Figure		Page
1	The System and its Environment	10
2	Example Use Case	17
3	Thermostat Dependency Diagram	30
4	High-Level Requirements for the Thermostat Function	32
5	Initial Isolette Fault Tree	36
6	Revised Isolette Fault Tree	37
7	Revised Thermostat Dependency Diagram	38
8	Regulate Temperature Dependency Diagram	41
9	Monitor Temperature Dependency Diagram	42
10	Regulate Temperature Function Modes	44
11	The Four-Variable Model	54
12	Extended Software Requirements	55
13	High- and Low-Level Software Requirements	62
14	Functional Decomposition of System 1	65
15	Decomposition of System 1 Into Subsystems	66
16	Allocation of FCS Requirements Into Subsystems	68

LIST OF TABLES

Table		Page
1	Develop the System Overview	5
2	Identify the System Boundary	9
3	Thermostat Monitored and Controlled Variables	11
4	Develop the Operational Concepts	14
5	Revised Thermostat Monitored and Controlled Variables	21
6	Preliminary Set of Isolette Thermostat Functions	21
7	Identify the Environmental Assumptions	22
8	Environmental Assumptions for the Current Temperature Monitored Variable	23
9	Develop the Functional Architecture	27
10	Revise the Architecture to Meet Implementation Constraints	33
11	Identify the System Modes	43
12	Definition of Regulator Status	45
13	Develop Detailed Behavior and Performance Requirements	46
14	Allowed Heat Source Latency Behavior	50
15	Tabular Specification of Requirements	51
16	Define the Software Requirements	52
17	Input Variable Curr Temp In	56
18	INPUT Variable Curr Temp Status In	58
19	IN' Relation for Value of Current Temperature'	58
20	IN' Relation for Status of Current Temperature'	59
21	OUTPUT Variable Heat Control OUT	60
22	OUT' Relation for Heat Control	61
23	Allocate System Requirements to Subsystems	63
24	Provide Rationale	72

LIST OF ACRONYMS AND ABBREVIATIONS

AP	Autopilot
AHS	Attitude Heading System
ARINC	Aeronautical Radio, Incorporated
ARP	Aerospace Recommended Practice
CoRE	Consortium Requirements Engineering
CSA	Controlled Surface Actuators
FCI	Flight Crew Interface
FCS	Flight Control System
FD	Flight Director
FG	Flight Guidance
FGS	Flight Guidance System
FHA	Functional Hazard Assessment
HMI	Human Machine Interface
msec	millisecond
N/A	Not Applicable
PFD	Primary Flight Display
PSSA	Preliminary System Safety Assessment
REM	Requirements Engineering Management
RSML	Requirements State Machine Language
$RSML^{-e}$	Requirements State Machine Language without events
SCR	Software Cost Reduction
SpecTRM	Specification Toolkit and Requirements Methodology

EXECUTIVE SUMMARY

This Handbook presents a set of recommended practices on how to collect, write, validate, and organize requirements. It attempts to bring together the best ideas from several approaches, organize them into a coherent whole, and illustrate them with concrete examples that make their benefits clear.

The Handbook is targeted to the domain of real-time, embedded systems and specifically to the avionics industry. It describes a set of recommended practices in which basic concepts can be practiced in isolation, but reinforce each other when practiced as a whole. These practices allow developers to progress from an initial, high-level overview of a system to a detailed description of its behavioral and performance requirements. Due to the growing importance of software in avionics systems, these practices emphasize techniques to ease the transition from system to software requirements.

Concrete examples are used throughout the Handbook to make the concepts clear, but there are many other formats that could be used to obtain the same objectives. It is expected that most organizations wanting to use these practices will want to modify them, perhaps significantly, to integrate them with their existing processes and tools.

1. INTRODUCTION.

This research was undertaken in response to the Requirements Engineering Management (REM) solicitation posted by the Federal Aviation Administration William J. Hughes Technical Center. The goal of this task was to determine methods that enable successful management, control, integration, verification, and validation of system and software requirements that may be developed by multiple entities.

1.1 PURPOSE.

This Handbook presents a set of recommended practices on how to collect, write, validate, and organize requirements. It attempts to bring together the best ideas from several approaches, organize them into a coherent whole, and illustrate them with concrete examples that make their benefits clear.

The literature on requirements engineering is vast, and practices vary widely even within a particular industry. For this reason, the Handbook is targeted to the domain of real-time, embedded systems and specifically to the avionics industry. Due to the rapidly growing importance of software in these systems, it emphasizes practices that ease the transition from system to software requirements.

The main-level recommended practices are presented roughly in the order they would be performed on a program, but there is no requirement that this order must be strictly adhered to. As with most processes, significant iteration between the different activities is expected as the requirements are refined. Rather than trying to specify a detailed process, the Handbook focuses on identifying what information is needed in a requirements specification and providing recommendations on how that information can be collected and organized.

To make these ideas concrete, two examples are used to illustrate the recommended practices. The first, a thermostat for an Isolette used in a neonatal care unit provides a small, easily understood example that illustrates application of the recommended practices on a single system. The second, a very simple Flight Control System (FCS), Flight Guidance System (FGS), and Autopilot (AP) illustrates how requirements could be allocated from a system to its subsystems to facilitate development by multiple subcontractors. Both examples are meant to illustrate the recommended practices and are not intended as complete examples of actual systems. Both examples are referred to throughout the discussion of the recommended practices. Their specifications are presented in the appendices.

While the examples might appear to suggest a specific style and format, their true purpose is to illustrate and clarify the recommended practices and their benefits. There are many different formats that could be used to satisfy the same objectives, and most organization wanting to use any of the recommended practices will need to modify them, perhaps significantly, to fit with their existing processes and tools.

1.2 BACKGROUND.

The recommended practices are based on the results of an industry survey and a literature search, both of which are documented in reference 1. The industry survey provided a useful overview of the current state of practice and identified many of the issues and concerns of the developers working in the avionics domain. The literature search identified several methodologies for REM that have been successfully applied to avionics systems and address many of the concerns raised in the survey. Much of the challenge in developing this Handbook consisted of finding ways for organizations to integrate these methodologies into their existing practices.

The practices described in this Handbook draw strongly on the Software Cost Reduction (SCR) methodology and the four-variable model originally proposed by Parnas and Madey [2] for specifying the requirements of the U.S. Navy's A-7E Corsair II aircraft [3]. Later, these ideas were extended by the Software Productivity Consortium into the Consortium Requirements Engineering (CoRE) methodology [4 and 5], which was used to specify the requirements for the C-130J aircraft [6]. Many of the recommended practices on how to organize the requirements are based on ideas originally developed with CoRE methodology. SCR also continued to evolve over the past two decades. The Naval Research Laboratory developed a number of tools for the specification and analysis of SCR models [7].

Another important source of best practices was the extensive work done by Leveson and her colleagues in developing the Requirements State Machine Language (RSML) [8] used to specify the Traffic Alert and Collision Avoidance System II [9]. This approach was later extended into the RSML without events ($RSML^{-e}$) [10] and used as the basis for the formal analysis of requirements specifications [11 and 12]. More recently, this notation was extended into the Specification Toolkit and Requirements Methodology (SpecTRM) developed by the Safeware Engineering Corporation [13, 14, and 15]. SpecTRM models are embedded within a larger intent specification that "provides a seamless flow of rationale and reasoning from the highest level goals of the system, down through the SpecTRM model all the way down to the implementation and operator training materials" [13, 14, and 16].

A great deal of research has been done over the last two decades to specify requirements as use cases. In particular, use cases appear to be an excellent technique for transitioning from the initial, informal system overview to the detailed, formal specification of the requirements. The discussion on operational concepts in this Handbook is based on the use of textual use cases similar to those described in references 17 through 21.

A good overall guide to REM is found in reference 22, which shares many similarities with the approach outlined in this Handbook. It provides many excellent recommendations on the REM process, descriptions of what makes a good requirement, and example checklists. Another well-known reference is 23. Leveson's text on system and software safety also discusses REM in detail [24].

Finally, the reader should be aware of a number of standards related to REM or relevant to the practice of REM in the avionics industry. These include:

- IEEE Guide for Developing System Requirements Specifications (IEEE Std 1233) [25]
- IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998) [26]
- Software Considerations in Airborne Systems and Equipment Certification (DO-178B) [27]
- Final Report for Clarification of DO-178B, "Software Considerations in Airborne Systems and Equipment Certification" (DO-248B) [28]
- Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance (DO-278) [29]
- Certification Considerations for Highly-Integrated or Complex Aircraft Systems Aerospace Recommended Practice (ARP) 4754) [30]
- Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, ARP 4761 [31]

2. RECOMMENDED PRACTICES.

What makes one requirements specification good and another bad? To paraphrase David Parnas, a good requirements specification should describe everything necessary to produce the correct system—and nothing more. This succinctly states the balance that requirements need to achieve—specifying everything needed of the system to be built, while not overconstraining the developers by venturing into design. As is often stated, the requirements should specify what the system will do, not how the system will do it.

However, this is almost always a surprisingly large amount of information. Considerable effort has to be invested in organizing the requirements so they are readable, often by audiences with very different concerns and expertise, and maintainable in the face of change. As a result, a good requirements specification consists of much more than just a list of shall statements. Many of the recommended practices described in this Handbook are devoted to putting into place the structure to ensure that the requirements are complete, consistent, clear, and well organized.

At the same time, the development of the requirements is typically a progression from a state in which relatively little is known about the system to one in which a great deal is known. To be effective, the requirements engineering process needs to progress in a similar fashion from informal practices early in requirements definition to more rigorous practices as the requirements are completed.

For large systems, it is usually impractical to state the detailed requirements of the system independent of the system architecture. Instead, high-level requirements are developed, the next level of design is completed, and more detailed requirements are developed for each component. This process is continued until the necessary level of detail is reached. In effect, the process of requirements specification is interleaved with developing the system architecture.

This Handbook describes the following 11 main-level recommended practices that allow developers to progress from an initial, high-level overview of the system to be developed to a detailed description of its behavioral and performance requirements.

1. Develop the System Overview
2. Identify the System Boundary
3. Develop the Operational Concepts
4. Identify the Environmental Assumptions
5. Develop the Functional Architecture
6. Revise the Architecture to Meet Implementation Constraints
7. Identify System Modes
8. Develop the Detailed Behavior and Performance Requirements
9. Define the Software Requirements
10. Allocate System Requirements to Subsystems
11. Provide Rationale

These 11 main-level recommended practices are shown and described in detail in sections 2.1 through 2.11.

With the exception of the last two main-level recommended practices, these recommended practices are presented in roughly the order they would be performed while in a software development program, but this order does not need to be strictly adhered to. Significant iteration between these steps is to be expected on any actual development. While the main-level recommended practices are applicable to almost any development, organizations may want to tailor these main-level recommended practices to better fit with their existing practices.

Within each of the main-level recommended practices, more detailed, sublevel recommended practices are specified. These sublevel recommended practices are more sensitive to how the main-level recommended practice is implemented and are more dependent on how the main-level recommended practice is performed. These sublevel recommended practices are more likely to have to be modified as the main-level recommended practice is tailored to fit existing methods and tools. These sublevel recommended practices are shown and described in detail in sections 2.1.1 through 2.11.7 and in corresponding tables.

2.1 DEVELOP THE SYSTEM OVERVIEW.

Refer to table 1 for a description of main- and sublevel recommended practices for developing the system overview.

Table 1. Develop the System Overview

Main- and Sublevel Recommended Practices	
2.1	Develop the System Overview: Develop an overview of the system that includes a brief synopsis, describes all contexts in which the system will be used, and lists the main goals, objectives, and constraints of the system. This helps to define the system scope while the requirements are being developed and serves as a means to quickly orient a new reader of the requirements.
2.1.1	Develop the system overview early in the requirements engineering process and use it as the introduction to the requirements specification. Keep the overview at a high level so it can be used to quickly orient new readers.
2.1.2	Provide a short textual synopsis of the system as the first part of the system overview. The synopsis should name the system, describe its purpose, and summarize the system capabilities.
2.1.3	Consider the entire life cycle of the system and identify each distinct context in which it will be used.
2.1.4	Use context diagrams in the system overview to provide a high-level, graphical depiction of the system, the external entities it interacts with, and those interactions.
2.1.5	For each context diagram, provide a brief description of each external entity and its interactions with the system.
2.1.6	Capture a preliminary set of system goals early in the requirements engineering process so they can be used to guide the development of the requirements.
2.1.7	Depending on the size and volatility of a project, collect and maintain the information about each system goal necessary to continuously assess its importance relative to the other goals.

2.1.1 Develop System Overview Early.

The system overview serves as the introduction to the system requirements. Though it will evolve continuously as the requirements are defined, it is usually one of the first artifacts created. Its purpose is to quickly orient the reader to the system and the environment in which it will operate. It should not attempt to completely describe the system. Rather, its intent is to provide the reader with a high-level view of what the system does, how it interacts with its environment, and why the system is needed. At a minimum, it should include a short synopsis of the system; one or more context descriptions; a brief description of each external entity in the context diagrams; and a set of high-level system goals, objectives, and constraints. It may also include other relevant information, such as historical background, but this needs to be balanced against the need to quickly orient a new reader. Examples of system overviews are provided in appendix A.1 for the Isolette Thermostat, appendix B.1 for the FCS, appendix C.1 for the FGS, and appendix D.1 for the AP.

Recommended Practice 2.1.1: Develop the system overview early in the requirements engineering process and use it as the introduction to the requirements specification. Keep the overview at a high level so it can be used to quickly orient new readers.

2.1.2 Provide System Synopsis.

The overview itself should begin with a synopsis that provides a brief narrative description of the system. Ideally, the synopsis is short, clear, and describes what the system will do without implying a particular system design. The synopsis should name the system to be developed, describe its purpose, and summarize the main capabilities provided by the system. An early version of the synopsis for the Isolette Thermostat states:

> The system being specified is the thermostat of an Isolette. An Isolette is an incubator for an infant that provides controlled temperature, humidity, and oxygen (if necessary). Isolettes are used extensively in Neonatal Intensive Care Units for the care of premature infants.
>
> The purpose of the Isolette Thermostat is to maintain the air temperature of an Isolette within a desired range. It senses the current temperature of the Isolette and turns the heat source on and off to warm the air as needed. The system allows the nurse to set the Desired Temperature Range within a safe range for infants.

As with all the system overviews, the synopsis will evolve throughout the requirements definition. Examples of complete system synopses are provided at the start of appendix A.1 for the Isolette Thermostat, appendix B.1 for the FCS, appendix C.1 for the FGS, and appendix D.1 for the AP.

Recommended Practice 2.1.2: Provide a short textual synopsis of the system as the first part of the system overview. The synopsis should name the system, describe its purpose, and summarize the system capabilities.

2.1.3 Identify System Contexts.

The purpose of the system context is to describe, at a high level, the external entities with which the system interacts and the nature of those interactions. Note that there may be more than one context in which the system will be used during its entire life cycle. For example, there will almost certainly be an operational context, which describes the entities with which the system interacts during normal operation, but there may also be a testing context used during system development in which the system interacts with test equipment through different interfaces, or there may be a maintenance context that is used to diagnose and repair the system in the field. Each relevant context for the system should be identified and described. Since different individuals will be interested in different contexts, it is usually better to describe each context separately than to try to combine them into a single context.

Recommended Practice 2.1.3: Consider the entire life cycle of the system and identify each distinct context in which it will be used.

2.1.4 Use Context Diagrams.

For each context, the external entities with which the system interacts and the nature of those interactions should be identified. A convenient way to do this is through the use of a context diagram that graphically depicts each external entity and its interaction with the system. An example context diagram for the Isolette Thermostat is shown in figure A-1. A similar context diagram for the FCS is shown in figure B-1 for the FGS in figure C-1, and for the AP in figure D-1. Note that the system itself is shown in the context diagram as a black box with no internal structure. It may also be helpful to identify the higher-level systems in which the system is embedded (for example, the Thermostat context includes the larger system content of the Isolette).

Recommended Practice 2.1.4: Use context diagrams in the system overview to provide a high-level, graphical depiction of the system, the external entities it interacts with, and those interactions.

2.1.5 Describe External Entities.

A brief description of each external entity and its interactions with the system should also be provided. Since this is part of the system overview, these descriptions should be kept short and simple (more detailed information on the entities and their interactions will be provided in later sections of the applicable specification). Examples of the descriptions of external entities and their interactions with the system are provided in appendix A.1.1 for the Isolette Thermostat, appendix B.1.1 for the FCS, appendix C.1.1 for the FGS, and appendix D.1.1 for the AP.

Recommended Practice 2.1.5: For each context diagram, provide a brief description of each external entity and its interactions with the system.

2.1.6 Capture Preliminary System Goals.

Goals are informal statements of the system stakeholders' needs. They are not requirements since they are not verifiable and do not provide enough information to build the system [14]. However, they provide important guidance on why the system is being built and what is important to the stakeholders. During the early phases of the project, they may be all that is available. Goals frequently conflict with each other. For example, one goal may be to produce a system with a sophisticated operator interface, while another goal is to produce the system at minimal cost. One of the objectives of the requirements engineering process is to refine the goals into verifiable requirements that resolve these conflicts as effectively as possible.

A preliminary set of system goals should be captured early in the requirements engineering process so they can be used to guide the specification of requirements. A natural place to present the system goals is in the system overview. However, in a large project, the system goals may be numerous enough to justify placing them in their own section or in another document entirely.

Recommended Practice 2.1.6: Capture a preliminary set of system goals early in the requirements engineering process so they can be used to guide the development of the requirements.

2.1.7 Maintain System Goal Information.

Goals frequently help to explain why a particular requirement exists or is stated in a particular way and may be cited in the rationale (see section 2.11) for the requirement. The operational concepts (see section 2.3) also trace back to the system goals.

Management of the system goals will vary with the size, complexity, and maturity of the system being developed. On small projects, the goals may be well understood. It may be possible to list the system goals on a single page and their management may consist of nothing more than an occasional review and update. This is the case for the Isolette Thermostat example, which initially has only two simple goals:

- G1—The infant should be kept at a safe and comfortable temperature.
- G2—The cost of manufacturing the thermostat should be as low as possible.

However, on a project of any size, and particularly for new large systems, the goals will probably evolve throughout project development. New needs will be identified, priorities will shift, managers will change, understanding of the project will grow, all resulting in a changing set of project goals. Management of the goals on such a project may be a significant task in its own right. To do this, information should be collected that facilitates tracing the goal back to its source and continuously assessing the importance of that goal relative to other goals.

Potential sources of system goals are numerous and include the customers, users, regulatory bodies, and previous developments. Goals may be provided directly by these organizations or may be discovered through interviews or review of documents provided by the stakeholders. Depending on the size and volatility of a project, information that may be desirable to collect about each goal may include:

- Origin—Where did the goal come from (customer, individual, document …)?
- Origin Date—When was the goal was first identified?
- Author—Who first documented the goal?
- Priority—What is the importance of the goal relative to other goals?
- Stakeholders—Which customers or users are most concerned about this goal?
- Stability—How likely is this goal to change?
- Schedule Date—What is the date this goal is scheduled to be implemented?

For items such as priority and stability, each organization would probably want to provide their own allowed values and definitions. If the system goals are expected to change frequently, they should be kept under configuration control to provide a history of their evolution.

Recommended Practice 2.1.7: Depending on the size and volatility of a project, collect and maintain the information about each system goal necessary to continuously assess its importance relative to the other goals.

2.2 IDENTIFY THE SYSTEM BOUNDARY.

Refer to table 2 for a description of main- and sublevel recommended practices for identifying the system boundary.

Table 2. Identify the System Boundary

Main- and Sublevel Recommended Practices	
2.2 Identify the System Boundary: Develop a clear definition of the boundary between the system and its environment. This provides a solid understanding of what lies within the system to be built and what lies within a larger environment. This is done by identifying a set of variables in the environment the system will monitor and control.	
2.2.1	Define the system boundary early in the requirements engineering process by identifying a preliminary set of monitored and controlled variables.
2.2.2	Choose environmental variables that exist in the environment independently of the system to be developed.
2.2.3	Choose controlled variables that are under the direct control of the system being specified.
2.2.4	Choose monitored variables that are being directly sensed by the system being specified.
2.2.5	Ensure the monitored and controlled variables are as abstract as possible and do not include implementation details.
2.2.6	Avoid incorporating details of the operator interface in the monitored and controlled variables. Instead, define monitored or controlled variables that describe the information to be conveyed independent of its presentation format.
2.2.7	Completely define all physical interfaces to the system, including definitions for all discrete inputs, all messages, all fields in a message, and all protocols followed.

One of the most important activities in requirements engineering is to clearly define the boundary between the system and its environment. This provides a sound understanding of what lies within the system to be built and what lies within a larger environment. Every system is embedded within the environment in which it operates, and this environment is often a larger collection of systems. Without a clear definition of the system boundary, it is very easy to write requirements that duplicate or conflict with those being defined at a higher level or to miss requirements because they are assumed to be provided by the environment. This is particularly important when a system is being developed by multiple entities.

One way to define the system boundary is to view the system as a component that interacts with its environment through a set of monitored and controlled variables (figure 1) [1].

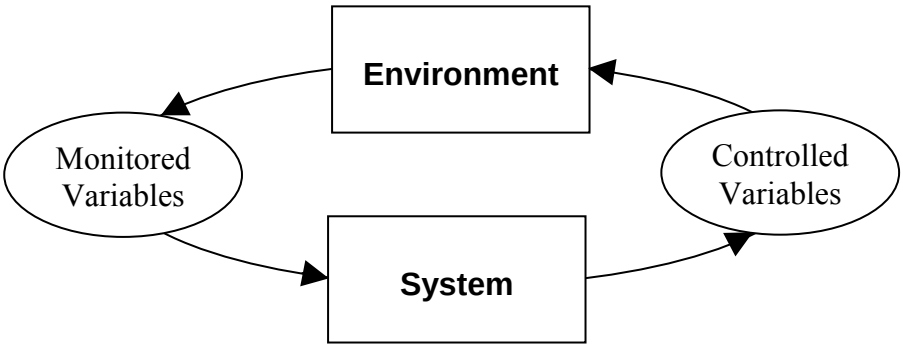


Figure 1. The System and its Environment

The monitored variables represent quantities in the environment that the system responds to, while the controlled variables represent quantities in the environment that the system will affect. For example, monitored values might be the actual altitude of an aircraft and its airspeed, while controlled variables might be the position of a control surface, such as an aileron or the displayed value of the altitude on the primary flight display. Conceptually, the monitored and controlled variables exist in the environment outside of the system and would continue to exist even if the system were eliminated.[2] The purpose of the system is to maintain a relationship between the monitored and controlled variables that achieves the system goals. The definition of the monitored and controlled variables and their attributes define the boundary of the system.

2.2.1 Identify the System Boundary Early.

Identification of the monitored and controlled variables should be started early in the requirements engineering process, even though it may not be precisely clear what they are. Having some notion of the system boundary makes many of the following activities simpler, and the process of identifying the monitored and controlled variables will often raise additional questions. As an example, a preliminary set of monitored and controlled variables for the Isolette Thermostat are shown in table 3.

[1] This is the approach taken in the SCR [2 and 3] and CoRE methodologies [4 and 5]. A similar notion is used in RSML [8], where they are referred to as the manipulated and controlled variables.

[2] While monitored and controlled variables appear similar to inputs and outputs, the terms inputs and outputs are used in this document to refer to the values provided to and generated by the software in the system.

Table 3. Thermostat Monitored and Controlled Variables

Name	Type	Physical Interpretation
Current Temperature	Monitored	Current air temperature inside Isolette
Operator Settings		Thermostat settings provided by the operator
Desired Temperature Range		Desired range of Isolette temperature
Lower Desired Temperature	Monitored	Lower value of the Desired Temperature Range
Upper Desired Temperature	Monitored	Upper value of the Desired Temperature Range
Operator Feedback		Information provided back to the operator
Display Temperature	Controlled	Displayed temperature of the air in Isolette
Heat Control	Controlled	Command to turn the heat source on or off

The Operator Settings and Operator Feedback monitored and controlled variables are grouped into aggregates and use the same names as shown in the context diagram of figure A-1[3]. While a good start, this list of monitored and controlled variables also raises more questions. What are the limits of the Desired Temperature Ranges? Is the temperature in degrees Centigrade or Fahrenheit? These questions can be resolved in subsequent steps. Early in the process, it is more important to simply identify a preliminary list of monitored and controlled variables.

Recommended Practice 2.2.1: Define the system boundary early in the requirements engineering process by identifying a preliminary set of monitored and controlled variables.

2.2.2 Choose Environmental Variables.

While it would be ideal to correctly identify the monitored and controlled variables at the very start, this is unrealistic unless the system is replacing an existing system. The system boundary will probably not be fully understood at the start of the project and is likely to shift as functionality is moved between systems during development. Nevertheless, it is better to have an incorrect documented definition of the system boundary that can be improved as time goes on, than no definition at all.

However, there are some rules of thumb that can be helpful in identifying the monitored and controlled variables. They should exist in the environment outside the system and should exist independent of the system itself. A useful heuristic is to ask if the monitored or controlled variable would continue to exist even if the system being defined were eliminated.

Recommended Practice 2.2.2: Choose environmental variables that exist in the environment independently of the system to be developed.

[3] The Operator Feedback aggregate currently only has a single element. Additional data elements will be added as the example evolves.

2.2.3 Choose Controlled Variables.

Controlled variables should be limited to those that the system can directly control. For example, in the Isolette Thermostat example, the air temperature is not selected as a controlled variable since the Thermostat can only affect it indirectly by turning on the Heat Source. Instead, the Heat Control, which is under direct control of the Thermostat, is chosen as the controlled variable.

Recommended Practice 2.2.3: Choose controlled variables that are under the direct control of the system being specified.

2.2.4 Choose Monitored Variables.

In similar fashion, the monitored variables should correspond to the physical quantity the system senses. For example, when defining the behavior of an entire avionics suite, the actual altitude of the aircraft might be an appropriate monitored variable. When defining the behavior of a subsystem, the appropriate monitored variable may be an estimation of the altitude produced by another subsystem. Carefully choosing monitored variables at the right level of abstraction for the system can avoid many sources of confusion. For example, if the purpose of the system is to determine the aircraft's altitude, it may need to obtain estimates of the altitude from several sources. In this situation, it would make no sense to choose the true altitude of the aircraft as the monitored variable. Instead, the altitude from each source would be identified as a distinct monitored variable and the system's requirements would define how to combine those estimates into an estimate of the aircraft's true altitude.

Recommended Practice 2.2.4: Choose monitored variables that are being directly sensed by the system being specified.

2.2.5 Ensure Environmental Variables are Sufficiently Abstract.

At the same time, the monitored and controlled variables should be as abstract as possible and should not contain details of how they are implemented. Thus, a monitored or controlled variable could be a real number with one digit of precision that ranges from -100.0 to +50,000.0, but they should not be an Aeronautical Radio, Incorporated (ARINC) 429 word. The monitored and controlled variables should contain no more detail than what the system needs to perform its function, assuming a perfect interface to the outside world.

Recommended Practice 2.2.5: Ensure the monitored and controlled variables are as abstract as possible and do not include implementation details.

2.2.6 Avoid Presentation Details in Environmental Variables.

Particular care should be taken to avoid incorporating presentation details into monitored and controlled variables that are part of the operator interface. For example, a display might indicate a warning to a pilot by displaying an altitude in yellow and a hazardous situation in blinking red. In this situation, there are actually two controlled variables, the altitude (which would be a

number) and the status (normal, warning, or hazard). The presentation of the controlled variables to the pilot is more appropriately dealt with as part of the detailed design of the human-machine interface (HMI).

There are a number of other attributes that should be collected for each monitored and controlled variable such as type, range, precision, and status. However, these attributes are more appropriately thought of as part of the environmental assumptions the system depends on and are discussed in section 2.4.

The definition of the monitored and controlled variables and their attributes can be presented in a single location in the requirements specification, similar to a data dictionary, or they can be grouped with the external entity with which they are associated.[4] A completed example of associating the monitored and controlled variables with their external entity is shown for the Isolette Thermostat in appendix A.3. Similar examples are provided in appendix B.3 for the FCS, in appendix C.3 for the FGS, and in appendix D.3 for the AP.

Recommended Practice 2.2.6: Avoid incorporating details of the operator interface in the monitored and controlled variables. Instead, define monitored or controlled variables that describe the information to be conveyed independent of its presentation format.

2.2.7 Define All Physical Interfaces.

Eventually, the system boundary should be extended into a complete definition of the physical interfaces to the system. This should identify all discrete inputs and outputs, all messages, all fields in a message, and the protocols used to receive and send messages. If the interface adheres to a standard interface or protocol, the standard should be cited.

Hooks and Farry [22] recommend doing this as soon as possible. Since mismatches between interfaces are a common source of error, this is reasonable. However, there is a danger in defining the physical interfaces without also identifying the monitored and controlled variables. The monitored and controlled variables are, by definition, at a level of abstraction that is stable and unlikely to change unless the problem domain changes. In contrast, the physical interfaces define the means by which the system senses and controls these quantities and are at a much lower level of abstraction. If the physical interfaces are used directly as the monitored and controlled variables, the requirements specification is immediately brought down to the same level of abstraction. This makes specification too dependent on implementation details and less robust and reusable.

For these reasons, the monitored and controlled variables and the physical interfaces should both be defined. If the physical interfaces are known at the start of the project, they can be used to help identify the monitored and controlled variables, providing some care is used to identify the true environmental quantities the system will monitor and control and to define them at the proper level of abstraction. If the physical interfaces are not known at the start of the project,

[4] Another alternative is to use tools to generate multiple views of the monitored and controlled variables.

they will be much easier to define if the monitored and controlled variables have been identified and the full behavior of the system defined.

Recommended Practice 2.2.7: Completely define all physical interfaces to the system, including definitions for all discrete inputs, all messages, all fields in a message, and all protocols followed.

2.3 DEVELOP THE OPERATIONAL CONCEPTS.

Refer to table 4 for a description of main- and sublevel recommended practices for developing operational concepts.

Table 4. Develop the Operational Concepts

Main- and Sublevel Recommended Practices
2.3 Develop the Operational Concepts: For all contexts in which the system will be used, define a black-box view of how the system will interact with its environment. This includes identifying the functions the operators or other systems expect the system to provide, the orders in which those functions can be invoked, the values that can be provided as inputs, and the information needed from the system as feedback. Use cases are one popular way to do this.
2.3.1 Document the nominal "sunny day" behavior of the system first. Later, as an extension of this nominal behavior, address failures and exceptions.
2.3.2 Include use cases that describe how the system of interest is used within the larger context of its operating environment.
2.3.3 Use the goal of each use case as its title.
2.3.4 Trace each use case back to the system goals it helps satisfy.
2.3.5 Identify the primary actor that initiates each use case, the preconditions that must hold before the use case starts, and the postconditions that must hold when the use case is finished.
2.3.6 Ensure each use case describes a dialogue between the primary actor, the system of interest, and the other actors.
2.3.7 Link each step of a use case to any system function it calls out.
2.3.8 Consolidate actions that are repeated in several use cases into a single use case that can be called from multiple locations.
2.3.9 Describe exceptional situations or ways in which a use case can fail to meet its goal or postconditions through the use of exception cases.
2.3.10 Describe alternate ways a use case can meet its goal and postconditions though the use of alternate courses.

Table 4. Develop the Operational Concepts (Continued)

Main- and Sublevel Recommended Practices
2.3.11 In a use case, use the names of external entities for the actors and the names of monitored and controlled variables in the precondition, postcondition, and steps of the use case.
2.3.12 Avoid specifying details of the operator interface in the operational concepts. Instead, call out the system capabilities to be invoked by the operator.
2.3.13 Update the system boundary with any new monitored and controlled variables identified during development of the use cases.
2.3.14 From the uses cases, assemble a preliminary set of functions to be provided by the system.

Operational concepts are scripts or scenarios describing how the system will be used [22]. They are a useful step in the progression from the system overview to the detailed requirements. Operational concepts view the system as a black box and describe how it interacts with its operators and other systems in its environment. They help to identify what functions the operators expect the system to perform, the values the operators can provide as inputs, and the information the operators need as feedback. They also help identify the sequence in which the operators can invoke System Functions during normal operation and the response of the operators when the system behaves differently than expected.

Ideally, operational concepts are written in a natural, intuitive style that all stakeholders can understand. This helps to build consensus between the different stakeholders and to identify System Functions that may have been overlooked. Since the operational concepts focus on how the operators and other systems interact with the system, they will often uncover operator interface issues, particularly in terms of what information is needed from the operator, when functions can be invoked by the operators, and what information the operators need from the system.

At the same time, the operational concepts should avoid defining specific details of the HMI since this will limit the applicability of the requirements to that interface. In many avionic systems, the HMI has become so complex that several accidents have been traced to poor design of the operator interface [24 and 32 through 37]. For this reason, design of the HMI is often treated as an integrated activity that proceeds in parallel with the development of the system itself [16 and 24].

For example, in the Isolette Thermostat, the Desired Temperature Range must be provided by the operator, but should it be entered through a keypad or by setting pointers on a dial? Should the alarm be a buzzer or a blinking light? While these decisions are important, it is not necessary that they be resolved at the operational concepts stage. Unfortunately, a full discussion of the design of the HMI is a complex topic that is beyond the scope of this Handbook, and the practices presented herein are limited to identifying and documenting only the high-level operational concepts. The reader is referred to references 24, 33, 38, and 39 for additional guidance in the design of the HMI.

Use cases are a popular way to identify and document interactions between a system, its operators, and other systems. The literature on use cases is large and covers the gamut from high-level requirements down to detailed design. Some of the best-known references include references 17, 18, and 19. While conventions for the format of use cases are similar, there are numerous styles that introduce varying degrees of rigor and formality. However, use cases seem to be especially appropriate for use early in the REM process by helping to understand and describe how operators and other systems will interact with the system. One of their attractions is that they can be used relatively informally to elicit a better understanding of the System Functions the operators need. In effect, they provide early validation of the system behavior.

2.3.1 Document Sunny Day System Behavior.

In the terminology of uses cases, the operators, the system of interest, and other systems that interact with the system are referred to as actors. Each use case describes a dialogue of requests and actions between the actors and the system to achieve some goal. The actor that initiates the use case is referred to as the primary actor. A precondition identifies conditions that must be true before the use case is started, and a postcondition identifies conditions that must be true when the use case is successfully completed. The main success scenario describes a sunny day dialogue in which nothing goes wrong. If there is more than one way to achieve the same goal, alternate courses can be defined. Ways in which a use case can fail to meet its goal or its postconditions are handled through exception cases. The Normal Operation of Isolette[5] use case shown in figure 2 illustrates many of these concepts.

This use case describes the sunny day operation of the Isolette Thermostat without considering failure conditions or exceptions. Early in the process, it is acceptable to just identify the nominal functionality required of the system. Failure conditions and exceptions are then addressed as an extension of this nominal behavior. In particular, this can be done during the system safety assessment and the functional system design.

Recommended Practice 2.3.1: Document the nominal "sunny day" behavior of the system first. Later, as an extension of this nominal behavior, address failures and exceptions.

[5] This Use Case differs from the Normal Operation of Isolette Use Case in appendix A.2.1. The example in the appendix includes changes made in later sections to handle implementation constraints and exception cases.

Use case A.2.1: Normal Operation of Isolette

This use case describes the normal operation of the Isolette by the Nurse.

Related System Goals: G1

Primary Actor: Nurse

Precondition:
Infant is ready to be placed in the Isolette.
Isolette and Thermostat are turned off.

Postcondition:
Infant is removed from the Isolette.
Isolette and Thermostat are turned off.

Main success scenario:
1. Nurse turns on the Isolette.
2. Isolette turns on the Thermostat.
3. Thermostat initializes and enters its normal mode of operation. [System Function A.5.1.2]
4. Nurse Configures the Isolette for the needs of the Infant. [UC A.2.2]
5. Nurse waits until the Current Temperature is within the Desired Temperature Range. [System Function A.5.1.1]
6. Nurse places the Infant in the Isolette.
7. Isolette maintains desired temperature. [UC A.2.3]
8. Nurse confirms that Current Temperature is in Desired Temperature Range during rounds. [System Function A.5.1.1]
9. Nurse removes the Infant.
10. Nurse turns off the Isolette.
11. Isolette turns off the Thermostat.

UC = Use case

Figure 2. Example Use Case

2.3.2 Include How the System is Used in its Operating Environment.

Note that the sunny day use case describes the operation of the Thermostat by describing how it is used within the larger context of the normal operation of the Isolette. This is another important benefit of use cases in that they describe the operation of the system in its environment in a very simple format. For example, this use case makes it clear in the main success scenario that the Thermostat is automatically turned on and off by the Isolette, i.e., the Nurse does not also have to turn the Thermostat on and off. It would be difficult to convey this in a use case that focused on just the interface to the Thermostat.

Recommended Practice 2.3.2: Include use cases that describe how the system of interest is used within the larger context of its operating environment.

2.3.3 Employ the Use Case Goal as its Title.

Also note that the title of the use case describes the goal of the use case, i.e., defining the normal operation of the Isolette. This makes it easy to find and reference the desired use case.

Recommended Practice 2.3.3: Use the goal of each use case as its title.

2.3.4 Trace Each Use Case to System Goals.

The use case is traced back to goal G1 for the Thermostat. Tracing the use cases back to goals helps to satisfy and ensure that the use case describes a needed behavior. It also facilitates maintenance if the system goals or the use case should change.

Recommended Practice 2.3.4: Trace each use case back to the system goals it helps satisfy.

2.3.5 Identify Primary Actor, Preconditions, and Postconditions.

The primary actor that initiates this use case is the Nurse. The precondition is that the Infant is ready to be placed in the Isolette and that the Isolette and the Thermostat are turned off. The precondition identifies the conditions that must be True at the start of the use case. Normally, the postcondition makes explicit what changes the use case will cause in the environment. Since this use case describes a complete cycle of normal use, the postcondition is the same as the precondition.

Recommended Practice 2.3.5: Identify in each use case the primary actor that initiates each use case, the preconditions that must hold before the use case starts, and the postconditions that must hold when the use case is finished.

2.3.6 Ensure Each Use Case Describes a Dialogue.

The main success scenario describes the interactions between the Nurse and the Thermostat when everything goes as planned. In general, use cases should describe a dialogue between the primary actor, the system of interest, and the other actors in which each line describes the action of a different participant. If one participant dominates the dialogue, the use case should probably be revised.

Recommended Practice 2.3.6: Ensure each use case describes a dialogue between the primary actor, the system of interest, and the other actors.

2.3.7 Link Use Case Steps to System Functions.

One of the benefits of use cases is that they help identify the functions the system will provide. For example, in step 5 of figure 2, the Nurse confirms that the Current Temperature is within the Desired Temperature Range before placing the Infant in the Isolette. This reveals that the operator expects the system to display the Current Temperature. This is indicated by linking that action to the System Function Manage Regulator Interface (A.5.1.1) that provides this capability. This is done so that if a System Function is to be changed, it is easy to find and review all the ways in which the function is used. It also allows a reader of the use case to look up the System Function to more closely examine the interaction between the use case and the system.

Recommended Practice 2.3.7: Link each step of a use case to any system function it calls out.

2.3.8 Consolidate Repeated Actions Into a Single Use Case.

In step 7, the use case calls out another use case, Maintain Desired Temperature (UC A.2.3), that describes how the Thermostat will maintain the Current Temperature within the Desired Temperature Range. Breaking use cases out in this way allows actions that are used in several places to be consolidated in a single use case and then reused. This also makes the use cases more compact and understandable.

Recommended Practice 2.3.8: Consolidate actions that are repeated in several use cases into a single use case that can be called from multiple locations.

2.3.9 Describe Exceptional Situations as Exception Cases.

Exception cases are used to describe the behavior of the system and the actors when an exception to the normal sunny day behavior occurs. A full discussion of exception cases is presented in sections 2.6.4 to 2.6.6, and an illustration of an exception case is provided in the completed Normal Operation of Isolette use case in appendix A.2.1. This use case was expanded to include changes made to accommodate implementation constraints and to handle exceptions. In step 5 of the Main Success Scenario, the use case references the Failure to Maintain Desired Temperature exception case in appendix A.2.6. Exception case A.2.6 describes how the Nurse responds to a situation in which the Current Temperature in the Isolette is not at the Desired Temperature Range. This is identified as an exception case because it is does not normally occur during the main success scenario and because it would prevent the postcondition of the use case from being satisfied. In all other respects, it is treated as any other use case.

Recommended Practice 2.3.9: Describe exceptional situations or ways in which a use case can fail to meet its goal or postconditions through the use of exception cases.

2.3.10 Describe Alternate Ways to Satisfy Postconditions as Alternate Courses.

If there are other sequences that can be routinely taken to meet the postconditions these can be identified as alternate courses. Although the current example is so simple that it contains no alternate courses, an example of an alternate course can be found in the Failure to Maintain

Desired Temperature exception case shown in appendix A.2.6. In step 1 of the Main Success Scenario, the Nurse attempts to correct the problem with an Isolette that cannot keep the Current Temperature within the Desired Temperature Range, e.g., by closing the Isolette door. The situation in which the Nurse is unable to correct the problem is described by Alternate Course 1 in which the Nurse obtains and sets up a new Isolette. This also illustrates how alternate courses or exception cases can be presented as additions to the main use case, rather than as completely separate use cases.

Recommended Practice 2.3.10: Describe alternate ways in which a use case can meet its goal and postconditions through the use of alternate courses.

2.3.11 Use Names of External Entities or Environmental Variables.

It should be noted that a large proportion of the actual words in the use case refer back to the external entities identified in section 2.1 or the monitored and controlled variables identified in section 2.2. This is to be expected since the use cases treat the system as a black box with no internal structure. The monitored and controlled variables provide a natural way to describe how the external entities, the actors, interact with the system. Doing this helps to maintain consistency with the rest of the requirements specification.

Recommended Practice 2.3.11: In a use case, use the names of external entities for the actors and the names of monitored and controlled variables in the precondition, postcondition, and steps of the use case.

2.3.12 Avoid Operator Interface Details.

Note that the use cases in figure 2 and appendix A.2 avoid incorporating operator interface details when the system capabilities are invoked. For example, the Configure the Isolette use case of appendix A.2.2 does not specify whether the Nurse enters the temperature ranges by setting pointers on a dial or by pressing keys on a keypad. While the physical operator interface details are extremely important, those decisions should be made as part of the HMI design. It is too early to be committing to specific physical interfaces during development of the operational concepts. Avoiding details of the physical interface also makes the use cases more general and applicable to a wider range of systems.

Recommended Practice 2.3.12: Avoid specifying details of the operator interface in the operational concepts. Instead, call out the system capabilities to be invoked by the operator.

2.3.13 Update the System Boundary.

When used in this way, use cases are an excellent way to obtain early validation of the requirements and to document how the operators and other systems (the actors) interact with the system being developed. Their use can help to identify inconsistencies and oversights and provides deeper insight into how the system will be used. They also help to identify overlooked external entities and monitored and controlled variables. For example, step 3 in the Main Success Scenario of figure 2 discusses the initialization of the Thermostat and its entry into

normal operation. This implies that the nurse is able to determine when the thermostat enters normal operation so it can be configured, which in turn implies the need for an additional controlled variable to display the Thermostat Status. The updated list of monitored and controlled variables is shown in table 5.

Table 5. Revised Thermostat Monitored and Controlled Variables

Name	Type	Physical Interpretation
Current Temperature	Monitored	Current air temperature inside Isolette
Operator Settings		Thermostat settings provided by operator
Desired Temperature Range		Desired range of Isolette temperature
Lower Desired Temperature	Monitored	Lower value of Desired Temperature Range
Upper Desired Temperature	Monitored	Upper value of Desired Temperature Range
Operator Feedback		Information provided back to the operator
Display Temperature	Controlled	Displayed temperature of air in Isolette
Thermostat Status	Controlled	Current operational status of the thermostat
Heat Control	Controlled	Command to turn heat source on or off

Additional information can be collected in the use cases, if desired. Examples include the scope of the use case, secondary actors, and the goals of each actor in the use case. The style presented here is heavily based on that found in reference 17. More information can also be found in references 18 and 19. The completed use cases for the Isolette Thermostat are shown in appendix A.2. These use cases were extended so sunny day use cases A.2.1 through A.2.3 include references to the exception use cases A.2.4 through A.2.6. They also reference capabilities not yet discussed, such as activating the alarm. Example use cases for the FCS are shown in appendix B.2.

Recommended Practice 2.3.13: Update the system boundary with any new monitored and controlled variables identified during development of the use cases.

2.3.14 Assemble a Preliminary Set of System Functions.

While creating the use cases, a preliminary list of System Functions should be assembled. These will be used as input to the activity described in section 2.5. Such a preliminary list of System Functions for the Isolette Thermostat is shown in table 6.

Table 6. Preliminary Set of Isolette Thermostat Functions

Turn the thermostat on and off	Indicate the thermostat status
Set the Desired Temperature	Turn heat source on and off
Display the current temperature	

Recommended Practice 2.3.14: From the uses cases, assemble a preliminary set of functions to be provided by the system.

2.4 IDENTIFY THE ENVIRONMENTAL ASSUMPTIONS.

Refer to table 7 for a description of main- and sublevel recommended practices for identifying environmental assumptions.

Table 7. Identify the Environmental Assumptions

Main- and Sublevel Recommended Practices
2.4 Identify the Environmental Assumptions: Every system makes specific assumptions about the environment in which it will operate. Some of these assumptions are nothing more than the types, ranges, and units of the inputs it will accept and the outputs it will produce. Often, correct behavior of the system is dependent on more complex assumptions about its environment. These are actually requirements levied by the system on its environment. Identification of a system's environmental assumptions is essential for maintenance and to enable reuse. Failure to identify the environmental assumptions and the subsequent misuse of the system is a common cause of system failure.
2.4.1 Define the type, range, precision, and units required for all monitored and controlled variables as part of the system's environmental assumptions.
2.4.2 Provide rationale that documents why the environmental assumptions are included.
2.4.3 Organize environmental assumptions together with the external entity they constrain so it is easy to identify all the obligations placed on each external entity.
2.4.4 If an environmental assumption defines a relationship among several external entities, define an external entity responsible for ensuring the assumption is met and associate the assumption with that entity.
2.4.5 Define a status attribute for each monitored variable. Each value of the status variable should correspond to a different system behavior. The initial status of the monitored variable should ensure that the monitored variable is not used until it is sensed at least once.

Environmental assumptions are the assumptions about the environment on which a system depends on for correct operation. These can be specified as a mathematical relationship between the controlled and the monitored variables.[6] This relationship can be as simple as the types, ranges, and units of the monitored and controlled variables (the system assumes the altitude will never be less than 100 feet below sea level or more than 50,000 feet above sea level), or as complex as a complete mapping from the controlled variables to the monitored variables that describes the full behavior of the environment.

[6] In the SCR [2 and 3] and CoRE [4 and 5] methodologies this is called the NAT (natural) relationship.

Identifying the assumptions a system makes about its environment is as important a part of REM as specifying the required behavior of the system. Hooks and Farry cite incorrect facts or assumptions as the most common form of requirements errors [22]. Failure to identify and document the environmental assumptions also has been the cause of several dramatic system failures. In some cases, the failure occurred because a subsystem developed by one team did not meet the assumptions made by another team. An essential first step to preventing such failures is to identify and document these assumptions.

Identifying the environmental assumptions is also essential for reuse of a component. In several cases, dramatic failures have occurred because an existing system was reused in a different environment and the developers were unaware of the assumptions made by the original developers. Documenting the environmental assumptions is a necessary perquisite for component reuse.

2.4.1 Define the Type, Range, Precision, and Units.

While it is desirable that a system depends on as few environmental assumptions as possible, it is not possible to design a system that does not make some environmental assumptions. No system can accept an infinite range of inputs, and at some point, at least the types and ranges of the inputs and outputs must be selected. These assumptions need to be documented along with the rationale for the values selected (see section 2.11). Often, there are other more complex assumptions that also need to be documented.

For example, before the Isolette Thermostat of appendix A can be built, the types, ranges, precision, and units of the monitored and controlled variables need to be defined. Some of the environmental assumptions for the Current Temperature monitored variable are shown in table 8.

Table 8. Environmental Assumptions for the Current Temperature Monitored Variable

Name	Type	Range	Units	Physical Interpretation
Current Temperature	Real	[68.0..110.0]	°F	Current air temperature inside Isolette

These state that the Current Temperature provided by the Temperature Sensor is assumed to be a real number between 68.0 and 110.0, with a precision of at least 0.1 that expresses a temperature in degrees Fahrenheit. In effect, this places requirements back on the Temperature Sensor and establishes a contract with it. For example, if the Temperature Sensor provides the temperature in degrees Celsius, the Thermostat is unlikely to perform its function correctly. Of course, this agreement can be changed, and the Current Temperature could be provided in degrees Celsius and the Thermostat could make a trivial conversion to degrees Fahrenheit. The important point is that the environmental assumptions are clearly documented so all obligations are known and then fully acted upon.

Recommended Practice 2.4.1: Define the type, range, precision, and units required for all monitored and controlled variables as part of the system's environmental assumptions.

2.4.2 Provide Rationale for the Assumptions.

However, even for something as simple as the Current Temperature, several questions immediately come up. Why is the temperature specified in degrees Fahrenheit rather than degrees Celsius? Why is the range 68.0° to 110.0°F? Why is a precision of 0.1°F specified? This information should be provided by documenting the rationale for each choice (see section 2.11). For the current temperature, these include:

- The Current Temperature will be provided to the Thermostat in degrees Fahrenheit.

  Rationale: Consistency with environmental-assumptions operator interface (EA-OI)-1 (All temperatures will be displayed in degrees Fahrenheit.).

- The Current Temperature will be sensed to an accuracy of ±0.1°F.

  Rationale: A precision and accuracy of 0.1°F is necessary to ensure the Thermostat can turn the Heat Source on and off quickly enough to maintain the Desired Temperature Range.

- The Current Temperature will cover the range of at least 68.0° to 103.0°F.

  Rationale: This is the specified range of operation of the Isolette. The lower end of this range is useful for monitoring an Isolette that is warming to the Desired Temperature Range. The upper end is set 1° greater than the Upper Desired Temperature to ensure that the Current Temperature will be sensed across the entire Desired Temperature Range.[7]

The rationale (section 2.11) provides a basis for discussing whether the environmental assumptions can be changed. It also provides valuable information for the developers and for future maintenance.

Recommended Practice 2.4.2: Provide rationale that documents why the environmental assumptions are included.

2.4.3 Organize Assumptions Constraining a Single Entity.

A useful way to organize the environmental assumptions is to present them with a more detailed description of the external entity they constrain. This makes it simple to review all the obligations being placed on each external entity. In the full Isolette Thermostat requirements specification, a section is created describing each external entity that lists the environmental assumptions made about it (see appendix A.3).

Recommended Practice 2.4.3: Organize environmental assumptions together with the external entity they constrain so it is easy to identify all the obligations placed on each external entity.

[7] In the final specification, the upper range is set to 106.0°F to exceed the Upper Alarm Temperature.

2.4.4 Organize Assumptions Constraining Several Entities.

Some environmental assumptions define more complex relationships between several environmental variables. For example, some assumptions made about the Desired Temperature Range include:

- The Lower Desired Temperature will always be ≥97°F.

  Rationale: Exposing the Infant to temperatures lower than 97°F may result in excessive heat loss and drop in heart rate secondary to metabolic acidosis.

- The Lower Desired Temperature will always be less than or equal to the Upper Desired Temperature minus 1°F.

  Rationale: If the Lower Desired Temperature is greater than or equal to the Upper Desired Temperature, it is unclear if the Heat Source should be on or off. This may result in excessive cycling of the Heat Source.

- The Upper Desired Temperature will always be ≤100°F.

  Rationale: Exposing the Infant to temperatures greater than 100°F may result in an incorrect diagnosis of fever, resulting in aggressive evaluation (blood culture and lumbar puncture) and treatment for infection.

Whether these assumptions are reasonable depends on how the operator interface is implemented. If the interface consists of a dial with pointers that can only be set in 1° increments, which ensures through its mechanical construction that the assumptions are met, they may be very reasonable. If the temperature ranges are entered through a digital keypad, the physical interface may not ensure the assumptions are satisfied. In that case, the system requirements will need to be strengthened to ensure the Thermostat works correctly.

The important point is that the environmental assumptions on which the system depends are clearly documented and can be checked. When combined with the requirements for the system, they effectively form a contract that allows components to be developed independently. The environmental assumptions define the obligations that must be met by the environment, including other systems with which the system interacts. If these are met, then the system under development is required to satisfy its requirements. In software development, environmental assumptions are often referred to as preconditions and the requirements as postconditions.

As mentioned earlier, the environmental assumptions also provide important documentation for later maintenance and reuse. If the system will be used in a different environment, the environmental assumptions can be checked to make sure they are still satisfied. For example, if the mechanical operator interface for the Isolette Thermostat will be replaced with a modern digital interface (that may not ensure all the environmental assumptions made by the thermostat), it should be straightforward for the developers to first find and review the relevant environmental assumptions.

Other common sources of environmental assumptions relate the monitored variables to the controlled variables (effectively a partial model of the plant being controlled) or limit the rate at which monitored values can change. For example, the Isolette Thermostat depends on the Current Temperature in the Isolette not changing too quickly so that the Thermostat can turn the Heat Source on and off quickly enough to keep the Current Temperature within the Desired Temperature Range. These environmental assumptions are stated as:

- When the Heat Source is turned on and the Isolette is properly shut, the Current Temperature will increase at a rate of no more than 1°F per minute.

- When the Heat Source is turned off and the Isolette is properly shut, the Current Temperature will decrease at a rate of no more than 1°F per minute.

However, these assumptions raise a problem of which external entity they should be associated with. They are not really assumptions about the Temperature Sensor, nor are they assumptions about the Heat Source. Rather, they span both of those entities. A useful heuristic is that they should be grouped with the entity that is responsible for ensuring that they are met. This would be the Isolette itself, which includes the Temperature Sensor, the Heat Source, and the Thermostat. For this reason, the requirements specification should include an external entity for the Isolette containing these environmental assumptions. An example of this can be found in appendix A.3.1.

Recommended Practice 2.4.4: If an environmental assumption defines a relationship among several external entities, define an external entity responsible for ensuring the assumption is met and associate the assumption with that entity.

2.4.5 Define a Status Attribute for Each Monitored Variable.

A more subtle form of environmental assumptions is related to monitored variables and the level of trust that can be placed in their value. Monitored variables must be sensed by the system, and the behavior of the system may be different when it is unable to sense the value of a monitored variable. For example, some monitored variables should not be trusted after power-up until they are sensed the first time, i.e., the monitored variable's value is unknown. In other cases, the monitored variable should not be trusted if their value has not been updated recently, i.e, the monitored variable's value is stale. Several serious accidents have been traced to systems that depended on the value of unknown or stale monitored variables [24].

One way to handle this is to associate a status attribute with each monitored variable. The possible values of status attribute should be in one-to-one correspondence with the different behaviors of the specified system. For example, if the system behaves one way when the monitored variable can be trusted and another way when it cannot be trusted, its status only needs to take on two values: valid or invalid. If the system behaves one way when the monitored variable can be trusted, another way when it is unknown, and a third way when the monitored variable becomes stale, then there should be three values of the status attribute: valid, unknown, and stale. The initial status of the monitored variable should be set to a value that indicates that it cannot be trusted until it is sensed at least once.

Of course, at the current stage, there may not be sufficient information to determine what all the values of the status will be in the final system. In that case, an initial range of valid and invalid can be specified and revisited as the requirements evolve.

Recommended Practice 2.4.5: Define a status attribute for each monitored variable. Each value of the status variable should correspond to a different system behavior. The initial status of the monitored variable should ensure that that the monitored variable is not used until it is sensed at least once.

2.4.6 Summary.

In summary, the environmental assumptions should identify all environmental behaviors the system depends on to operate correctly. This is a basic step in allowing components to be developed independently. Environmental assumptions should be associated with the external entity responsible for ensuring the assumptions are met. This makes it easy to identify which assumptions would be violated when the system is used in a different environment. On the other hand, if environmental assumptions can be weakened or eliminated by strengthening the behavior of the system being developed, it is better than depending on environmental assumptions. Stated another way, a robust system will have fewer dependencies on the environment than a fragile system. However, every realizable system will always have some environmental assumptions, even if they consist of nothing more than the types, ranges, and units of the monitored and controlled variables.

2.5 DEVELOP THE FUNCTIONAL ARCHITECTURE.

Refer to table 9 for a description of main- and sublevel recommended practices for developing the functional architecture.

Table 9. Develop the Functional Architecture

Main- and Sublevel Recommended Practices	
2.5	Develop the Functional Architecture: To enhance readability of the requirements and to make them robust in the face of change, the requirements are organized into functions that are logically related with minimal dependencies between the functions. To allow the requirements to scale to large systems, functions are broken down into smaller functions.
2.5.1	Organize System Functions into groups that are closely related and likely to change together.
2.5.2	Use data flow diagrams to graphically depict System Functions and their dependencies.

Table 9. Develop the Functional Architecture (Continued)

Main- and Sublevel Recommended Practices	
2.5.3	Minimize dependencies between functions by ensuring that the information shared between functions represent stable, high-level concepts from the problem domain that are unlikely to change. Push volatile dependencies as far down in the function hierarchy as possible.
2.5.4	Define the type, range, precision, and units of all internal variables introduced to specify data dependencies between functions.
2.5.5	Organize large requirements specifications into multiple levels by nesting functions and data dependencies.
2.5.6	If providing high-level requirements, define only what can be stated at that level of the hierarchy. Do not use terms defined at lower levels of the hierarchy.
2.5.7	If providing high-level requirements for functions, avoid incorporating rationale into the requirement in an attempt to specify details more appropriately specified in a lower-level function.

For example, as small as the Isolette Thermostat is, the entire requirements specification can be written in a few pages. For real systems, this is not the case. For example, when the CoRE methodology was applied to the avionics of the C-130J aircraft, there were over 1600 monitored and controlled variables [6]. To be usable, a requirements specification for a system of any size must be organized in some manner.

The best organization may depend on how the requirements will be used. One structure may enhance the readability of the requirements, while another may make it easier to define a family of products. These goals may conflict with each other—organizing to support a product family may produce a specification that is harder to understand than one tailored for a single member of the family. Automated tools can help in this respect, producing different views of the same underlying specification as needed.

2.5.1 Organize System Functions Into Related Groups.

The functional architecture is developed through a process that recursively identifies the functions to be provided by the system, grouping together the functions that are logically related and likely to change together. Ideally, this structure would be used to organize the detailed system requirements so they are readable and robust in the face of change.[8] In actuality, other factors, such as handling the safety requirements of the system or dealing with implementation constraints, usually require that this structure be modified. These issues are explained in section 2.6, which describes an iterative approach to modifying the functional architecture to handle safety requirements and implementation constraints. However, the starting point for that activity is produced in this step.

[8] This is very similar to the principles used to organize requirements in the CoRE methodology [4 and 5].

Another important function of the functional architecture is that it provides the traceability of requirements within the specification. That is, since the requirements are grouped recursively by function, the structure of the document automatically traces the lower-level requirements to the higher-level requirements.

This process starts with examining the preliminary set of System Functions developed in section 2.3 and grouping those that are closely related into the major functions of the system. The decomposition of each major function into the next level of functions proceeds naturally as the requirements are refined.

As an example, consider the preliminary list of System Functions (table 6) developed in section 2.3 for the Isolette Thermostat. The functions to turn the Thermostat on and off, to indicate the Thermostat Status, to set the Desired Temperature Range, to display the Current Temperature, and to indicate if the Thermostat has failed are closely related to the operator interface and are all likely to be affected if the physical operator interface is changed. These could be grouped into a broader function called Manage Operator Interface. The remaining preliminary function to turn the Heat Source on and off is relatively independent of the others and could be allocated to a function called Manage Heat Source.

Recommended Practice 2.5.1: Organize System Functions into groups that are closely related and likely to change together.

2.5.2 Use Data Flow Diagrams to Depict System Functions.

A simple way of depicting the System Functions and their dependencies on each other is through a dependency diagram. The topmost dependency diagram for the Isolette Thermostat is shown in figure 3.[9]

The dependency diagram of figure 3 opens up the black box of the system from the context diagram of figure A-1 to reveal the main System Functions of the Thermostat and the dependencies between them. Each System Function just described are shown on the diagram, along with one additional System Function (Manage Thermostat Mode) introduced to manage the major modes of the system.[10]

In figure 3, the monitored variables are shown as arrows that do not originate from a function (Current Temperature and Operator Settings). Controlled variables are shown as arrows that do not terminate in a function (Heat Control and Operator Feedback), and data dependencies between functions are shown as arrows that both originate and terminate at functions (Mode, Desired Range, and Thermostat On/Off).

[9] The dependency diagram of figure 3 differs from figure A-2 in appendix A, because the latter is modified to accommodate implementation constraints.

[10] Most systems operate in more than one mode, and it is helpful to define the system behavior separately for each mode. This is discussed in more detail in section 2.7. Inputs to the Manage Thermostat Mode Function will be added later when internal and input failures are considered.

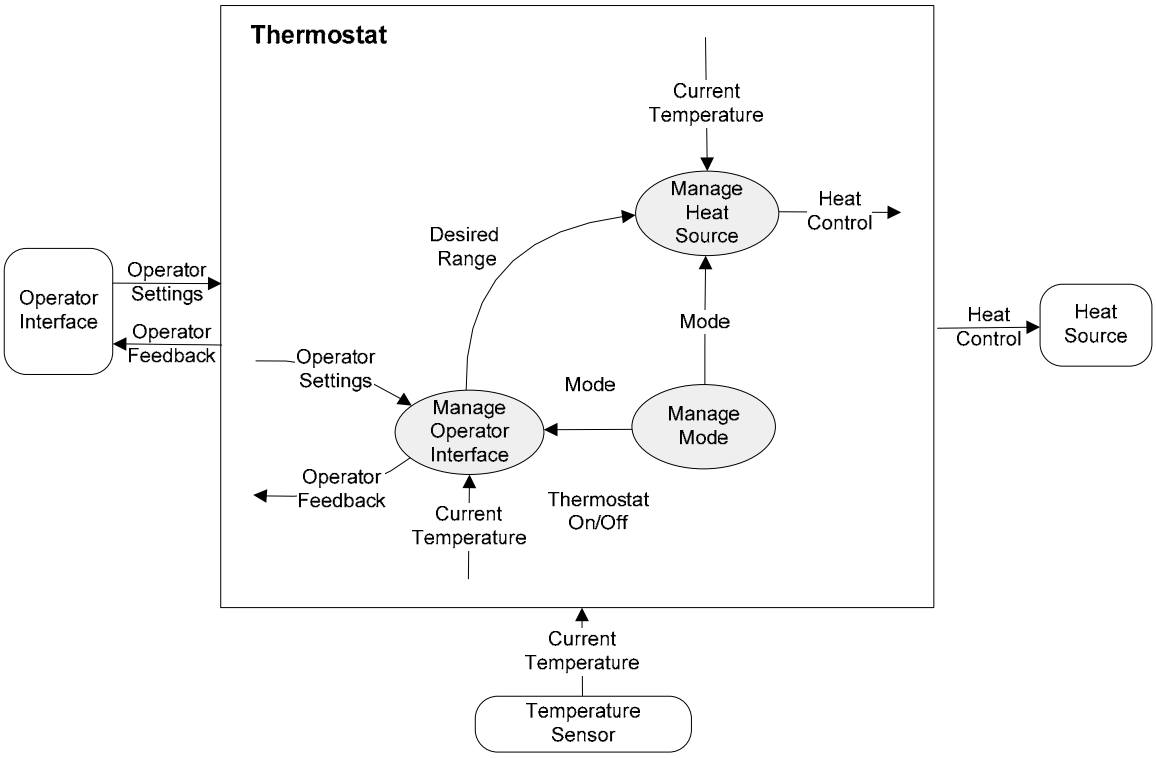


Figure 3. Thermostat Dependency Diagram

Strictly speaking, the dependencies between functions run in the opposite direction from that shown in figure 3. For example, the specification of the Manage Heat Source Function depends on the definition of the Desired Range internal variable specified in the Manage Operator Interface Function and the arrow should point from the Manage Heat Source Function to the Manage Operator Interface Function. However, drawing the dependency diagrams with the arrows pointing in the direction of dependency is both nonintuitive and conflicts with the widely accepted use of data flow diagrams in which the arrows indicate the direction of data flow. In practice, drawing the arrows as shown is much more readily accepted, and the practical impact on the specification is minor, even though technically, no data moves in a requirements specification.

Recommended Practice 2.5.2: Use data flow diagrams to graphically depict System Functions and their dependencies.

2.5.3 Minimize Dependencies Between Functions.

The arrows flowing into a function thus indicate all the definitions needed by that function, while the arrows flowing out of a function show all the values defined by that function. To make the requirements specification robust in the face of change, the label on each arrow should represent a stable, high-level concept from the problem domain that is unlikely to change. Dependencies that are likely to change are hidden within functions. This creates firewalls that help to prevent changes from rippling throughout the specification. Ideally, the more volatile a dependency, the further it should be pushed down in the function hierarchy.

Recommended Practice 2.5.3: Minimize dependencies between functions by ensuring that the information shared between functions represent stable, high-level concepts from the problem domain that are unlikely to change. Push volatile dependencies as far down in the function hierarchy as possible.

2.5.4 Define Internal Variables.

Organizing the requirements in this way can also facilitate the reuse of requirements since the areas most likely to change between products are more likely to be localized. For a more systematic, planned approach to reuse, a commonality and variability analysis can be introduced into the overall process to try to identify what part of the requirements are stable and what parts are likely to change [40]. The functionality analysis can then be conducted with that additional information as an input to the process.

Definitions of the internal variables of a dependency diagram can be provided in a single location with the diagram. An example is shown for the Isolette Thermostat in table A-1. Alternately, the internal variables can be defined within the specification of the function in which the variable originates, similar to the way monitored and controlled variables are defined in external entities. For example, the Alarm Range internal variable for the Isolette Thermostat would be defined in the Manager Operator Interface Function and the Mode internal variable would be defined in the Manage Thermostat Mode Function.

Recommended Practice 2.5.4: Define the type, range, precision, and units of all internal variables introduced to specify data dependencies between functions.

2.5.5 Nest Functions and Data Dependencies for Large Specifications.

As systems become larger, a single level of System Functions may not be adequate. To handle this, functions can be nested within functions, allowing arbitrarily large specifications to be created. This will be shown in figures 7, 8, and 9 in section 2.6. In a similar fashion, the dependencies (arrows) (see section 2.5.2) can represent aggregates of variables to allow arbitrarily large collections of monitored, controlled, and intermediate variables to be represented.

Recursively decomposing functions into subfunctions in this way provides a natural framework to organize the requirements in which logically related requirements are grouped together, and the dependencies between groups of requirements are encapsulated into the internal variables. In the original CoRE methodology [4, 5, and 6], and even structured analysis using data flow diagrams [41], the detailed system requirements were presented in the lowest level functions. In a similar fashion, section 2.8 discusses how the detailed behavior and performance requirements of the system are specified at the lowest level of the functional architecture.

Recommended Practice 2.5.5: Organize large requirements specifications into multiple levels by nesting functions and data dependencies.

2.5.6 Provide High-Level Requirements That are Really High Level.

The dependency diagrams and the definitions of the internal variables implicitly define the high-level system requirements for each function that is not at the lowest level (i.e., each nonterminal function). If desired, explicit high-level requirements can also be defined for each nonterminal function. However, care should be taken to not pull up details from the lower-level functions into the high-level functions in an effort to be thorough and precise. Doing this negates the entire purpose of developing the functional architecture, which is to provide a framework to organize a complex specification.

Figure 4 shows an example of high-level requirements for the Thermostat Function. These simply state that the Thermostat will set the value of the Heat Control and the Operator Feedback controlled variables, but does not define how they will be set. As shown in the context diagram of figure A-1, the Thermostat Function only sets two (aggregate) controlled variables, the Heat Control and the Operator Feedback. At this level of abstraction, all that is known is that the values are set by the Thermostat Function and range of those values (since their allowed values are defined as part of the environmental assumptions).

REQ-1	The Thermostat Function shall set the value of the Heat Control. Rationale. A primary function of the Thermostat is to turn the Heat Control on and off to maintain the Current Temperature in the Isolette within the Desired Temperature Range.
REQ-2	The Thermostat Function shall set the value of the Operator Feedback. Rationale. The Thermostat provides the Operator Feedback to the Operator Interface that it will use it to report the overall status of the Thermostat to the Operator.

Figure 4. High-Level Requirements for the Thermostat Function

The precise details of how those variables will be set is provided in the lower levels of the function hierarchy. Attempts to provide more information at the highest level either introduces ambiguity or duplicates information defined at the lower levels. For example, on the surface, it might seem that REQ-1 could be better stated as

- REQ-1—The Thermostat shall set the value of the Heat Control to maintain the Current Temperature in the Isolette within the Desired Temperature Range.

There are two problems with this. First, it is only True when the Thermostat is in its NORMAL mode of operation. During the INIT or FAILED mode, the Heat Control is set to the value of off. Unfortunately, the system modes are not defined until the next lower level of refinement, so specifying precisely how the Heat Control is to be set requires using terms and definitions that are not available at this level.

Recommended Practice 2.5.6: If providing high-level requirements, define only what can be stated at that level of the hierarchy. Do not use terms defined at lower levels of the hierarchy.

2.5.7 Do Not Incorporate Rationale Into the Requirements.

Second, it confuses what the system will do with why it will do it; i.e., it is mixing rationale in with the requirement itself (see section 2.11). The complete specification of how Heat Control is to be set is a surprisingly complex algorithm of its Current Mode, the Current Temperature, the Desired Temperature Range, and its previous state (see appendix A.5.1.3 for a complete specification of this algorithm). Maintaining the Current Temperature in the Isolette within the Desired Temperature Range is really an explanation of why this requirement exists (its rationale). Note that in figure 4, this information is provided to help the reader understand why the requirement is included, but it is provided as rationale, not as a requirement.

All examples in appendices A through D include textual high-level requirements for the functions that are not at the lowest level of decomposition. While the rationale provided with these is helpful in understanding the examples, the high-level requirements themselves essentially restate the information in the dependency diagrams and are not strictly necessary.

Recommended Practice 2.5.7: If providing high-level requirements for functions, avoid incorporating rationale into the requirement in an attempt to specify details more appropriately specified in a lower-level function.

2.6 REVISE THE ARCHITECTURE TO MEET IMPLEMENTATION CONSTRAINTS.

Refer to table 10 for a description of main- and sublevel recommended practices for reviewing the architecture to meet implementation constraints.

Table 10. Revise the Architecture to Meet Implementation Constraints

Main- and Sublevel Recommended Practices
2.6 Revise the Architecture to Meet Implementation Constraints: The organization produced through functional analysis is a logical architecture that may not take into account additional constraints, such as the need to satisfy system safety requirements, integrate with legacy systems, or to meet implementation constraints imposed by a particular platform. This practice describes an iterative process that starts from the previously developed functional architecture and leads to an architecture that addresses these concerns. This architecture is then used as the framework for organizing the detailed requirements.
2.6.1 If implementation constraints cannot be satisfied with the ideal functional architecture that is developed during functional analysis, modify the functional architecture as necessary, and use the final system architecture as the framework for organizing the detailed requirements.

Table 10. Revise the Architecture to Meet Implementation Constraints (Continued)

Main- and Sublevel Recommended Practices	
2.6.2	When modifying the functional architecture to accommodate implementation constraints, keep the final system architecture as close to the ideal functional architecture as possible.
2.6.3	Revise the system overview to reflect any changes in how the system interacts with its environment, any new functionality added to the system to satisfy the implementation constraints, or any changes in system goals.
2.6.4	Revise the operational concepts to reflect any changes in how operators or other systems interact with the revised system architecture.
2.6.5	Review the use cases to identify steps where exceptions to the nominal behavior could occur. Develop exception cases to identify how each exception will be handled.
2.6.6	If an exception can only occur at a few points, link those steps to the exception case. If the exception can occur at almost any point, use the exception case precondition to identify when the exception case occurs.
2.6.7	Revise the system boundary to reflect any changes in the monitored and controlled variables.
2.6.8	Identify and document any new or changed environmental assumptions for the revised functional architecture.
2.6.9	Revise the dependency diagrams to show the revised functional architecture.
2.6.10	Revise any high-level requirements affected by the changes in the revised functional architecture.

Ideally, the functional architecture developed in section 2.5 could be used as the structure of the requirements specification with the detailed behavior and performance requirements defined for each function. Unfortunately, components fail with consequences for safety, new systems must integrate with existing legacy systems, and implementation constraints affect the system architecture. Many of these concerns are addressed in the system architecture design with the consequence that the final system architecture may not map directly to the logical architecture developed during functional analysis. Rather than trying to continuously map between the ideal functional architecture and the final architecture of the system, it is more practical to revise the functional architecture to take the most important of these constraints into account.

2.6.1 Modify the Architecture to Meet Implementation Constraints.

The purpose of this recommended practice is to introduce an iterative process that starts from the ideal functional architecture developed in section 2.5 and leads to a functional architecture that describes the final architecture of the system. This final architecture can then be used as the framework for organizing the detailed requirements.

Much of the motivation for developing this architecture is to provide a natural framework for specifying the detailed functional and performance requirements of the system. In effect, the diagrams in this section can be considered a graphical table of contents for the detailed requirements specification. Specification of the detailed requirements is described in section 2.8. However, prior to specifying the detailed requirements, the system modes need to be discussed as shown in section 2.7.

Recommended Practice 2.6.1: If implementation constraints cannot be satisfied with the ideal functional architecture that is developed during functional analysis, modify the functional architecture as necessary, and use the final system architecture as the framework for organizing the detailed requirements.

2.6.2 Keep Final System Architecture Close to Ideal Functional Architecture.

On the other hand, the original functional architecture was developed through analyzing the problem domain without consideration of implementation constraints. Generally, the problem domain is less likely to change than the constraints imposed by the implementation. For this reason, the closer the final architecture can be kept to the original functional architecture, the more stable it is likely to be. For this reason, it is also desirable to minimize, as much as possible, the differences between the ideal functional architecture and the final architecture.

Recommended Practice 2.6.2: When modifying the functional architecture to accommodate implementation constraints, keep the final system architecture as close to the ideal functional architecture as possible.

2.6.3 Revise the System Overview.

For safety-critical systems, the process of modifying the functional architecture to accommodate implementation constraints is often driven by the need to achieve the very high levels of reliability dictated by the system safety process [24]. For commercial avionics systems, this process is described in ARP 4761 [31]. One of the first steps of this process is the system Functional Hazard Assessment (FHA) that identifies the high-level system hazards. The FHA is then used during the Preliminary System Safety Assessment (PSSA) to determine if the system could contribute to the realization of any of these hazards. If so, derived system design safety requirements are levied by the PSSA.

For example, the FHA for the Isolette system was completed prior to the requirements definition of the Isolette Thermostat and identified the following relevant hazard:

H1. Prolonged exposure of Infant to unsafe heat or cold

Classification: catastrophic
Probability: $<10^{-9}$ per hour of operation

The Isolette system PSSA (not the Thermostat itself) identifies several ways this hazard could be realized.

- The Thermostat could fail and turn the Heat Source on or off for too long.
- The Temperature Sensor could provide an incorrect temperature to the Thermostat.
- The Operator Interface could provide the wrong Desired Temperature Range to the Thermostat.
- The Heat Source could fail, either by remaining on or off for too long or by failing to provide sufficient heat to maintain the Desired Temperature Range.

The fault tree derived during the PSSA of the Isolette system is shown in figure 5. Since each System Function could cause hazard H1, each function is assigned a probability of failure of less than 2×10^{-10} per hour of operation.

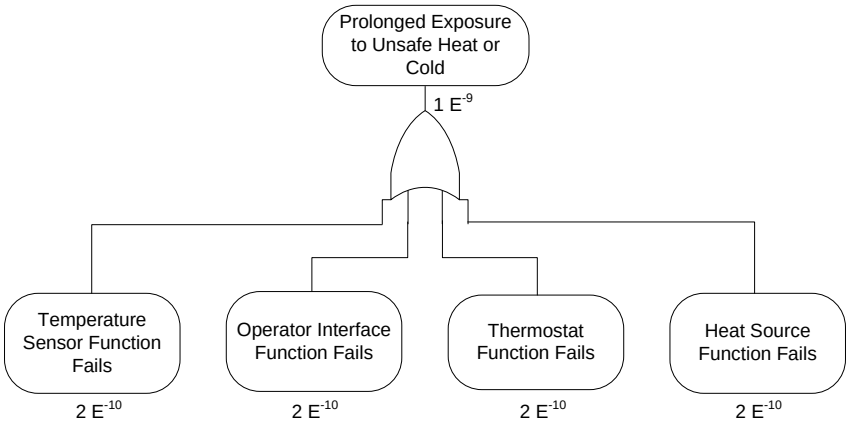


Figure 5. Initial Isolette Fault Tree

Developing individual components that achieve this level of reliability would be very expensive. Even designing a Thermostat that provides this level of reliability would contradict goal G2 to produce the Thermostat at minimal manufacturing cost. A less costly solution is to add a monitor that activates an alarm if the Current Temperature in the Isolette falls below or rises above a safe level. Another PSSA shows that the combination of such an alarm and the normal monitoring of the Infant by the Nurse would protect against a failed Thermostat Function and a failed Manage Heat Source Function (but not against a misleading Temperature Sensor Function or a misleading Operator Interface Function) and only require that the Thermostat, Heat Source, and Monitor have a probability of failure of less than 10^{-5} per hour of operation.[11] The revised fault tree derived during this PSSA is shown in figure 6.

[11] Note that the reliability requirements for the temperature sensor and the operator interface are actually for the device and its communication path to the thermostat, i.e., they are for the delivery of the devices outputs to the thermostat.

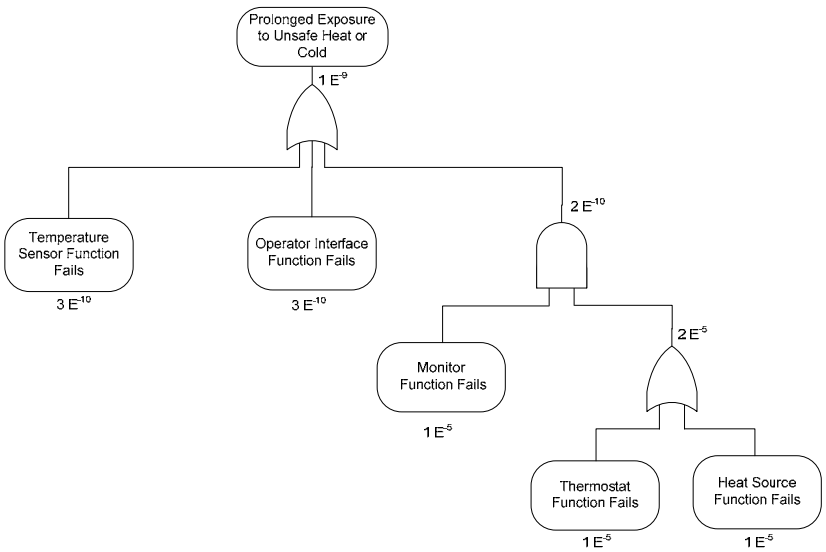


Figure 6. Revised Isolette Fault Tree

This solution still requires a highly reliable Temperature Sensor Function and Operator Interface Function. However, to keep this example simple, it is assumed that this can be achieved, perhaps through the use of fail-passive design using two or more temperature sensors and a simple, but highly reliable, mechanical operator interface. The remainder of the example will focus on the Thermostat and Monitor functions. The PSSA leads to the following derived safety requirements on these components:

- The Isolette shall include an independent Thermostat Function that maintains the Current Temperature within the Desired Temperature Range inside the Isolette.

  Rationale: The Desired Temperature Range will be set to the ideal range by the Nurse based on the Infant's weight and health. The Thermostat should maintain the Current Temperature within this range under normal operation.

  Allowed probability of failure: $<10^{-5}$ per hour.

- The Isolette shall include an independent Monitor Function that activates an Alarm within 5 seconds whenever

  - the Current Temperature inside the Isolette falls below or rises above the Alarm Temperature Range.

  - the Current Temperature is flagged as invalid.

  Rationale: The Alarm Temperature Range will be set by the Nurse based on the Infant's weight and health. The Infant should be removed from the Isolette within 15 seconds after the Current Temperature falls below or rises above the Alarm Temperature Range. With the normal monitoring provided by the Nurse, this can be accomplished within 10

seconds, leaving 5 seconds for the system to activate the Alarm. Activating the Alarm in less time is desirable.

Allowed probability of failure: $<10^{-5}$ per hour.

To meet these requirements while minimizing manufacturing costs, the Isolette designers proposed a design in which the monitor function is implemented within the thermostat itself. After extending the PSSA to this design, this was acceptable, providing the independence of the monitor is maintained.[12] To avoid confusion, the Thermostat Function was renamed as the Regulator Function, where the Thermostat is now considered the combination of the Regulator and Monitor Functions. This led to a top-level dependency diagram for the Thermostat, as shown in figure 7.

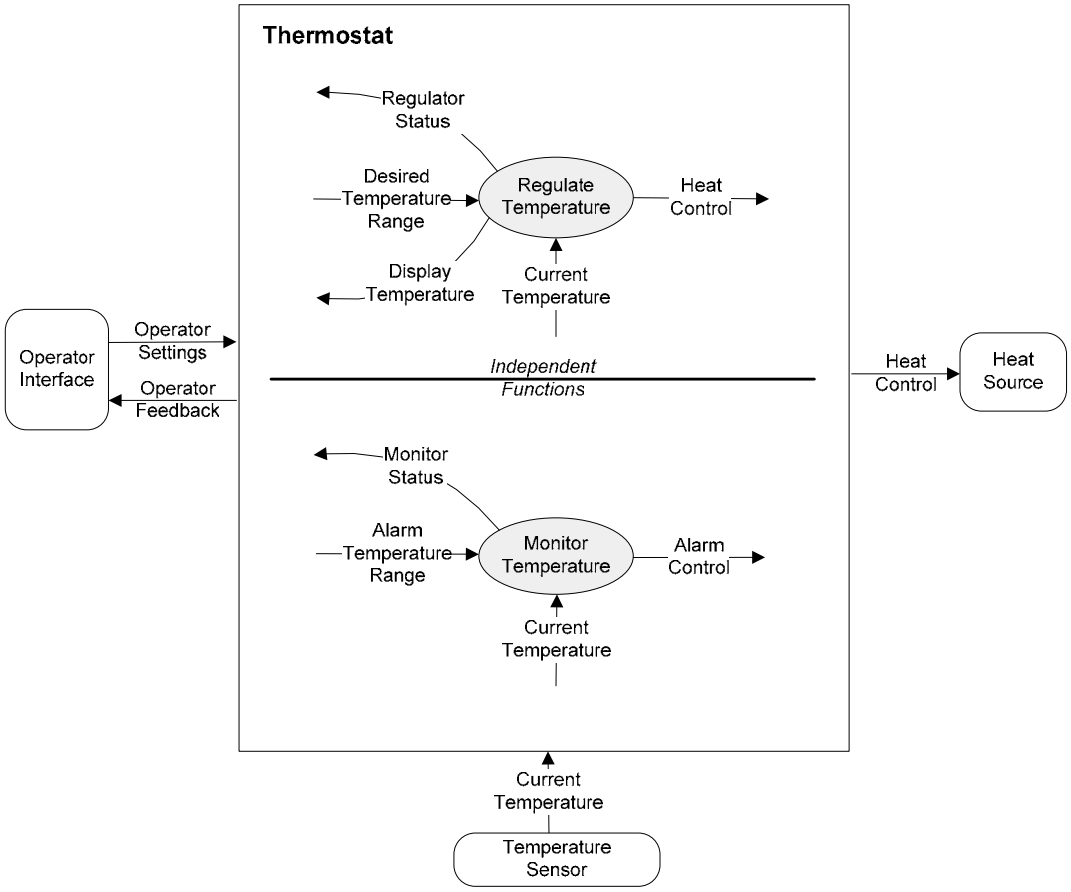


Figure 7. Revised Thermostat Dependency Diagram

In figure 7, the functionality of the Thermostat was allocated to two functions, Regulate Temperature and Monitor Temperature, whose implementations must be independent (i.e., each function must fail independently of the other). Regulate Temperature controls the Heat Source and ensures that the Current Temperature remains within the Desired Temperature Range.

[12] This may require separate power supplies or an additional monitor on the power supply of the Isolette.

Monitor Temperature activates the Alarm in the Operator Interface whenever the Current Temperature falls below or rises above the Alarm Temperature Range.

In reviewing the monitored and controlled variables for these functions, it is apparent that the system boundary has been modified. There is a new monitored variable Alarm Temperature Range provided by the Operator Interface. Also, the Thermostat Status has been renamed the Regulator Status and a new controlled variable Monitor Status has been defined. This is necessary since the overall status of the Thermostat is now determined by two independent functions. Also, note that the Display Temperature controlled variable is provided to the Operator Interface only by the Regulate Temperature Function since its value cannot contribute to any hazardous conditions, and it can be provided by a single source.

These changes require that the system overview, the system boundary, the operational concepts, and the environmental assumptions all be revised. In the system overview, the additional functionality of setting the Alarm Temperature Range and activating an Alarm needs to be added to the synopsis. Also, a new goal to warn the Nurse if the Infant becomes too cold or too hot should be added to the list of system goals. These additions are discussed in appendix A.1.

Recommended Practice 2.6.3: Revise the system overview to reflect any changes in how the system interacts with its environment, any new functionality added to the system to satisfy the implementation constraints, or any changes in system goals.

2.6.4 Revise the Operational Concepts.

If the interaction with other systems or system operators was changed to meet implementation constraints, the operational concepts should also be updated.

Recommended Practice 2.6.4: Revise the operational concepts to reflect any changes in how operators or other systems interact with the revised system architecture.

2.6.5 Develop Exception Cases.

Since the PSSA initiated consideration of how failures should be handled, this is also an appropriate time to go back and extend the use cases with exception cases. As the use cases are reviewed and new functionality is added, steps at which exceptions to the nominal (sunny day) behavior might occur should be identified. Exception cases should be defined, describing how each exception will be handled.

Recommended Practice 2.6.5: Review the use cases to identify steps where exceptions to the nominal behavior could occur. Develop exception cases to identify how each exception will be handled.

2.6.6 Link Exception Cases to Use Cases.

If the exception can only occur at a few points, a link should be made from those steps in the use cases to the exception case. If the exception can occur at almost any time (such as a system failure), it is not reasonable to create a link from every use case step, but the precondition for the exception case should make it clear when the exception can occur.

As the exception cases are identified, consideration should be given to whether or not they could contribute to a system hazard identified by the FHA. Examples of exception cases and their links are provided in appendix A.2 for the Isolette Thermostat and in appendix B.2 for the FCS.

Recommended Practice 2.6.6: If an exception can only occur at a few points, link those steps to the exception case. If the exception can occur at almost any point, use the exception case precondition to identify when the exception case occurs.

2.6.7 Revise the System Boundary.

If the revised functional architecture introduced new monitored and controlled variables, the definition of the system boundary needs to be updated. For the Isolette Thermostat, the Alarm Temperature Range monitored variable and the Alarm Control controlled variable were added, and the Thermostat Status controlled variable was replaced by the Regulator Status and the Monitor Status controlled variables.

Recommended Practice 2.6.7: Revise the system boundary to reflect any changes in the monitored and controlled variables.

2.6.8 Document Changes to Environmental Assumptions.

New environmental assumptions need to be identified and documented. For the Isolette Thermostat, the endpoints for the Alarm Temperature Range need to be documented, along with rationale for their values. This is particularly important since it defines the bounds under which the Monitor operates and is closely tied to system hazard H1. As more monitored and controlled variables are added, care should be taken to ensure that all environmental assumptions are identified and documented. These can be quite subtle. For example, there are now several more relationships that the operator interface must maintain between the Alarm Temperature Range and the Desired Temperature Range. These are documented in appendix A.3.

Recommended Practice 2.6.8: Identify and document any new or changed environmental assumptions for the revised functional architecture.

2.6.9 Revise Dependency Diagrams.

Finally, the dependency diagram in figure 3 should be replaced with the revised Thermostat dependency diagram in figure 7 and the dependency diagrams created for the Regulate Temperature and Monitor Temperature Functions. The dependency diagram for the Regulate Temperature Function is shown in figure 8.

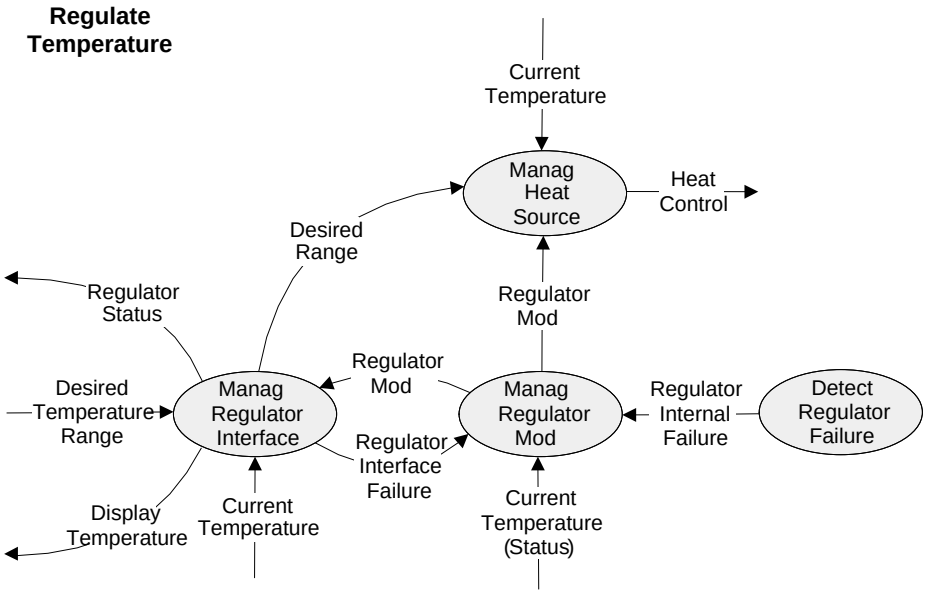


Figure 8. Regulate Temperature Dependency Diagram

In many ways, this dependency diagram looks like the original Thermostat dependency diagram shown in figure 3. This is not surprising since it covers the same basic functionality. One change that was made was to break the Operator Settings and Operator Feedback data aggregates down into their component values of Desired Temperature Range, Regulator Status, and Display Temperature. This makes the diagram a bit easier to read and makes it easier to ensure that the inputs and outputs of the diagram match those of its parent diagram.

Also, a new function, Detect Regulator Failure, has been added. This function is responsible for detecting internal failures through self-tests or other checks. Internal variables of Regulator Internal Failure and Regulator Interface Failure, along with the Status attribute of the Current Temperature monitored variable, were also added. These dependencies were added to specify how to handle failures in sensing monitored variables or internal failures of the Regulate Temperature Function. Their use will be explained further in section 2.7.

The dependency diagram for the Monitor Temperature Function is shown in figure 9. It is very similar to the Regulate Temperature Function shown in figure 8.

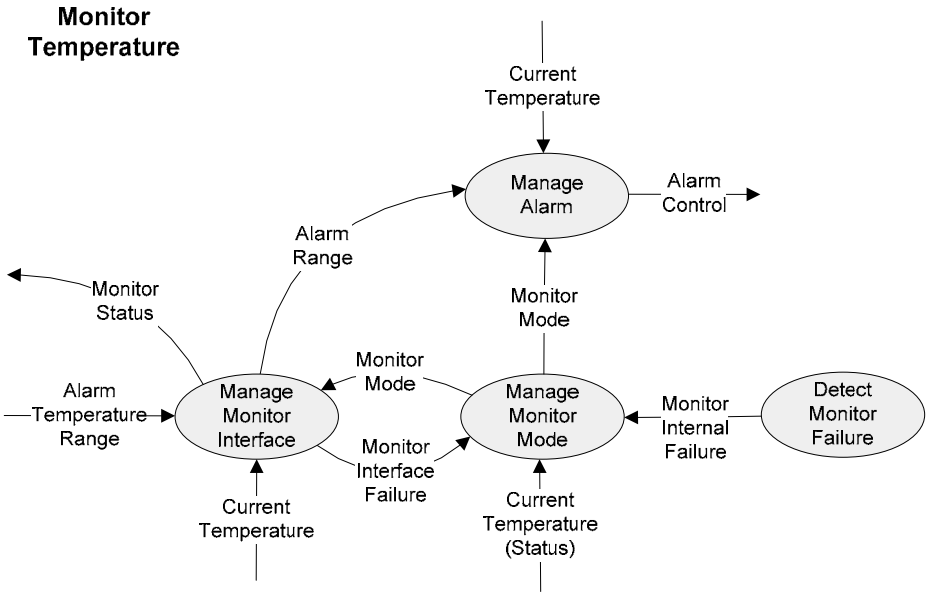


Figure 9. Monitor Temperature Dependency Diagram

The diagrams of figures 7, 8, and 9 depict how the functions of the Isolette Thermostat are related to each other, both through the function hierarchy and their data dependencies. This structure was derived from the original logical architecture developed during functional analysis (figure 3) in the course of handling the derived safety requirements. Other constraints, such as the need to integrate with a legacy system or to meet specific implementation constraints, could have a similar effect on the system architecture of interest.

Recommended Practice 2.6.9: Revise the dependency diagrams to show the revised functional architecture.

2.6.10 Revise High-Level Requirements.

Finally, any high-level requirements should be updated if the change in the system functional architecture requires them to be updated.

Recommended Practice 2.6.10: Revise any high-level requirements affected by the changes in the revised functional architecture.

2.7 IDENTIFY THE SYSTEM MODES.

Refer to table 11 for a description of main- and sublevel recommended practices for identifying system modes.

Table 11. Identify the System Modes

Main- and Sublevel Recommended Practices	
2.7	Identify the System Modes: Modes define disjointed behaviors of the system that are visible to its operators or to other systems. The detailed behavioral and performance system requirements are frequently different for the various system modes. Identification of the system modes is a useful step that simplifies detailed behavioral and performance requirements specification.
2.7.1	Identify the major system modes before defining the detailed system requirements.
2.7.2	Define how the system is allowed to transition between modes.
2.7.3	Introduce modes only to identify the externally visible discontinuities in system behavior. Do not define modes that cannot be inferred from the externally visible behavior of the system.

Modes are defined by Leveson as distinct behaviors of the system [35]. For example, a system may respond to a stimuli, such as pressing a button one way during system power-up, another way during a self-test, and yet another way during normal operation. These behaviors are three modes of the system. The system modes can be very simple or very complex. They may be relevant to the entire system or only to a part of the system. However, modes are always introduced to handle discontinuities in the system behavior.

Identification of the major system modes is helpful when writing the detailed system requirements, as discussed in section 2.8. However, at this point, it may not be clear if all the system modes were identified or if all the modes listed are actually needed. The usefulness of any proposed modes will become clear during detailed system requirements specification, and the need for any additional modes will become more apparent. As with many of the earlier steps, the system modes defined at this point serve primarily as a starting point for the work to come.

Since the system modes are so closely related to the externally visible system behavior, poor system mode design can lead to mode confusion that may cause the operator to become confused about what mode the system is in. This is an important safety concern that has been implicated in several aviation accidents [32 and 34].

Some of the potential sources of mode confusion include lack of appropriate feedback to the operators, errors in interpreting or entering information in different modes, inconsistent system behavior in different modes, different operator authority limits in different modes, silent (unannounced) mode transitions, and unintended side effects of mode transitions. Developing systems in which the modes are clearly indicated to the operators, and the system behavior is consistent, easily anticipated, and understood by the operators, is a challenging task that is beyond the scope of this Handbook. More information is provided in references 33, 35, 36, and 37.

2.7.1 Identify Major System Modes.

A system mode may or may not be explicitly displayed to a system user, but is, by definition, visible, since the system will respond differently to stimuli while in different modes. In this sense, the modes are an externally visible part of the system behavior that needs to be specified in the system requirements. Identifying the system modes also simplifies the system requirements specification by allowing the relationship between the monitored and controlled variables to be broken down into smaller pieces for each system mode. For these reasons, it is helpful to identify the major system operating modes before starting to write the detailed functional requirements.

Recommended Practice 2.7.1: Identify the major system modes before defining the detailed system requirements.

2.7.2 Define How System Transitions Between Modes.

As an example, the Isolette Thermostat Regulate Temperature Function system modes are shown in figure 10.

The system starts in the INIT mode when it is powered on and remains in this mode until the Regulator Status (shown in table 12) becomes valid. This occurs when the Regulator has completed its initialization sequence and passed all self tests (i.e., the Regulator Internal Failure, as shown in figure 8, is False) and successfully sensed all its monitored variables (the Regulator Interface Failure, as shown in figure 8, is False, and the status attribute of Current Temperature is valid).

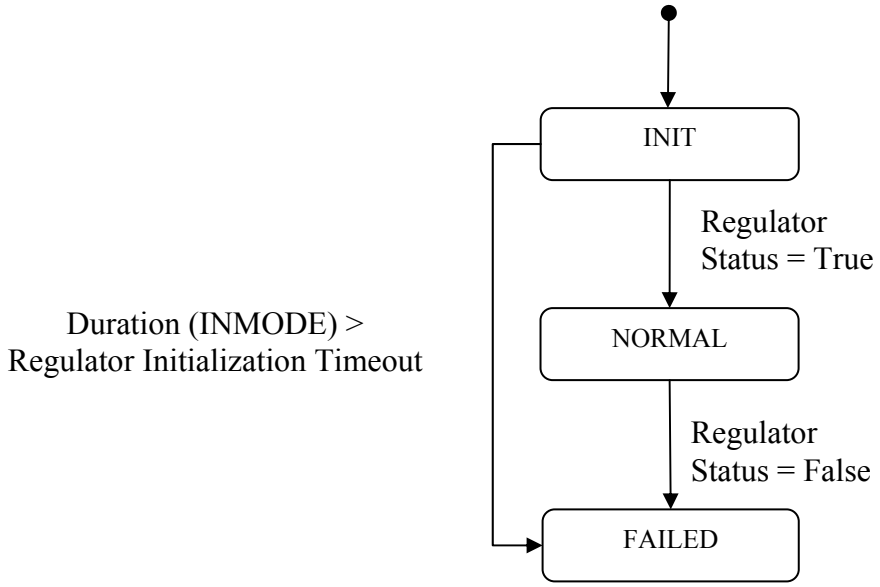


Figure 10. Regulate Temperature Function Modes

Table 12. Definition of Regulator Status

Name	Type	Definition
Regulator Status	Boolean	NOT (Regulator Interface Failure OR Regulator Internal Failure) AND Current Temperature status = Valid

At this point, it enters the NORMAL operating mode, where it typically remains until the system is powered off. If the system does not complete its initialization within a specified time-out period, or if the Regulator Status becomes False while in NORMAL mode, the function enters the FAILED mode where it remains until it is powered off and on. A similar set of modes are defined for the Monitor Temperature Function.

Recommended Practice 2.7.2: Define how the system is allowed to transition between modes.

2.7.3 Introduce Modes for Externally Visible Discontinuities.

While state transition diagrams, such as that shown in figure 10, are a popular way to specify the system modes [42], overly complex diagrams may be an indication that design decisions are being included that should be left out of the requirements specification. Modes should be defined only to identify the externally visible discontinuities in the system behavior. This allows requirements to be written in the following form:

- "If the system is in the initialization mode, the controlled variable shall be set to …"
- "If the system is in failed mode, the controlled variable shall be set to …"

This also makes it easier to determine if the requirements are complete and consistent (see section 2.8 for more examples).

Recommended Practice 2.7.3: Introduce modes only to identify externally visible discontinuities in the system behavior. Do not define modes that cannot be inferred from the externally visible behavior of the system.

2.8 DEVELOP THE DETAILED BEHAVIOR AND PERFORMANCE REQUIREMENTS.

Refer to table 13 for a description of main- and sublevel recommended practices for developing detailed behavior and performance requirements.

The preceding sections have discussed how to create the system overview, define the system boundary, develop the operational concepts, define the environmental assumptions, begin the functional analysis, modify the functional analysis to accommodate implementation constraints, and identify the main system modes. With these activities partially or fully completed, the detailed behavior and performance requirements can be specified.

Table 13. Develop Detailed Behavior and Performance Requirements

Main- and Sublevel Recommended Practice
2.8 Develop the Detailed Behavior and Performance Requirements: The behavioral and performance requirements define how the system must change the controlled variables in response to changes in the monitored variables. This includes specifying the assigned value of the controlled variable for each system state and inputs, the allowed tolerance about this value, and performance characteristics such as the allowed latency. This practice provides guidelines on how to produce a complete and consistent set of detailed system behavioral and performance requirements.
2.8.1 Use the names of the monitored and controlled variables, the system modes, and the internal variables when writing the detailed system requirements
2.8.2 For each requirement, specify the system modes and the conditions under which the requirement will apply, followed by the change in the affected variable.
2.8.3 Ensure the detailed requirements are complete, i.e., an ideal value is assigned to each controlled variable and internal variable for every system state. Use a value of UNSPECIFIED when no meaningful assignment exists for a system state.
2.8.4 Ensure the detailed requirements are consistent, i.e., that only one ideal value is assigned to each controlled variable and each internal variable for every possible system state.
2.8.5 Ensure no two detailed requirements duplicate each other, i.e., specify the same outcome for overlapping modes and conditions.
2.8.6 Present the detailed system requirements in the function that produces the variable being specified. This creates an organization in which the definition of each variable is directly traceable to its parent function
2.8.7 Define the acceptable latency for each controlled variable along with the rationale for its value as part of the detailed system requirements.
2.8.8 Define the acceptable tolerance for each numerical controlled variable.
2.8.9 Do not define latency and tolerance for internal variables.

The detailed requirements define what behavior the system will impose on its environment. These can be specified as another mathematical relationship between the monitored and controlled variables.[13] This relationship defines how the controlled variables will change in response to changes in the monitored variables. Due to the size and complexity of most systems, this is done by first defining, at the lowest level of functional architecture, what the value of each controlled and internal variable would be for a perfect system. In CoRE methodology, this is referred to as the ideal value function for a variable [5]. This defines a complete chain of assigments that define how the controlled variables will change (in the ideal case) in response to changes in the monitored variables. However, this is usually too restrictive. For most systems, a range of values near the ideal value are acceptable. Next, to capture this, a tolerance about the ideal value is specified for each controlled variable. This tolerance can be a simple constant or an arbitrarily complex function of the system state. Finally, the system performance aspects need to be specified. This is done by specifying a latency, i.e., a period of time, by which each

[13] In the SCR [2 and 3] and CoRE [4 and 5] methodologies, this is called the requirement (REQ) relationship.

controlled variable must complete its response. Again, the latency can be a simple constant or an aribtrarily complex function of the system state.[14]

2.8.1 Specify the Behavior of Each Controlled Variable.

The following example illustrates how to do this using traditional "shall" statements for the requirements. It is usually simplest to start by defining the behavior of each controlled variable. For example, part of the ideal value function for the Heat Control controlled variable is given by the following requirements:

1. If the Regulator Mode is NORMAL and the Current Temperature is less than the Lower Desired Temperature, the Heat Control shall be set to On.

2. If the Regulator mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

Several things are worth noting about these two requirements. The names of the monitored and controlled variables, the modes, internal variables, and their values are used as the requirements' vocabulary. This helps to tie the requirements to the framework that was created for their definition in the previous steps.

Recommended Practice 2.8.1: Use the names of the monitored and controlled variables, the system modes, and the internal variables when writing the detailed system requirements.

2.8.2 Specify the Requirement as a Condition and an Assigned Value.

With regard to the two requirements provided in section 2.8.1, also note how each requirement is broken into a condition under which the requirement holds an assignment of a value to the controlled variable. The condition under which the requirement holds is further broken down into the system modes (e.g., the regulator mode is NORMAL) and other conditions based on the monitored variables and the internal variables (e.g., the Current Temperature is less than the Lower Desired Temperature). This pattern is very common when specifying the requirements as shall statements.

Recommended Practice 2.8.2: For each requirement, specify the system modes and the conditions under which the requirement will apply, followed by the change in the affected variable.

2.8.3 Ensure That Detailed Requirements are Complete.

The two requirements provided in section 2.8.1 are incomplete. These requirements say nothing about the Heat Control value when the Regulator Mode is INIT or FAILED or when the Current Temperature is between the Lower Desired Temperature and the Upper Desired Temperature.

[14] Specification of the requirements as an ideal value function, tolerance, and latency are described in the CoRE methodology [4 and 5].

Ideally, the value of every controlled variable and every internal variable should be specified for all possible modes and conditions within that mode. The value of the Heat Control can be completely specified with three additional requirements:

3. If the Regulator Mode is INIT, the Heat Control shall be set to Off.

   Rationale: A Regulator that is initializing cannot regulate the Current Temperature and the Heat Control should be turned off.

4. If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

   Rationale: When the Isolette is warming towards the Upper Desired Temperature, the Heat Source should be left on until the Upper Desired Temperature is reached. In similar fashion, if the Isolette is cooling towards the Lower Desired Temperature, the Heat Source should be left off until the Lower Desired Temperature is reached.

5. If the Regulator Mode is FAILED, the Heat Control shall be set to Off.

   Rationale: In Failed Mode, the Regulator cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

This set of five requirements ensures that the ideal value function is defined for the Heat Control variable for all possible system states. This process should be repeated for every controlled and internal variable. This will result in a chain of assignments that defines how the controlled variables will ideally change in response to changes in the monitored variables.

Sometimes it is not important to make an assignment to a variable in a particular mode or condition. For example, when the Regulator Mode is INIT or FAILED, the display temperature given to the operator interface may be untrustworthy. However, it is important to document this since the users of this information (in this case the operator interface) need to know that the value of the display temperature is not significant under those conditions.

This can be done by adding a requirement that the value is unspecified, for example:

6. If the Regulator Mode is not NORMAL, the value of the Display Temperature is UNSPECIFIED.

   Rationale: In modes other than NORMAL, the value of the Display Temperature is not important and should not be used.

Recommended Practice 2.8.3: Ensure the detailed requirements are complete, i.e., an ideal value is assigned to each controlled variable and internal variable for every system state. Use a value of UNSPECIFIED when no meaningful assignment exists for a system state.

2.8.4 Ensure That Detailed Requirements are Consistent.

Care should also be taken to ensure the requirements are consistent, i.e., the requirements do not specify two values for a variable for the same system state. For example, if the first requirement was written as

1. If the Regulator Mode is NORMAL and the Current Temperature is less than or equal to the Lower Desired Temperature, the Heat Control shall be set to On.

It would conflict with the fourth requirement, since the first requirement would specify the heat control takes on the value of ON and the fourth requirement would specify that the heat control retain its previous value when the current temperature was less than or equal to the lower desired temperature.

Recommended Practice 2.8.4: Ensure the detailed requirements are consistent, i.e., only one ideal value is assigned to each controlled variable and each internal variable for every possible system state.

2.8.5 Ensure That Detailed Requirements are not Duplicated.

The requirements should also be checked to ensure the same requirement is not stated more than once, either through direct duplication or by having requirements in which modes and conditions overlap.

Recommended Practice 2.8.5: Ensure no two detailed requirements duplicate each other, i.e., specify the same outcome for overlapping modes and conditions.

2.8.6 Organize the Requirements.

The requirements for each controlled variable and internal variable are presented in the function that defines that variable. In this way, the dependency diagrams serve as a visual table of contents for the detailed requirements. For example, ideal value requirements for the heat control controlled variable are given in the Manage Heat Source Function in appendix A.5.1.3.

Recommended Practice 2.8.6: Present the detailed system requirements in the function that produces the variable being specified. This creates an organization in which the definition of each variable is directly traceable to its parent function.

2.8.7 Define Acceptable Latency for Each Controlled Variable.

Once the ideal value function is specified for each controlled variable and internal variable, the requirements are completed by specifying the tolerance and latency for each controlled variable.

The latency specifies the maximum lag between the time one or more monitored variables change value and an affected controlled variable must change its value. For example, if the latency for the Heat Control controlled variable is specified as 6 seconds, then the Heat Control

controlled variable must take on its new value within 6 seconds after the Current Temperature, or any other monitored variable, changes its value. The latency can be a single constant or it can be an arbitray function of the monitored and internal variables. Often, many controlled variables will share the same latency, and a single constant or function can be defined and assigned to several controlled variables. In any case, a rationale for the specified latency should always be included.

For the Heat Control controlled variable, a constant latency of 6 seconds is defined in the Manage Heat Source Function in appendix A.5.1.3. Note that a rationale for the latency is provided that traces back to the environmental assumptions (see table 14).

Table 14. Allowed Heat Source Latency Behavior

Name	Type	Value	Units	Physical Interpretation
Allowed Heat Source Latency	Real	6.0	Sec	The maximum time by which the Heat Source must be turned on or off to ensure acceptable operation of the Isolette system.
Rationale: Since a closed Isolette will warm or cool at a maximum rate of 1°F per minute (EA-IS1 and EA-IS2), turning the Heat Source on or off within 6 seconds ensures that the Current Temperature will not change by more than 0.1°F, the required accuracy and resolution of the Temperature Sensor (EA-TS2).				

Recommended Practice 2.8.7: Define the acceptable latency for each controlled variable along with the rationale for its value as part of the detailed system requirements.

2.8.8 Define Acceptable Tolerance for Each Controlled Variable.

When a controlled variable represents a numerical value (as opposed to a Boolean or enumerated value), the acceptable tolerance from the ideal value should also be specifed. For example, a controlled value that displays the altitude of the aircraft should include the delta above and below the actual altitude that can be tolerated. For Boolean and enumerated values, the tolerance is not usually important and should not be stated. For example, since the Heat Control controlled variable can only take on the values of on and off, its tolerance is specified as not applicable (N/A) (see appendix A.5.1.3).

Recommended Practice 2.8.8: Define the acceptable tolerance for each numerical controlled variable.

2.8.9 Do Not Define Latency and Tolerance for Internal Variables.

Note that tolerance and latency are not specified for internal variables. This is because the requirement is for the controlled variable to change with the specified latency and tolerance. No such requirements exist for the internal variables, which are simply aids used to break the specification of the ideal value function into manageable pieces.

Recommended Practice 2.8.9: Do not define latency and tolerance for internal variables.

The detailed system requirements for the Isolette Thermostat using shall statements are given in appendix A.5.

2.8.10 Alternative Ways to Specify Requirements.

There are other ways that the requirements could be specified besides shall statements. One approach is to use a graphical model to define the ideal value function and supplement it with the tolerances and latencies for each controlled variable. One disadvantage of this approach is that it is no longer obvious what constitutes an individual requirement.

Another approach is to use tables to specify the requirements. An example for specifying the ideal value of the Heat Control is shown in table 15.

In table 15, the value of the Heat Control is given in the bottom row as a function of the Regulator Mode and the current conditions. Each unshaded interior cell corresponds to a requirement and is labeled with the same identifiers as used in appendix A.5.1.3 to label the shall statements. Note that the Heat Control value specification in the bottom row can be a constant (e.g., on or off) or a function of the previous and current state (e.g., previous value). The main advantage of such a tabular presentation is that it is easier to confirm that the requirements are complete (an ideal value is defined for every mode and system state) and consistent (only one ideal value is defined for every mode and system state).

Table 15. Tabular Specification of Requirements

Regulator Mode	Condition		
INIT			REQ-MT1 ALWAYS
NORMAL	REQ-MT2 Current Temperature < Lower Desired Temperature	REQ-MT4 Lower Desired Temperature ≤ Current Temperature ≤ Upper Desired Temperature	REQ-MT3 Current Temperature > Upper Desired Temperature
FAILED			REQ-MT5 ALWAYS
Heat Control =	On	Previous Value	Off

MT = Monitor Temperature

Many different formats have been proposed for specifying functions using tables. SCR [2 and 3] and CoRE [4 and 5] make heavy use of condition tables (similar to table 15), event tables, and

mode tables. RSML [8] and SpecTRM [15] use a format called and/or tables. Several other tabular formats are described in reference 43.

However, the key issue is not whether the requirements are specified as shall statements or graphical models or tables, but that the requirements should specify the relationship the system will maintain between the monitored and controlled variables, and it should be stated in a way that is complete, consistent, unambiguous, and testable.

2.9 DEFINE THE SOFTWARE REQUIREMENTS.

Refer to table 16 for a description of main- and sublevel recommended practices for defining software requirements.

Table 16. Define the Software Requirements

Main- and Sublevel Recommended Practices	
2.9	Define the Software Requirements: With careful structuring, the software requirements and their architecture can map directly to the system requirements and their architecture. This recommended practice describes how to define the software requirements as a straightforward extension of the system requirements.
2.9.1	For each input the software must read, provide a description of anything the software developer must know to access and correctly interpret the input. This may include an input description, the data format, the range of values it may assume, its location, and any protocols to follow when accessing it.
2.9.2	For each input the software must read, provide a specification of its accuracy, where accuracy refers to the amount that its value may deviate from its ideal value.
2.9.3	For each input the software must read, provide a specification of its latency, where latency refers to the maximum time that its value may lag behind the true value of the monitored variable or variables it represents.
2.9.4	For each monitored variable, specify how to recreate an image of the monitored variable in software from the input variables.
2.9.5	For each monitored variable, specify how to recreate its status attribute from the input variables.
2.9.6	If design choices must be made when recreating a monitored variable in software that may affect the externally visible system behavior or system safety, flag those decisions as derived software requirements to be reviewed by the safety assessment process.

Table 16. Define the Software Requirements (Continued)

Main- and Sublevel Recommended Practices	
2.9.7	For each output the software must set, provide a description of anything the software developer must know to access and correctly set its value. This may include an output description, the data format, the range of values it may assume, its location, and any protocols to follow when accessing it.
2.9.8	For each output variable the software must set, provide a specification of its latency, where latency refers to the maximum allowed time from when the output variable is set until it changes the controlled variable value.
2.9.9	For each output the software must set, provide a specification of its accuracy, where accuracy refers to the amount the affected control variable can diverge from its ideal value.
2.9.10	For each controlled variable, specify how to set the output variables values based on the value of the controlled variable image in software.
2.9.11	For each controlled variable, confirm that the latency and accuracy specified in the system requirements can be met given the latency and accuracy of the input and output variables and the computation time of the software.

At some point, it becomes necessary to allocate the system requirements to hardware and software. As the use of general-purpose processors has grown, more and more of the system requirements are allocated to software. In fact, the detailed system requirements and the software requirements often look remarkably alike. However, there are usually enough differences that the system requirements and the software requirements are treated as two separate specifications. This treatment is largely because the inputs and outputs to the software do not exactly match the monitored and controlled variables that form the basis of the system requirements. This section describes an approach that provides a seamless transition from the system requirements to the software requirements in which the software requirements are created by extending the system requirements.

The basis for the approach of this section is the four-variable model developed by Parnas and Madey [2] as part of the SCR methodology. The four-variable model makes clear the relationship between system and software functional requirements so that the software requirements can be specified as an addition to the system requirements. An overview of the four-variable model is shown in figure 11.

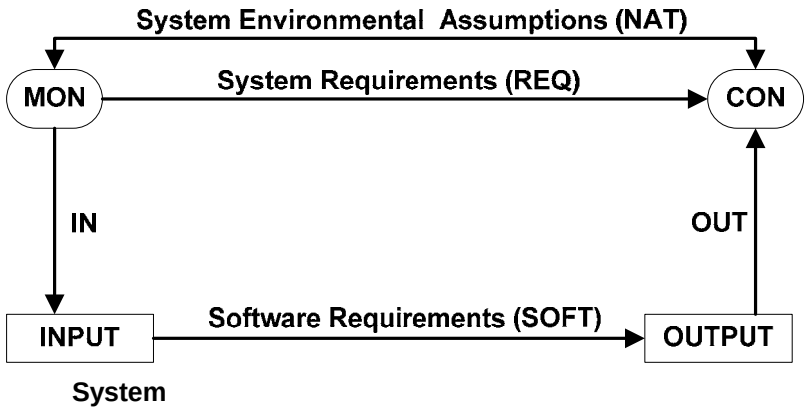


Figure 11. The Four-Variable Model

The monitored and controlled variables, the system environmental assumptions, and the system requirements have been discussed extensively already. The monitored variables (MON) consist of the quantities in the environment the system will monitor, and the controlled variables (CON) are the quantities in the environment the system will control. The environmental assumptions (NAT) are the relationships maintained by the environment on which the system depends, while the system requirements (REQ) define the relationship the system will maintain between the monitored and controlled variables.

The monitored and controlled variables cannot be directly sensed or controlled by the software. Instead, the monitored variables must be provided to the software by the system as input (INPUT) variables the software can read. In similar fashion, the system must translate the output (OUTPUT) variables the software can write into changes upon the controlled variables in the environment. The monitored and controlled variables are typically at a higher level of abstraction than the input and output variables. For example, while a monitored variable might be the altitude of an aircraft reported by a radar altimeter that is an integer between -20 and 2500 feet, the corresponding input variable might be an ARINC 429 bus word in which the altitude is encoded as a 17-bit 2's complement integer located in bits 13 to 28 of the bus word with the sign in bit 29 and the decimal point located between bits 15 and 16 representing a value between -8192.0 and +8191.875.

The IN relation defines the relationship of each input variable to one or more monitored variables. This consists of the definition of the input variables ideal value function, its latency, and its accuracy. The ideal value function defines the ideal value of the input variable as a function of one or more monitored variables. The latency specifies the maximum time from when a monitored variable (that the input variable depends on) is changed until that change is indicated by the input variable. The accuracy specifies the amount by which the input variable's actual value may deviate from its ideal value.

In similar fashion, the OUT relationship defines the relationship of each controlled variable to one or more output variables. This includes a definition of the controlled variable's ideal value function, its latency, and its accuracy. The ideal value function defines the controlled variable's ideal value as a function of one or more output variables. The latency specifies the maximum

time from when an output variable (that the controlled variable depends on) is changed until that change is indicated by the controlled variable. The accuracy specifies the amount by which the controlled variable's actual value may deviate from its ideal value.

Specification of the NAT, REQ, IN, and OUT relations implicitly bounds the allowed software behavior, shown in figure 11, as the SOFT relation, without specifying the software design. While the SOFT relation defines the true requirements for the software, the mapping between the system requirements specified in REQ and the software requirements in SOFT is not obvious. The system requirements and the software requirements are at different levels of abstraction and have different domains and ranges. One way to address these concerns is to extend the software requirements SOFT into three parts, IN', REQ', and OUT' [44], as shown in figure 12.

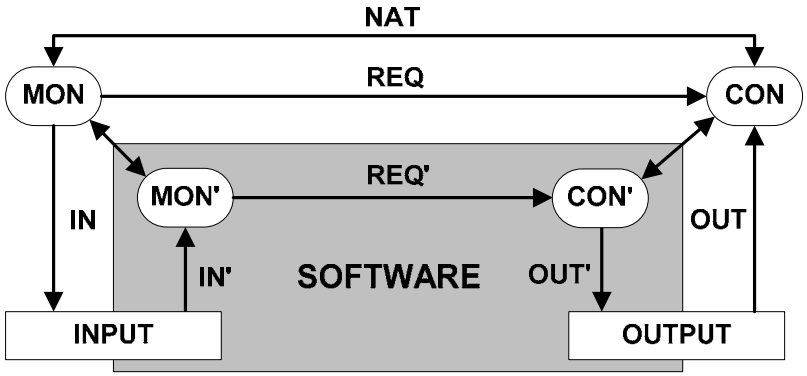


Figure 12. Extended Software Requirements

In figure 12, IN' is the inverse of IN and defines how to recreate an image of the monitored variables (MON') in software from the INPUT variables. Similarly, OUT' is the inverse of OUT and defines how the OUTPUT variables will change as an image of the controlled variables (CON') is changed in software. Just as REQ describes the relationship that must be maintained by the system between the monitored and controlled variables, REQ' describes the relationship that must be maintained by the software between the monitored and controlled variable images. The SOFT relation specifying the software requirements is replaced by IN', REQ', and OUT'.

The primary advantage of this extension is that it makes the relationship between the system requirements and the software requirements direct and straightforward. The ideal value function defined in the system requirement, REQ maps directly onto an identical function in the software requirement, REQ'. As a result, the software requirements actually consist of an addition to the system requirements defining the INPUT and OUTPUT variables and the IN, OUT, IN', and OUT' relations. Another advantage is that since IN' and OUT' are implemented in software (it may be helpful to think of them as the specification for hardware drivers), they provide a useful separation of concerns in the software requirements; REQ' will change as the system requirements change, while IN' and OUT' will change as the underlying hardware changes. This helps to insulate the software implementing the system requirements from changes in the hardware platform.

It should be noted that that MON' and CON' are not the same as the system-level variables represented by MON and CON. They are the monitored and controlled variable images recreated in the software. Small differences in value are introduced by the hardware and software, and latencies are introduced between the images and the actual quantities in the environment. For example, the value of an aircraft's altitude recreated in software is always going to lag behind, and differ somewhat, from the aircraft's true altitude. These differences must be taken into account to ensure that the allowed latencies and tolerances specified for the controlled variables are met. For the sake of clarity in the following discussion, a prime (') will be used to distinguish a specific monitored or controlled variable recreated in software (for example, airspeed') from the actual monitored or controlled variable (airspeed). In practice, this distinction is usually clear from the context.

2.9.1 Specify the Input Variables.

To make these ideas concrete, consider the Isolette Thermostat example from appendix A. The first step was to define the input variables. The objective of this is to provide the software engineer with all the information necessary to correctly read and interpret the input variables. How this is to be done will vary with each type of hardware interface.

For example, the Current Temperature monitored variable is specified in appendix A.3.2 to be a real number between 68.0° and 105.0°F with a precision of at least 0.1°F. Information about this monitored variable is provided to the thermostat software by two input variables, "Curr Temp In" and "Curr Temp Status In."

The Curr Temp In input variable is defined in table 17 as a 16-bit unsigned integer that represents the current temperature multiplied by 256 (i.e., 8 bits are used to represent fractions of a degree). This integer will be found in the two contiguous 8-bit memory words located at address X'0048' and X'0049'.

Table 17. Input Variable Curr Temp In

Description	Current Temperature of the Isolette in °F multiplied by 256
Data Representation	16-bit unsigned integer
Values	[0..65,535]
Location	Memory words X'0048' and X'0049'
Ideal Value	Current Temperature multiplied by 256
Latency	0.20 millisecond
Accuracy	±20 (i.e., ±0.08°F)

Recommended Practice 2.9.1: For each input the software must read, provide a description of anything the software developer must know to access and correctly interpret the input. This may include an input description, the data format, the range of values it may assume, its location, and any protocols to follow when accessing it.

2.9.2 Specify the Accuracy of Each Input Variable.

The accuracy of Curr Temp In is specified to be at least ±20, which is approximately ±0.08°F. This is the maximum deviation introduced by the hardware from the true value of the Current Temperature monitored variable, which is defined to be the value produced by the Temperature Sensor.[15]

Recommended Practice 2.9.2: For each input the software must read, provide a specification of its accuracy, where accuracy refers to the amount that its value may deviate from the ideal value.

2.9.3 Specify the Latency of Each Input Variable.

In similar fashion, the latency specifies the maximum time from when the monitored variable is changed until that change is indicated by the input variable. For Curr Temp In, it is specified to be 0.20 millisecond (msec). The accuracy and latency are needed to determine how accurately and how quickly the software must complete its computation. This is discussed in section 2.9.11.

Recommended Practice 2.9.3: For each input the software must read, provide a specification of its latency, where latency refers to the maximum time that its value may lag behind the true value of the monitored variable or variables it represents.

2.9.4 Specify IN' for Each Monitored Variable.

Note that the information presented in table 17 describes both the input variable Curr Temp In and the essential part of its IN relation, i.e., the ideal function defining how to compute the input variable from the monitored variable(s), its latency, and its accuracy.

The other input variable used to provide the software with information about the Current Temperature value is Curr Temp Status In, which specifies whether the value of Curr Temp In is valid or not. Its definition is shown in table 18. If the value of Curr Temp In is believed to be valid, it is set to 0. Otherwise, it is assigned a value other than 0. This information is presented as an 8-bit unsigned integer located at memory address X'0046'. Note that its initial value is 255. This ensures that Curr Temp In is treated as invalid until it has been successfully read at least once.[16] The ideal value describing how the value of the input variable relates back to the monitored variables provides the minimum information needed to implement the system requirements. This could be extended by assigning specific values to different types of failures, but since the system requirements specify the same behavior regardless of the failure type, this information is not necessary.

[15] Note that the Temperature Sensor may introduce its own inaccuracy and latency in sensing the true temperature in the Isolette.

[16] Recall that the status attribute of all monitored variables should be initialized to invalid.

Table 18. INPUT Variable Curr Temp Status In

Description	Indication of whether Curr Temp In is valid
Data Representation	8-bit unsigned integer
Values	[0 .. 255]
Initial Value	255
Location	Memory word X'0046'
Ideal Value	No error in sensing Curr Temp In → 0, Error in sensing Curr Temp In → 1 to 255
Latency	0.20 msec
Accuracy	N/A

The next step is to define the IN' relation describing how to recreate the Current Temperature monitored value in software. The IN' relation can be a simple one-to-one mapping from an input variable to a monitored variable, or it can be a complex mapping from several input variables to a monitored variable. For example, the IN' relation describing how to recreate the value of Current Temperature' is a simple function of the Curr Temp In input variable as shown in table 19.

Table 19. IN' Relation for Value of Current Temperature'

	Curr Temp In < 17,408	17,408 ≤ Curr Temp In ≤ 26,880	Curr Temp In > 26,880
Value =	68.0	Curr Temp In/256.0	110.0

If the value of Curr Temp In is less than 17,408, the value of Current Temperature' is set to 68.0°F. If Curr Temp In is greater than or equal to 17,408 and less than or equal to 26,880, the value of Current Temperature' is set to Curr Temp In divided by 256.0. If the value of Curr Temp In is greater than 26,880, the value of Current Temperature' is set to 105.0°F.

Recommended Practice 2.9.4: For each monitored variable, specify how to recreate an image of the monitored variable in software from the input variables.

2.9.5 Specify the Status of Each Monitored Variable.

Saturating the computed value of Current Temperature' at 68.0° and 105.0°F is done because environmental assumption EA-TS3 for the temperature sensor states that the Current Temperature monitored variable will be between 68.0° and 105.0°F, and there may be other parts of the specification that depend on this assumption. The fact that the Curr Temp In input variable can physically contain a value that (after conversion) ranges from 0.0° to 256.0°F is an artifact of the standard word size of the implementation rather than a requirement.

As discussed in section 2.4, every monitored variable should have a status attribute associated with it to indicate its health, i.e., how well it can be sensed by the system. How to set the status attribute of the recreated monitored variable should be specified in exactly the same manner as specifying how to set its value. As an example, the specification for setting the status attribute of the Current Temperature monitored variable is shown in table 20. Here, the status is set to Invalid if the Curr Temp Status In input variable is not 0 or if the Curr Temp In value is below 17,409 or greater than 16,880.

Table 20. IN' Relation for Status of Current Temperature'

		Curr Temp Status In = 0	
	Curr Temp Status In ≠ 0	Curr Temp In < 17,408 OR Curr Temp In > 26,880	17,408 ≤ Curr Temp In AND Curr Temp In ≤ 26,880
Status =	Invalid	Invalid	Valid

Recommended Practice 2.9.5: For each monitored variable, specify how to recreate its status attribute from the input variables.

2.9.6 Flag Design Decisions as Derived Requirements.

The choice to saturate the value of Current Temperature' is an example of a derived requirement, as defined in RTCA DO-178B [27] and DO-248B [28], i.e., a requirement that is not directly traceable to a higher-level requirement. This is indicated by the fact that other design choices could be made that would affect the externally visible system behavior, while still meeting the system requirements. For example, the range shown in table 20 could be expanded to 17,000 to 27,000. This would reduce the risk of putting the thermostat into a failed state due to noise in the value of Curr Temp In, but opens the possibility of operating with a dangerously high temperature. Decisions such as these that affect the externally visible system behavior should be flagged as a derived software requirement to ensure they are provided to the system safety assessment process, as called out in DO-178B.

Recommended Practice 2.9.6: If design choices must be made when recreating a monitored variable in software that may affect the externally visible system behavior or system safety, flag those decisions as derived software requirements to be reviewed by the safety assessment process.

2.9.7 Specify the Output Variables.

The definition of the OUTPUT variables and the OUT and OUT' relations are done in much the same way, except that the ideal value, latency, and accuracy are specified for the controlled variable, not the output variable. For example, the heat source can be turned on and off by the thermostat by writing to the heat control out output variable. The definition of this variable is shown in table 21.

Table 21. OUTPUT Variable Heat Control OUT

Description	Command to turn heat source on or off
Data Representation	8-bit unsigned integer
Values	[0 .. 255]
Location	Memory word X'0060'
Ideal Value	0 → heat control = off 2 to 255 → heat control = on
Latency	0.60 msec
Accuracy	N/A

This output variable is an 8-bit unsigned integer located at memory address X'0060'. A value of 0 commands the heat source off, a value of 2 to 255 commands the heat source on.[17]

Recommended Practice 2.9.7: For each output the software must set, provide a description of anything the software developer must know to access and correctly set its value. This may include an output description, the data format, the range of values it may assume, its location, and any protocols.

2.9.8 Specify the Latency of Each Output Variable.

Just as with the input variables, the latency and accuracy of the output variables should also be specified. However, for output variables, the latency specifies the maximum possible lag between the time the output variable is set until it changes the value of the controlled variable.

Recommended Practice 2.9.8: For each output variable the software must set, provide a specification of its latency, where latency refers to the maximum allowed time from when the output variable is set until it changes the controlled variable value.

2.9.9 Specify the Accuracy of Each Output Variable.

For output variables, the accuracy defines how much the controlled variable can diverge from the ideal value determined by the output variable. For a numerical quantity, this should be set to a range. For a discrete output variable, the accuracy should be set to N/A.

As with the input variables, the specification of this output variable includes the output variable definition itself and the essential part of the OUT relation needs to implement the requirements, i.e., the ideal function defining the value of controlled variable from the output variable and the controlled variable's latency and accuracy.

[17] The value 1 is not used to ensure that the commands to turn the heat control on or off differ by more than a single bit.

Recommended Practice 2.9.9: For each output the software must set, provide a specification of its accuracy, where accuracy refers to the amount the affected control variable can diverge from its ideal value.

2.9.10 Specify OUT' for Each Controlled Variable.

The OUT' relation is defined for each output variable. This defines how to set the output variable based on the current value of one or more images of controlled variables in software. For example, the value of the output variable Heat Control Out is a direct mapping from the image of the Heat Control' controlled variable maintained in software, as shown in table 22. Note that when the image of the Heat Control' controlled variable in software is ON, the implementer can choose to set the value of Heat Control Out to any value from 1 to 255 since they all have the same effect.

Table 22. OUT' Relation for Heat Control

	Heat Control' = Off	Heat Control' = On
Value =	0	[1..255]

With the definition of the INPUT and OUTPUT variables and the IN' and OUT' relationships, the software requirements specification is essentially complete. The software developer now knows how to recreate the monitored variable images in software and how to set the output variables based on the images of the controlled variables in the software. The other information the software developer needs to know is how to change the controlled variable image in software when the monitored variable image in software change (i.e., REQ'). However, the ideal value function for REQ' is identical to the ideal value function defined for the detailed system requirements (i.e., the REQ relation) defined in section 2.8.

Recommended Practice 2.9.10: For each controlled variable, specify how to set the output variables values based on the value of the controlled variable image in software.

2.9.11 Confirm Overall Latency and Accuracy.

The one remaining task is to confirm that the overall latency and accuracy specified for each controlled variable can be met given the latency and accuracy of the input and output variables and the computation time of the software. For example, the latency and accuracy of the heat control controlled variable is specified to be 6 seconds and N/A in appendix A.5.1.3. The latency introduced in sensing the current temperature is specified in table 18 to be 0.20 msec. The latency in setting the value of the Heat Control controlled variable is specified in table 21 to be 0.60 msec. Thus, the total latency in sensing the monitored variables and setting the controlled variables is 0.8 seconds, indicating that the software must complete its computations in 5.2 seconds. Since the Heat Control is a discrete two-valued variable, the confirmation of its accuracy is immediate.

DO-178B [27] specifies that the software requirements should be organized into high-level and low-level software requirements. Also, the high-level software requirements should comply with the system requirements (objective 1 of table A-3 in appendix A), and the low-level software requirements should comply with the high-level software requirements (objective 1 of table A-4 in appendix A). These are defined in DO-178B as

> "High-level requirements: Software requirements developed from analysis of system requirements, safety-related requirements, and system architecture.
>
> Low-level requirements: Software requirements derived from high-level requirements, derived requirements, and design constraints from which source code can be directly implemented without further information."

Which of the requirements discussed so far are high-level software requirements, which are low-level software requirements, and how can compliance of the software requirements with the system requirements be shown? These relationships are clarified in figure 13.

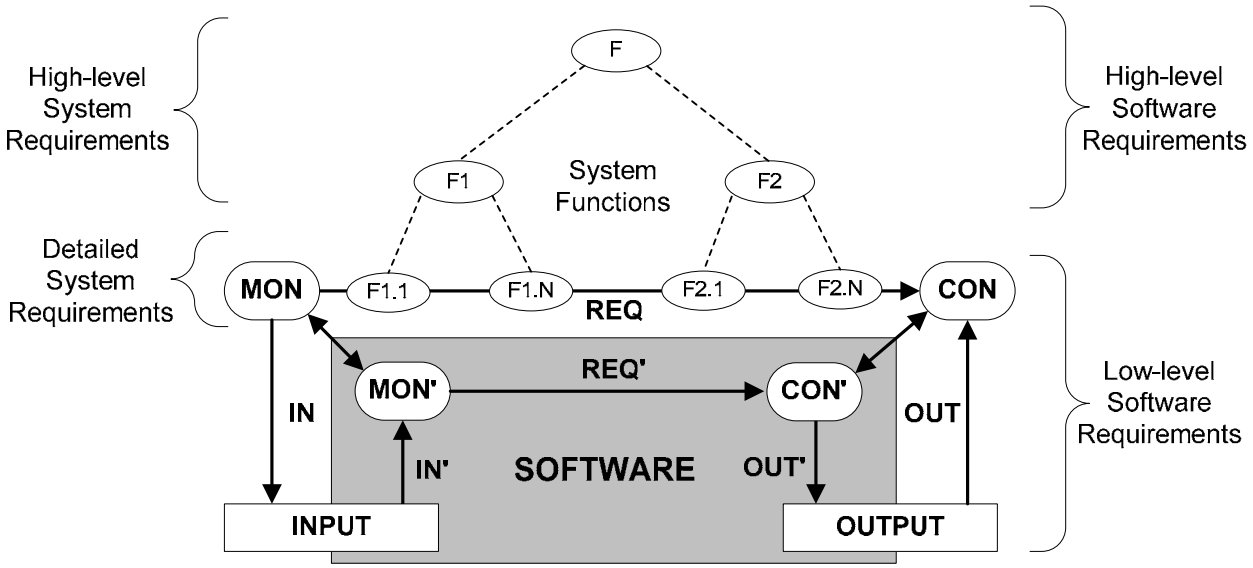


Figure 13. High- and Low-Level Software Requirements

In figure 13, the nested System Functions structure is depicted above the four-variable model. The lowest-level System Functions (F1.1…F2.N) define the detailed system behavioral[18] requirements, as discussed in section 2.8. In other words, each low-level System Function defines its part of the REQ relation between the monitored and controlled variables. The higher-level System Functions (F, F1, and F2) define the high-level system requirements, as discussed in section 2.6.

[18] The latency and tolerance defined for each controlled variable are end-to-end requirements and are more appropriately viewed as high-level requirements.

The low-level software requirements, i.e., those from which source code can be directly implemented without further information, include IN', REQ' (whose behavior portion is identical to REQ), and OUT'. In other words, the low-level software requirements consist of the detailed system requirements along with the definitions of IN' and OUT'.

Since the behavioral software requirements are identical to the behavior system requirements, the obvious choice for the high-level software requirements are the system requirements that are not also low-level software requirements, i.e., the dependency diagrams, the internal variable definitions, and any high-level system requirements, as discussed in section 2.5. This is a natural consequence of implementing the system requirements directly in software.

For example, for the Isolette Thermostat, the high-level software requirements would be those defined for the Thermostat Function (A.5), the Regulate Temperature Function (A.5.1), and the Monitor Temperature Function (A.5.2). The low-level software requirements would be those defined for all the lowest-level functions, such as the Manage Regulator Interface Function (A.5.1.1) and all the IN' and OUT' relations.

Recommended Practice 2.9.11: For each controlled variable, confirm that the latency and accuracy specified in the system requirements can be met given the latency and accuracy of the input and output variables and the computation time of the software.

2.10 ALLOCATE SYSTEM REQUIREMENTS TO SUBSYSTEMS.

Refer to table 23 for a description of main- and sublevel recommended practices for allocating system requirements to subsystems.

Table 23. Allocate System Requirements to Subsystems

Main- and Sublevel Recommended Practices	
2.10 Allocate System Requirements to Subsystems: For large systems, the requirements must be allocated to subsystems that can be developed independently by subcontractors. This practice describes how this can be done in a manner that is consistent with the other recommended practices.	
2.10.1	Complete enough of the parent system functional architecture (taking into account implementation constraints) to be able to identify functions that are likely candidates to be allocated to subsystems.
2.10.2	When allocating a system function to a separate subsystem, duplicate the high-level requirements of the function in the subsystem. This provides direct traceability between the topmost subsystem requirements to the system function requirements.
2.10.3	Develop a system overview for each subsystem specification. This provides an important high-level view of the system needed by the subcontractor and helps to clarify the scope the subsystem and clarify its relationship to the system.

Table 23. Allocate System Requirements to Subsystems (Continued)

Main- and Sublevel Recommended Practices	
2.10.4	Identify monitored and controlled variables for the subsystem that are shared with the parent system. Ensure their definition in the subsystem specification is consistent with their definition in the system specification. Do not include monitored and controlled variables from the system specification that are not used by the subsystem.
2.10.5	Create new monitored and controlled variables for the internal system variables that are exposed by allocating System Functions to the subsystem. Ensure these are consistent with the definitions of the internal variables in the system specification.
2.10.6	Specify the operational concepts for subsystem in order to provide the developers with an understanding of how the system will be used and how it will interact with its environment.
2.10.7	Identify any environmental assumptions the subsystem shares with its parent system. Document these environmental assumptions in the subsystem specification and ensure they are consistent with those for the parent system.
2.10.8	Identify any environmental assumptions associated with the new monitored and controlled variables. Ensure these are consistent with the internal variables defined in the parent system.
2.10.9	Complete the subsystem requirements specification by • completing the functional decomposition. • modifying the functional architecture to take into account safety requirements or implementation constraints. • identifying the subsystem modes. • developing the detailed behavior and performance requirements for the subsystem. • developing software requirements if the subsystem will be implemented in software.
2.10.10	Ensure the latencies and tolerances specified for the controlled variables in the subsystems are consistent with the end-to-end latencies and tolerances specified for the overall system.

The previous sections discussed how to develop and specify the requirements for a single system by defining the relationships the system will maintain between its monitored and controlled variables. Section 2.9 discussed how to transition seamlessly from the system requirements to the software requirements. However, to handle the complexity of large systems, it is standard practice to divide the system into subsystems that can be developed independently and to allocate the system requirements to these subsystems [24 and 45]. This section discusses how this can be done within the framework described herein.

2.10.1 Identify Subsystem Functions.

Consider the situation shown in figure 14. Here, the contractor specifying the requirements for system 1 (contractor 1) identified the monitored and controlled variables and completed the functional decomposition through functions F1.1 … F1.N, F2, and F3.1 … F3.N. At this point (or perhaps even sooner), the contractor decides to spin-off functions F1 and F3 as separate subsystems for development by independent subcontractors.

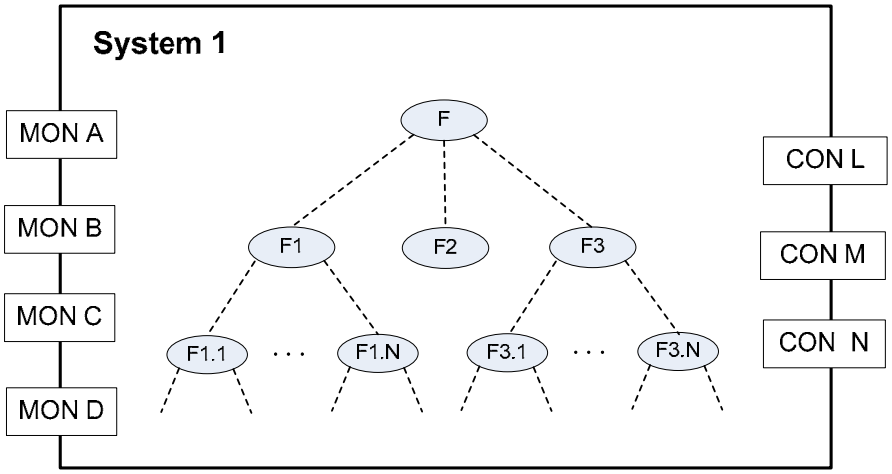


Figure 14. Functional Decomposition of System 1

An overview of the approach for doing this is illustrated in figure 15. Here, function F1 and its child functions were moved to the specification for system 1.1, and function F3 and its child functions were moved to the specification for system 1.3. The objective is to create the entire system 1 requirements specification for use by contractor 1, a requirements specification for system 1.1 for use by both contractor 1 and subcontractor 1.1, and a requirements specification for system 1.3 for use by both contractor 1 and subcontractor 1.3.

The requirements for system 1 will be completed and maintained by contractor 1. It contains a specification for the entire system that the contractor may not wish to share with subcontractor 1.1 and subcontractor 1.3. However, the specification for system 1 will not contain all the detailed requirements for functions F1 and F3.[19] Instead, the detailed requirements for function F1 will be developed and refined cooperatively by contractor 1 and subcontractor 1.1. In a similar manner, the detailed requirements for function F3 will be developed cooperatively by contractor 1 and subcontractor 1.3.

[19] Specification for system 1 may contain the detailed requirements for function F2 that were not allocated to a separate subsystem.

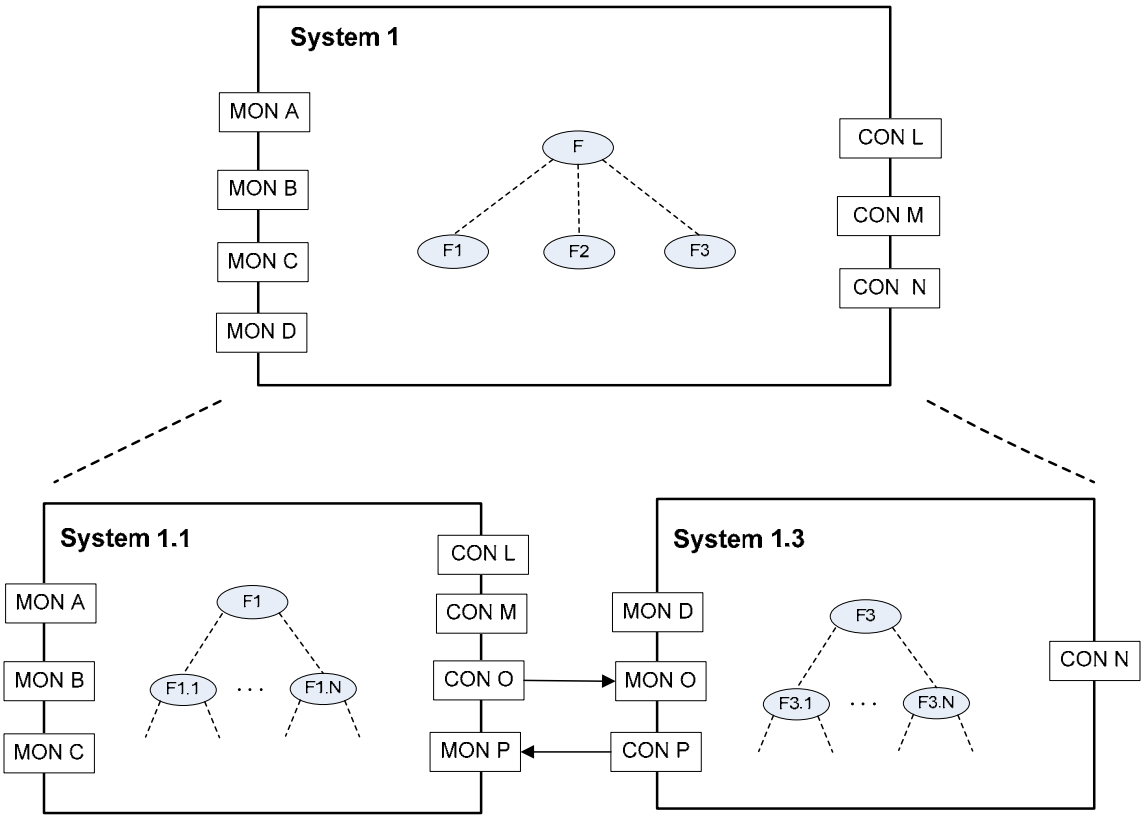


Figure 15. Decomposition of System 1 Into Subsystems

The division of labor between the contractor and the subcontractor in developing the requirements for either subsystem will likely depend on the confidence the contractor has in the subcontractor. If the contractor has high confidence in the subcontractor, the contractor may provide only the initial subsystem requirements specification and allow the subcontractor to do most of the work of refining it into a complete requirements specification. If the contractor does not have high confidence in the subcontractor, the contractor may elect to do all of the refinement and provide the subcontractor with a complete requirements specification. Typically, the subsystem specification will not be completed when the actual legal contract is signed, but the allocation of responsibilities for completing the subsystem requirements specification will be specified in the contract.

The purpose of the subsystem requirements specification is to provide a common specification that the contractor and the subcontractor share. However, it may not be necessary for the subcontractor to share all information with the contractor. There may be details, such as algorithms, or even some detailed requirements, that the subcontractor considers proprietary. This may result in the situation in which the subcontractor maintains some portion of the subsystem specification separately.

Recommended Practice 2.10.1: Complete enough of the parent functional architecture system (taking into account implementation constraints) to be able to identify functions that are likely candidates to be allocated to subsystems.

2.10.2 Duplicate Overlapping System to Subsystem Functions.

What information should be in the initial requirements specification for the subsystem? As shown in figure 15, function F1 and its child functions were moved into subsystem 1.1, and function F3 and its child functions were moved into subsystem 1.3. This implies that the function hierarchy for F1 and F3, and whatever requirements developed for that hierarchy, are now part of the specification of subsystem 1.1 and subsystem 1.3. This ensures that no work done by the contractor is discarded. If the decomposition into subsystems was planned, the contractor may not have developed any child functions for F1 and F3.

In addition, figure 15 also shows that a copy of functions F1 and F3 (but not their child functions) is retained along with their high-level requirements in the specification for system 1. Duplication of the high-level requirements for F1 and F3 in the specification of system 1 is done deliberately to provide traceability and continuity between system 1 and subsystem 1.1. Each high-level requirement of subsystems 1.1 and 1.3 traces directly to an identical requirement in system 1.

At this point, the requirements specification for subsystems 1.1 and 1.3 consist of little more than the functions copied from system 1 and their high-level requirements. As discussed in sections 2.1 through 2.9, a complete requirements specification consists of much more than just a list of shall statements. The next step is to fill in the information needed to make the subsystem specifications a complete requirements specification in its own right. This consists of adding a system overview, monitored and controlled variables, operational concepts, and environmental assumptions to each subsystem specification. Once this is done, the functional decomposition of the subsystems can be continued until the detailed behavior and performance requirements are specified.

An example of this is shown in figure 16 and appendices B, C, and D. In figure 16, a specification for a simple FCS (appendix B) is decomposed into a Flight Guidance (FG) Function and an AP Function. These functions of the FCS are split off into a subsystem specification for a simple FGS (appendix C) and a simple AP system (appendix D).

The specification for the overall FCS (appendix B) contains both the FG and the AP Functions. The system overview for the FCS can be found in appendix B.1. As shown in its context diagram (figure B-1 in appendix B), it monitors the Aircraft Attitude from the Attitude Heading System (AHS) and the Pilot Inputs from the Flight Crew Interface (FCI). It sets the values of the Flight Director (FD) Guidance, FCS Failed, and AP Status controlled variables provided to the Primary Flight Displays (PFD), as well as the Actuator Commands controlled variable provided to the Control Surface Actuators.

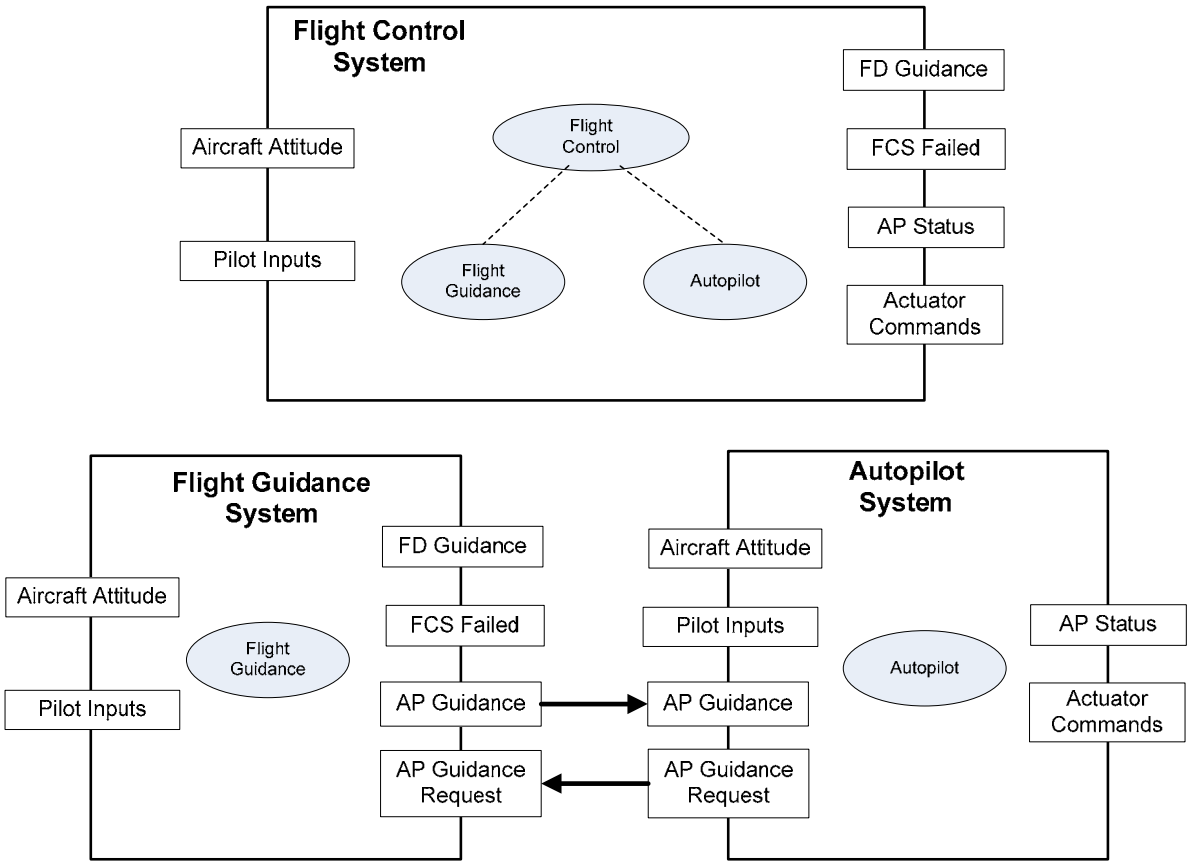


Figure 16. Allocation of FCS Requirements Into Subsystems

The operational concepts of the FCS are specified as use cases in appendix B.2. The external entities with which the FCS interacts are defined, along with their monitored and controlled variables and their environmental assumptions, in appendix B.3. The System Functions for the FCS, along with their high-level requirements, are defined in appendix B.4. In particular, the high-level requirements for the FG Function are specified in appendix B.4.1, and the high-level requirements for the AP Function are specified in appendix B.4.2.

The high-level requirements for the FG Function are repeated as the topmost requirements of the FGS subsystem in appendix C.4. In a similar fashion, the high-level requirements for the AP Function are repeated as the topmost requirements of the Autopilot subsystem in appendix D.4. This duplication connects the requirements of the subsystem specifications to their parent system specification.

Recommended Practice 2.10.2: When allocating a system function to a separate subsystem, duplicate the high-level requirements of the function in the subsystem. This provides direct traceability between the topmost subsystem requirements to the system function requirements.

2.10.3 Develop a System Overview for Each Subsystem.

The next step in developing the FGS subsystem specification is to complete its system overview. A new overview describing the purpose, context(s), and goals of the FGS subsystem is given in appendix C.1. This information provides the high-level view needed by the subcontractor of the subsystem.

Recommended Practice 2.10.3: Develop a system overview for each subsystem specification. This provides an important high-level view of the system needed by the subcontractor and helps to clarify the scope the subsystem and clarify its relationship to the system.

2.10.4 Identify the Subsystem Monitored and Controlled Variables.

Just as described in section 2.2, the next step consists of defining the subsystem boundary by identifying the monitored and controlled variables of the subsystem. Note that the FGS subsystem shares many of the same external interfaces with the FCS. Both systems interact with the FCI, the AHS, and the PFD, and the definitions of the monitored and controlled variables for these external entities should be consistent with those in the FCS specification. At the same time, there are differences that need to be accounted for (see figure 16). For example, the FGS does not use the engage AP field of the Pilot Inputs monitored variable, and it does not set the AP Status controlled variable provided to the PFD. It also does not set the Actuator Commands controlled variables provided to the Control Surface Actuators.

Recommended Practice 2.10.4: Identify monitored and controlled variables for the subsystem that are shared with the parent system. Ensure their definition in the subsystem specification is consistent with their definition in the system specification. Do not include monitored and controlled variables from the system specification that are not used by the subsystem.

2.10.5 Create New Monitored and Controlled Variables.

The biggest difference between the FCS boundary and the FGS subsystem boundary is that the FCS interacts directly with the Control Surface Actuators, while the FGS interacts directly with the AP subsystem. As a result, the FGS has one new controlled variable (AP Guidance) that the FGS provides to the AP subsystem and one new monitored variable (AP Guidance Request) that the AP subsystem provides to the FGS subsystem. These correspond to the internal variables shown in the dependency diagram for the FCS (figure B-2 in appendix B). Allocating the FCS system into two separate subsystems exposes these internal variables, which are now treated as monitored and controlled variables of the FGS and AP subsystems.

The monitored and controlled variables for the FGS subsystem are defined along with the definition of its external entities in appendix C.3.

Recommended Practice 2.10.5: Create new monitored and controlled variables for the internal system variables that are exposed by allocating System Functions to the subsystem. Ensure these are consistent with the definitions of the internal variables in the system specification.

2.10.6 Specify the Subsystem Operational Concepts.

As described in section 2.3, the next step is to define the operational concepts for the new subsystem. This would describe how the subsystem will interact with its operators and other subsystems. Just as with the system overview, this provides important context for the subcontractors and helps to identify what functions the subsystem should provide, what information is needed to invoke those functions, and what information the system needs to provide back to its operators and other systems. One means to do this is to develop use cases for the subsystem. Other approaches defined by the contractor and the subcontractor could also be used, with the scope and level of rigor depending on the familiarity of the subcontractor with the operation of the subsystem.

Placeholders for the operational concepts of the FGS and AP systems are provided in appendices C.2 and D.2, but no examples are provided, since several examples of use cases were previously developed.

Recommended Practice 2.10.6: Specify the operational concepts for subsystem to provide the developers with an understanding of how the system will be used and how it will interact with its environment.

2.10.7 Identify Subsystem Environmental Assumptions Shared With Parent System.

The next step consists of identifying the environmental assumptions for the new subsystem. If the system and subsystem share several of the same interfaces to external entities, many of the environmental assumptions will be found in the system specification. This is particularly true of the types, ranges, and units of the monitored and controlled variables.

Recommended Practice 2.10.7: Identify any environmental assumptions the subsystem shares with its parent system. Document these environmental assumptions in the subsystem specification and ensure they are consistent with those for the parent system.

2.10.8 Identify Environmental Assumptions of the New Monitored and Controlled Variables.

Of course, the types, ranges, and units of the new monitored and controlled variables will need to be defined. These should be consistent with the definitions of the internal variables found in the parent specification. An effort should also be made to ensure that more complex environmental assumptions relating the new monitored and controlled variables to the other monitored and controlled variables of the subsystem are identified. These assumptions may not have been identified in the parent system, since relationships between internal variables and monitored and controlled variables are usually implicitly captured as part of the detailed behavior and performance requirements.

Recommended Practice 2.10.8: Identify any environmental assumptions associated with the new monitored and controlled variables. Ensure these are consistent with the internal variables defined in the parent system.

2.10.9 Complete the Subsystem Requirements Specification.

At this point, the initial version of the subsystem requirements specification is complete. It contains high-level requirements that can be directly traced back to the System Function from which it was derived. It also contains a system overview, monitored and controlled variables, operational concepts, and environmental assumptions that are consistent with the parent system. An example of such an initial specification for the FCS is provided in appendix C. The example for the AP system is provided in appendix D.

Of course, the subsystem requirements specification is not yet complete. At this point, the contractor and subcontractor are ready to continue the development of the subsystem requirements until the detailed behavior and performance requirements are specified. As discussed in section 2.10.1, this task is performed by both the contractor and the subcontractor, though the division of labor between them will probably be based on the contractor's confidence in the subcontractor.

The first task in this process is completion of the functional decomposition of the subsystem as discussed in section 2.5. Once the major subsystem functions have been identified, the PSSA should be resumed for the subsystem, as discussed in section 2.6, to identify any derived system safety requirements. The functional architecture of the subsystem should be modified to take into account these safety requirements or other implementation constraints. The major subsystem modes should be identified. As discussed in section 2.7, the detailed behavior and performance requirements should be developed, as discussed in section 2.8, and if the subsystem will be implemented in software, the software requirements should be developed, as discussed in section 2.9.

Recommended Practice 2.10.9: Complete the subsystem requirements specification by

- completing the functional decomposition.
- modifying the functional architecture to take into account safety requirements or implementation constraints.
- identifying the subsystem modes.
- developing the detailed behavior and performance requirements for the subsystem.
- developing software requirements if the subsystem will be implemented in software.

2.10.10 Ensure Latencies and Tolerances are Consistent.

Developing the detailed behavior and performance requirements for each subsystem will include specifying the latency and tolerance for each controlled variable in the subsystem. These should also be reviewed to ensure they are consistent with the system specification. The sum of the latencies for the controlled variables in the subsystems should at least be less than the end-to-end

latency specified for the corresponding controlled variable in parent system.[20] For example, in figure 16, the sum of the latencies specified for the FGS controlled variable AP Guidance and the AP controlled variable Actuator Commands should be less than the end-to-end latency specified for the FCS controlled variable Actuator Commands. In a similar fashion, the composition of the tolerances specified for subsystem controlled variables should be checked to ensure they are consistent with the end-to-end tolerance specified for the corresponding controlled variable in the parent system.

Recommended Practice 2.10.10: Ensure the latencies and tolerances specified for the controlled variables in the subsystems are consistent with the end-to-end latencies and tolerances specified for the overall system.

2.11 PROVIDE RATIONALE.

Refer to table 24 for a description of main- and sublevel recommended practices for providing rationale.

Table 24. Provide Rationale

Main- and Sublevel Recommended Practices	
2.11	Provide Rationale: Within each recommended practice, the requirements developers are encouraged to provide additional commentary, or rationale, on why the requirement or environmental assumption exists or why a particular value or range is specified. This practice discusses where and how rationale should be provided in a requirements specification.
2.11.1	Provide rationale during all phases of requirements development to explain why a requirement exists or why specific values are specified.
2.11.2	Do not use rationale to specify what the system will do, i.e., as an alternate source of requirements. If the rationale is essential to the required system behavior, it must be stated as a requirement.
2.11.3	Provide rationale for every requirement that does not have an obvious reason for why the requirement exists.
2.11.4	Provide rationale for every environmental assumption that does not have an obvious reason why the assumption is included.
2.11.5	Provide rationale for every value or range in a requirement or assumption explaining why that particular value or range was specified.
2.11.6	Keep the rationale short and relevant to the statement being explained. Summarize the relevance of long documents and cite the document in the rationale.
2.11.7	Capture rationale by the original author of the statement being explained as soon as possible.

[20] If the physical architecture implementing the system introduces additional latencies (for example, if a bus delay is introduced between two subsystems), these latencies need to included in this check.

Requirements document what a system will do. Design documents, how the system will do it. Rationale documents why a requirement exists or why it is written the way it is. Rationale should be included whenever there is something in the requirement that may not be obvious to the reader or that might help the reader to understand why the requirement exists.

2.11.1 Provide Rationale to Explain why a Requirement Exists.

Rationale should be added during all phases of requirements definition. Rationale can consist of free text, pointers to trade studies, other documents, papers, or book citations. The system goals (section 2.1.3) are often a useful source of rationale.

Hooks and Farry claim that providing rationale is the single most effective way of reducing the cost and improving the quality of requirements [22]. Rationale can reduce the amount of time required to understand a requirement by providing background information on why the requirement exists. Since most requirements specifications are read multiple times, investing the effort to provide the rationale should save time and money over the long run. Rationale can also help the readers to avoid making incorrect assumptions, the cost of which grows rapidly as development proceeds towards implementation. Rationale can also decrease the cost of maintenance by documenting why certain choices or values were selected. It can also make the development of variations of the same product cheaper by providing insight into the impact of changing the requirements and assumptions. Finally, rationale can decrease the cost of educating new employees by providing them with background information about why the system behaves the way it does.

Providing rationale can also improve the quality and reduce the cost of creating the requirements. Coming up with the rationale for a bad requirement or assumption can be difficult. Forcing the specifier to think about why the requirement is necessary or why the assumption is being made will often improve the quality of the requirement. It may even lead to the elimination of the requirement as the writer realizes it is not necessary or actually an implementation detail. This eliminates additional constraint on the developers and the cost of verifying an unnecessary requirement. Rationale also makes it easier to keep the requirements focused on what the system will do by providing a place for background information.

Recommended Practice 2.11.1: Provide rationale during all phases of requirements development to explain why a requirement exists or why specific values are specified.

2.11.2 Avoid Specifying Requirements in the Rationale.

At the same time, rationale should not document what the system must do, i.e., rationale should not include requirements or be used as an excuse for writing poor requirements. Rationale is not contractually binding and does not need to be implemented. If anything included in the rationale is essential to the required system behavior, it must be specified as a requirement.

Recommended Practice 2.11.2: Do not use rationale to specify what the system will do, i.e., as an alternate source of requirements. If the rationale is essential to the required system behavior, it must be stated as a requirement.

2.11.3 Provide Rationale When the Reason a Requirement is not Obvious.

A few examples of use of rationale are presented below. As mentioned above, rationale should be provided whenever the reason a requirement exists is not obvious. A good example of this can be found in the Isolette Thermostat example.

- REQ-MHS-4 If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

  Rationale: When the Isolette is warming towards the Upper Desired Temperature, the Heat Source should be left on until the Upper Desired Temperature is reached. In a similar fashion, if the Isolette is cooling towards the Lower Desired Temperature, the Heat Source should be left off until the Lower Desired Temperature is reached.

It is not immediately obvious why the Heat Source should be left unchanged in this range. However, the rationale makes the reasoning behind it much clearer. Hooks and Farry recommend capturing the rationale for every requirement [22] to improve the quality of all requirements. At a minimum, rationale should be provided for every requirement for which the reason why the requirement exists is not obvious.

Recommended Practice 2.11.3: Provide rationale for every requirement that does not have an obvious reason for why the requirement exists.

2.11.4 Provide Rationale for Environmental Assumptions.

In similar fashion, rationale should also be provided for every environmental assumption upon which the system depends. For example, one of the environmental assumptions for the Isolette Thermostat states:

- EA-IS-1: When the Heat Source is turned on and the Isolette is properly shut, the Current Temperature will increase at a rate of no more than 1°F per minute.

  Rationale: If the Current Temperature can increase at a rate of more than 1°F per minute, the Thermostat may not be able to turn the Heat Source off quickly enough to maintain the Desired Temperature Range unless the allowed latency specified for the Heat Control is reduced.

It is not immediately obvious from the assumption itself why it was included. As it turns out, the Thermostat turns the Heat Control for the Heat Source on and off, and the allowed latency specified on that controlled variable is calculated based on this assumption. The rationale helps to make this clear.

Recommended Practice 2.11.4: Provide rationale for every environmental assumption that does not have an obvious reason why the assumption is included.

2.11.5 Provide Rationale for Values and Ranges.

Rationale should also be captured whenever a number or range is entered in the specification. This helps the developer to understand why a particular value is or is not important. This can prove invaluable during development and maintenance. For example, in the Isolette Thermostat, the rationale for one of the environmental assumptions states:

- EA-OI-3 The Lower Alarm Temperature will always be ≥93°F.

  Rationale: Exposure to temperatures less than 93°F will result in hypothermia, which can lead to death within a few minutes for severely ill preterm infants.

The rationale makes it obvious why this number should not be changed to 90°F, even if this might allow use of a more commonly available and less expensive operator interface.

Recommended Practice 2.11.5: Provide rationale for every value or range in a requirement or assumption explaining why that particular value or range was specified.

2.11.6 Keep Rationale Short and Relevant.

At the same time, the best rationale is short and to the point. Rather than copying a trade study justifying a particular requirement, summarize the relationship of the trade study to the requirement and cite the trade study in the rationale.

Recommended Practice 2.11.6: Keep the rationale short and relevant to the statement being explained. Summarize the relevance of long documents and cite the document in the rationale.

2.11.7 Capture Rationale as Soon as Possible.

Rationale should be collected along with the development of the requirement or assumption it is explaining. This ensures that the justification is captured by the author while he or she is thinking about it. It also helps to ensure that there is time for others to review and question the rationale. In the example above, relating the latency on the Heat Control to the maximum rise in temperature in the Isolette, it is much simpler to record the rationale when the latency was being computed than to try to re-engineer the reasoning later.

Recommended Practice 2.11.7: Capture rationale by the original author of the statement being explained as soon as possible.

3. SUMMARY.

The management of requirements is one of the most important activities in the development of digital systems. In reference 46, Fred Brooks states the problem succinctly:

> "The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements...No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later."

While numerous methodologies for REM have been developed over the years, the results of an industry survey, described in reference 1, indicate the best of these practices are rarely being used, if at all, and digital system developers have many questions on how to effectively collect, document, and organize requirements.

This Handbook attempts to address this situation by bringing together the best ideas from several approaches, organizing them into a coherent whole, and illustrating them with concrete examples that make their benefits clear. It describes 11 main-level recommended practices that allow a developer to progress from informal approaches early in requirements definition to more rigorous methods as the requirements near completion. These practices are targeted to the domain of real-time, embedded systems and specifically to the avionics industry. Due to the rapidly growing importance of software in these systems, emphasis on the concepts eases the transition from system to software requirements.

While the recommended practices are presented in roughly the order they would be performed, there is no requirement that this order should be strictly adhered to. As with most processes, significant iteration between the different activities is to be expected as the requirements are refined.

Each main-level recommended practice is explained in the Handbook by selecting a particular approach to its implementation and illustrating that approach with a running example. While the examples and the sublevel recommended practices might appear to suggest a particular style and format, their real purpose is to illustrate and clarify the main-level practices and their benefits. It is expected that most organizations wanting to incorporate any of the recommended practices will want to modify them, perhaps significantly, for use in their environment.

To use this Handbook, an organization should identify which main-level practices are not already being used effectively within their organization, then determine how, and in what order, they wish to incorporate them. If the detailed approach described in the Handbook meets their needs (e.g., use cases for defining the operational concepts), they may want to implement the practice as described in this Handbook. If more detail or further tailoring is needed, the references cited for that practice are an excellent source of additional information. It should be noted that several of the cited approaches are already tailored in this Handbook to complement the other recommended practices.

As illustrated in this Handbook, a good set of requirements consists of much more than just a list of shall statements and is not easy to produce. However, investing the time and effort at the start of a project to produce good requirements was shown to ultimately reduce costs while improving the quality of the final product [22-24 and 47]. Incorporating the main-level practices recommended in this Handbook will help to provide the structure needed to produce requirements that are complete, consistent, clear, maintainable, and well organized.

4. REFERENCES.

1. Lempia, D. and Miller, S., "Requirements Engineering Management Findings Report," FAA report DOT/FAA/AR-08/34, May 2009.

2. Parnas, D. and Madey, J., "Functional Documentation for Computer Systems Engineering (Version 2)," Technical Report CRL 237, McMaster University, Hamilton, Ontario, September 1991.

3. Van Schouwen, A., "The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems," Technical Report 90-276, Queens University, Hamilton, Ontario, 1990.

4. Faulk, S., Brackett, J., Ward, P., and Kirby, J., Jr., "The CoRE Method for Real-Time Requirements," *IEEE Software*, Vol. 9, No. 5, September 1992, pp. 22-33.

5. Faulk, S., Finneran, L., Kirby, J., and Moini, A., "Consortium Requirements Engineering Guidebook," Technical Report SPC-92060-CMS, Software Productivity Consortium, 2214 Rock Hill Road, Herndon, Virginia, December 1993.

6. Faulk, S., Finneran, L., Kirby, J., Shah, S., and Sutton, J., "Experience Applying the CoRE Method to the Lockheed C-130J Software Requirements," *Proceedings of the Ninth Annual Conference on Computer Assurance*, Gaithersburg, Maryland, June 1994, pp. 3-8.

7. Heitmeyer, C., Kirby, J., and Labaw, B., "Automated Consistency Checking of Requirements Specification," *ACM Transactions on Software Engineering and Methodology* (TOSEM), Vol. 5, No. 3, July 1996, pp. 231-261.

8. Leveson, N., Heimdahl, M., Hildreth, H., and Reese, J., "Requirements Specifications for Process-Control Systems," *IEEE Transactions on Software Engineering*, Vol. 20, No. 9, September 1994, pp. 684-707.

9. Leveson, N., Heimdahl, M., Hildreth, H., and Reese, J., "TCAS II Collision Avoidance System (CAS) System Requirements Specification Change 6.00," Federal Aviation Administration, U.S. Department of Transportation, March 1993.

10. Thompson, J., Heimdahl, M., and Miller, S., "Specification-Based Prototyping for Embedded Systems," *Proceedings Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering*, LNCS 1687, September 1999, pp. 163-179.

11. Leveson, N., Heimdahl, M., and Reese, J., "Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future," *Proceedings Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering,* LNCS 1687, September 1999, pp. 127-145.

12. Miller, S., Tribble, A., Whalen, M., and Heimdahl, M., "Providing the Shalls," *International Journal on Software Tools for Technology Transfer (STTT)*, February 2006.

13. Howard, J. and Anderson, P., "The Safety Risk of Requirements Incompleteness," *Proceedings of the 20th International System Safety Conference* (ISSC 2002), Denver, Colorado, August 2002.

14. Lee, G., Howard, J., and Anderson, P., "Safety-Critical Requirements Specification Using SpecTRM," *Proceedings of the 2nd Meeting of the US Software System Safety Working Group*, February 2002.

15. Leveson, N., Reese, J., and Heimdahl, M., "SpecTRM: A CAD System for Digital Automation," *Proceedings of 17th Digital Avionics System Conference (DASC98)*, Seattle, Washington, November 1998.

16. Leveson, N., "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Transactions on Software Engineering*, Vol. 26, No. 1, January 2000, pp. 15-35.

17. Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, Boston, Massachusetts, 2001.

18. Alexander, I. and Zink, T., "An Introduction to Systems Engineering With Use Cases," *IEEE Computer and Control Engineering*, December 2002.

19. Leffingwell, D. and Widrig, D., *Managing Software Requirements*, Addison-Wesley, Reading, Massachusetts, 2003.

20. Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, Addison Wesley, Reading, Massachusetts, 1999.

21. Fowler, M., *UML Distilled*, *A Brief Guide to the Standard Object Modeling Language*, Third Edition, Addison Wesley, Reading, Massachusetts, September 2003.

22. Hooks, I. and Farry, K., "Customer Centered Products: Creating Successful Products Through Smart Requirements Management," AMACOM American Management Association, New York, New York, 2001.

23. Davis, A., *Software Requirements (Revised): Object, Functions, and States*, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

24. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

25. "IEEE Guide for Developing System Requirements Specifications," IEEE Std 1233, The Institute of Electrical and Electronics Engineers, New York, New York, December 1998.

26. "IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998, The Institute of Electrical and Electronics Engineers, New York, New York, June 1998.

27. "Software Considerations in Airborne Systems and Equipment Certification," DO-178B, RTCA, Washington, DC, December 1, 1992.

28. "Final Report for Clarification of DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," DO-248B, RTCA, Washington, DC, October 12, 2001.

29. "Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance," DO-278, March 5, 2002.

30. "Certification Considerations for Highly-Integrated or Complex Aircraft Systems," ARP 4754, SAE International, November 1996.

31. "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," ARP 4761, SAE International, December 1996.

32. Hughes, D. and Dornheim, M., "Automated Cockpits Special Report, Parts I & II," *Aviation Week & Space Technology*, January 30-February 6, 1995.

33. Billings, C., *Aviation Automation: The Search for a Human Centered Approach*, Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 1997.

34. Commercial Aviation Safety Team, "Final Report of the Loss of Control JSAT: Results and Analysis," P. Russell and J. Pardee, Co-chairs, December 15, 2000.

35. Leveson, N., et al., "Analyzing Software Specifications for Mode Confusion Potential," *Proceedings of a Workshop on Human Error and System Development*, Glasgow, Scotland, March 1997, pp. 132-146.

36. Leveson N., "Designing Automation to Reduce Operator Errors," *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, October 1997.

37. Sarter, N., Woods, D., and Billings, C., "Automaton Surprises," *Handbook of Human Factors/Ergonomics*, 2nd ed., G. Salvendy, ed., John Wiley & Sons, New York, New York, 1997.

38. Woodson, W., Tilman, P., and Tilman, B., *Human Factors Design Handbook, Second Edition*, McGraw-Hill Companies, 1992.

39. Boff, K. and Lincoln, J., *Engineering Data Compendium: Human Perception and Performance*, Wright-Patterson Air Force Base, Harry G. Armstrong Aerospace Medical Research Laboratory, Ohio, 1998.

40. Coplien, J., Hoffman, D., and Weiss, D., "Commonality and Variability in Software Engineering," *IEEE Software*, Vol. 20, No. 6, November 1998.

41. De Marco, T., "*Structured Analysis and System Specification*," Yourdon Press, New York, 1979.

42. Harel, D., "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, Vol. 8, 1987, pp. 231.

43. Hoffman, D. and Weiss, D., "*Software Fundamentals: Collected Papers by David L. Parnas*," Addison-Wesley Professional, Boston, Massachusetts, 2001.

44. Miller, S. and Tribble, A., "Extending the Four-Variable Model to Bridge the System-Software Gap, 20th Digital Avionics Systems Conference (DASC01), Daytona Beach, Florida, October 14-18, 2001.

45. Stevens, R., Jackson, B., and Arnold, S., "*Systems Engineering: Coping With Complexity,*" Prentice-Hall, London, 1998.

46. Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, April 1987.

47. Boehm, B., "*Software Engineering Economics*," Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

APPENDIX A—ISOLETTE THERMOSTAT EXAMPLE

This appendix contains an example requirements specification for the Isolette Thermostat discussed in section 3. The presented format is one example of how the best practices of section 2 could be realized. There are many other formats that would be equally effective.

A.1. SYSTEM OVERVIEW.

The system being specified is the Thermostat of an Isolette.[1] An Isolette is an incubator for an Infant that provides controlled temperature, humidity, and oxygen (if necessary). Isolettes are used extensively in Neonatal Intensive Care Units for the care of premature infants.

The purpose of the Isolette Thermostat is to maintain the air temperature of an Isolette within a desired range. It senses the Current Temperature of the Isolette and turns the Heat Source on and off to warm the air as needed. If the temperature falls too far below or rises too far above the Desired Temperature Range, it activates an alarm to alert the Nurse. The system allows the Nurse to set the Desired Temperature Range and to set the Alarm Temperature Range outside the Desired Temperature Range of which the alarm should be activated.

A.1.1 SYSTEM CONTEXT.

The operational context of the Isolette Thermostat is shown in figure A-1.

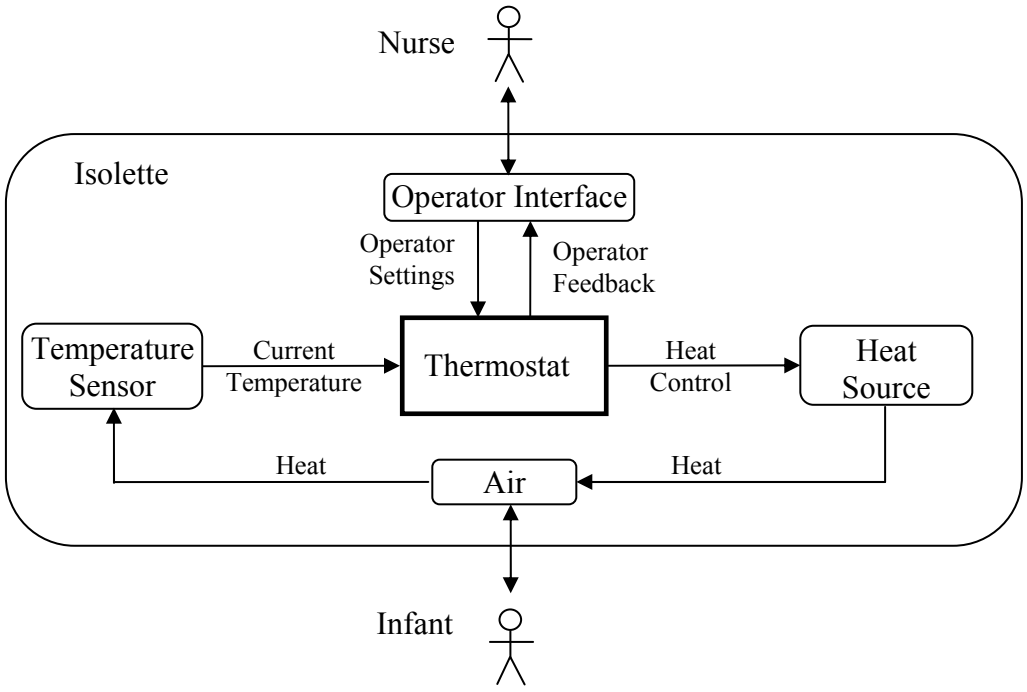


Figure A-1. Context Diagram for the Isolette Thermostat

[1] To simplify this example, the Operator Interface is treated as an external entity outside of the Thermostat.

The Thermostat interacts directly with three entities that are part of the Isolette:

- The Temperature Sensor provides the Current Temperature of the air in the Isolette to the Thermostat.
- The Heat Source heats the Air in the Isolette. It is turned on and off by the Heat Control.
- The Operator Interface provides the Operator Settings for the Thermostat and receives Operator Feedback from the Thermostat.

The Thermostat also interacts indirectly with other entities outside of the Isolette:

- The Nurse who uses the Operator Interface to enter the Operator Settings and view the Operator Feedback.
- The Air in the Isolette.
- The Infant that is placed in the Isolette and is warmed by the Air.

A.1.2 SYSTEM GOALS.

The high-level goals (G) of the system are:

- G1—The Infant should be kept at a safe and comfortable temperature.
- G2—The Nurse should be warned if the Infant becomes too hot or too cold.
- G3—The cost of manufacturing the Thermostat should be as low as possible.

A.2 OPERATION/AL CONCEPTS.

The following use and exception cases describe how the operators interact with the Isolette and the Thermostat. A summary of the use and exception cases is provided in table A-1.

Table A-1. Summary of Isolette Thermostat Use and Exception Cases

ID	Primary Actors	Title and Description
A.2.1	Nurse	Normal Operation of Isolette Describes the normal operation of the Isolette by the Nurse
A.2.2	Nurse	Configure the Isolette Describes how the Nurse configures the Isolette and Thermostat for the Infant
A.2.3	Thermostat	Maintain Desired Temperature Describes how the Thermostat turns the Heat Source on and off to maintain the Current Temperature in the Isolette within the Desired Temperature Range

Table A-1. Summary of Isolette Thermostat Use and Exception Cases (Continued)

ID	Primary Actors	Title and Description
A.2.4	Thermostat	Failure to Maintain Safe Temperature Describes how the Thermostat and the Nurse respond when the Isolette is unable to maintain the Current Temperature within the Alarm Temperature Range
A.2.5	Thermostat	Respond to Thermostat Failure Describes how the Thermostat and the Nurse respond when the Thermostat detects an internal failure
A.2.6	Nurse	Failure to Maintain Desired Temperature Describes how the Nurse deals with an Isolette that cannot keep the Current Temperature within the Desired Temperature Range but can keep the Current Temperature within the Alarm Temperature Range

The actors and their primary goals are shown in table A-2.

Table A-2. Isolette Thermostat Primary Actors and Goals

Actor	Primary Goals of the Actor
Nurse	Provide the Infant with proper nursing care, including keeping the Infant warm
Infant	Be comfortable and healthy
Isolette	Hold the Infant and maintain the Current Temperature within the Desired Temperature Range
Thermostat	Maintain Current Temperature in the Isolette within the Desired Temperature Range

A.2.1 USE CASE: NORMAL OPERATION OF ISOLETTE.

This use case describes the normal operation of the Isolette by the Nurse.

- Related System Goals: G1 and G2
- Primary Actor: Nurse
- Precondition:
  - Infant is ready to be placed in the Isolette
  - Isolette and Thermostat are turned off

- Postcondition:

  - Infant is removed from the Isolette
  - Isolette and Thermostat are turned off

- Main Success Scenario:

  1. Nurse turns on the Isolette
  2. Isolette turns on the Thermostat
  3. Thermostat initializes and enters its normal mode of operation (exception case 1) (A.2.5, A.5.1.2 and A.5.2.2)
  4. Nurse configures the Isolette for the needs of the Infant (A.2.2)
  5. Nurse waits until the Current Temperature is within the Desired Temperature Range (A.2.6 and A.5.1.1)
  6. Nurse places the Infant in the Isolette
  7. Isolette maintains Desired Temperature (A.2.3)
  8. Nurse confirms that the Current Temperature is in the Desired Temperature Range during rounds (A.2.6 and A.5.1.1)
  9. Nurse removes Infant
  10. Nurse turns off the Isolette
  11. Isolette turns off the Thermostat

- Exception Case 1:

  1. Alarm is activated because Current Temperature is outside the Alarm Temperature Range (A.5.2.3)
  2. Nurse ignores the Alarm[2]
  3. Continue with Main Success Scenario, step 4.

[2] In the interest of simplicity, the functionality to turn the Alarm off is not specified. As an exercise, the reader might want to consider what changes would be necessary to add this capability to the example.

A.2.2 USE CASE: CONFIGURE THE ISOLETTE.

This use case describes how the Nurse configures the Isolette and Thermostat for the Infant.

- Related System Goals: G1 and G2
- Primary Actor: Nurse
- Precondition: The Isolette and Thermostat are turned on
- Postcondition:
  - The Desired Temperature Range is set for the needs of the Infant
  - The Alarm Temperature Range is set for the needs of the Infant
  - The Current Temperature in the Isolette is in the Desired Temperature Range
- Main Success Scenario:
  1. Nurse sets the Alarm Temperature Range for the Infant (A.5.2.1)
  2. Nurse sets the Desired Temperature Range for the Infant (A.5.1.1)
  3. Thermostat maintains Desired Temperature Range (A.2.3)

A.2.3 USE CASE: MAINTAIN DESIRED TEMPERATURE.

This use case describes how the Thermostat turns the Heat Source on and off to maintain the Current Temperature in the Isolette within the Desired Temperature Range.

- Related System Goals: G1
- Primary Actor: Thermostat
- Precondition: Isolette and Thermostat are turned on
- Postcondition:
  - Isolette and Thermostat are turned on
  - Current Temperature is in the Desired Temperature Range
- Main Success Scenario:
  1. Current Temperature falls below the Desired Temperature Range
  2. Thermostat turns the Heat Source on to warm up the Isolette (A.5.1.3)
  3. Current Temperature rises above the Desired Temperature Range
  4. Thermostat turns the Heat Source off to cool the Isolette (A.5.1.3)
  5. Repeat steps 1 through 4

A.2.4 EXCEPTION CASE: FAILURE TO MAINTAIN SAFE TEMPERATURE.

This exception case describes how the Thermostat and Nurse respond when the Isolette is unable to maintain Current Temperature within the Alarm Temperature Range.

- Related System Goals: G2

- Primary Actor: Thermostat

- Precondition:

  - The Isolette and Thermostat are turned on
  - The Current Temperature is within the Alarm Temperature Range
  - The Alarm is off

- Postcondition:

  - The Isolette and Thermostat are turned on
  - The Current Temperature is within the Desired Temperature Range
  - The Alarm is off

- Main Success Scenario:

  1. Current Temperature falls below or rises above the Alarm Temperature Range
  2. Thermostat activates the Alarm (A.5.2.3)
  3. Nurse responds to the Alarm and sees that the Display Temperature is in the Alarm Temperature Range (A.5.1.1)
  4. Nurse removes Infant from the Isolette
  5. Nurse corrects the problem, e.g., closing an open door (alternate course 1)
  6. Nurse waits until the Display Temperature is within the Desired Temperature Range (A.2.6 and A.5.1.1)
  7. Nurse places Infant back in the Isolette

- Alternate Course 1:

  1. Nurse is unable to correct the problem
  2. Nurse obtains another Isolette
  3. Nurse starts normal operation of the new Isolette (A.2.1)

A.2.5 EXCEPTION CASE: RESPOND TO THERMOSTAT FAILURE.

This exception case describes how the Thermostat and the Nurse respond when the Thermostat detects an internal failure.

- Related System Goals: G2

- Primary Actor: Thermostat

- Precondition:

  - The Isolette and Thermostat are turned on
  - The Thermostat status is on
  - The Alarm is off

- Postcondition:

  - The Isolette and Thermostat are turned on
  - The Current Temperature is in the Desired Temperature Range
  - The Alarm is off

- Main Success Scenario:

  1. Thermostat detects an internal failure (A.5.1.4 and A.5.2.4)
  2. Thermostat enters the FAILED mode (A.5.1.2 and A.5.2.2)
  3. Thermostat sets its Displayed Status to failed (A.5.1.1 and A.5.2.1)
  4. Thermostat activates the Alarm
  5. Nurse responds to the Alarm and sees that the Thermostat is failed
  6. Nurse removes Infant from the Isolette
  7. Nurse obtains another Isolette
  8. Nurse starts normal operation of the new Isolette (A.2.1)

A.2.6 EXCEPTION CASE: FAILURE TO MAINTAIN DESIRED TEMPERATURE.

This exception case describes how the Nurse handles an Isolette that cannot keep the Current Temperature within the Desired Temperature Range, but can keep the Current Temperature within the Alarm Temperature Range.

- Related System Goals: G1
- Primary Actor: Nurse

- Precondition:

  - The Isolette and Thermostat are turned on
  - The Current Temperature is not within the Desired Temperature Range
  - The Current Temperature is within the Alarm Temperature Range

- Postcondition:

  - The Isolette and Thermostat are turned on
  - The Current Temperature is in the Desired Temperature Range

- Main Success Scenario:

    1. Nurse attempts to correct the problem, e.g., closing an open door

    2. Nurse waits until the Current Temperature of the Isolette is within the Desired Temperature Range (alternate course 1) (A.5.1.1)

    3. Return to calling scenario

- Alternate Course 1:

    1. Display Temperature fails to enter the Desired Temperature Range (A.5.1.1)
    2. Nurse removes Infant from the Isolette
    3. Nurse obtains another Isolette
    4. Nurse starts normal operation of the new Isolette (A.2.1)
    5. Return to calling scenario

A.3 EXTERNAL ENTITIES.

The following sections describe the external entities with which the Thermostat directly interacts: the Temperature Sensor, the Operator Interface, and the Heat Source. The monitored and controlled variables associated with each entity are listed, along with any environmental assumptions made about the entity. An Isolette external entity is also defined to specify environmental assumptions that span more than one external entity.

A.3.1 ISOLETTE.

An Isolette is an incubator for an Infant that provides controlled temperature, humidity, and oxygen (if necessary). It encompasses the Thermostat, the Temperature Sensor, the Operator Interface, and the Heat Source. The following environmental assumptions are made by the Thermostat about the Isolette.

- EA-IS-1: When the Heat Source is turned on and the Isolette is properly shut, the Current Temperature will increase at a rate of no more than 1°F per minute.

    Rationale: If the Current Temperature can increase at a rate of more than 1°F per minute, the Thermostat may not be able to turn the Heat Source off quickly enough to maintain the Desired Temperature Range unless the allowed latency specified for the Heat Control is reduced.

- EA-IS-2: When the Heat Source is turned off and the Isolette is properly shut, the Current Temperature will decrease at a rate of no more than 1°F per minute.

    Rationale**:** If the Current Temperature can decrease at a rate of more than 1°F per minute, the Thermostat may not be able to turn the Heat Source on quickly enough to maintain the Desired Temperature Range unless the allowed latency specified for the Heat Control is reduced.

A.3.2 TEMPERATURE SENSOR.

The Temperature Sensor provides the Current Temperature of the Air in the Isolette to the Thermostat. The monitored variables are shown in table A-3.

Table A-3. Thermostat Monitored Variables for Temperature Sensor

Name	Type	Range	Units	Physical Interpretation
Current Temperature	Real	[68.0..105.0]	°F	Current air temperature inside Isolette
	Status	●Invalid, Valid		

● denotes initial value

The following environmental assumptions are made:

- EA-TS-1: The Current Temperature will be provided to the Thermostat in degrees Fahrenheit

  Rationale: Consistency with environmental-assumption Operator Interface EA-OI-1

- EA-TS-2: The Current Temperature will be sensed to an accuracy of ±0.1°F.

  Rationale: An accuracy of 0.1°F is necessary to ensure the Thermostat can turn the Heat Source on and off quickly enough to maintain the Desired Temperature Range.

- EA-TS-3: The Current Temperature will cover the range of at least 68.0° to 105.0°F.

  Rationale: This is the specified range of operation of the Isolette. The lower end of this range is useful for monitoring an Isolette that is warming to the Desired Temperature Range. The upper end is greater than the Upper Alarm Temperature to ensure that the Current Temperature will be sensed across the entire Alarm Temperature Range.

A.3.3 HEAT SOURCE.

The Heat Source heats the Air in the Isolette. It is turned on and off by changing the value of the Heat Control controlled variable. The controlled variables are shown in table A-4. No environmental assumptions are made.

Table A-4. Thermostat Controlled Variables for Heat Source

Name	Type	Range	Units	Physical Interpretation
Heat Control	Enumerated	Off, On		Command to turn Heat Source on and off

A.3.4 OPERATOR INTERFACE.

The Operator Interface provides the Operator Settings for the Thermostat and receives Operator Feedback from the Thermostat. The environmental assumptions associated with the Operator Interface are quite strong, which simplifies the manage Operator Interface Function. If these assumptions were not satisfied by the Operator Interface external entity, the Manage Operator Interface Function would need to be strengthened to ensure consistent inputs to the Thermostat. The monitored and controlled variables are shown in tables A-5 and A-6, respectively.

Table A-5. Thermostat Monitored Variables for Operator Interface

Name	Type	Range	Units	Physical Interpretation
Operator Settings				Thermostat settings provided by operator
Desired Temperature Range				Desired range of Isolette temperature
Lower Desired Temperature	Integer	[97..99]	°F	Lower value of Desired Temperature Range
	Status	●Invalid, Valid		
Upper Desired Temperature	Integer	[98..100]	°F	Upper value of Desired Temperature Range
	Status	●Invalid, Valid		
Alarm Temperature Range				Activate Alarm when outside of this range
Lower Alarm Temperature	Integer	[93..98]	°F	Lower value of Alarm Temperature Range
	Status	●Invalid, Valid		
Upper Alarm Temperature	Integer	[99..103]	°F	Upper value of Alarm Temperature Range
	Status	●Invalid, Valid		

● denotes initial value

Table A-6. Thermostat Controlled Variables for Operator Interface

Name	Type	Range	Units	Physical Interpretation
Operator Feedback				Information provided back to the operator
Regulator Status	Enumerated	Init, On, Failed		Status of the Thermostat Regulator Function
Monitor Status	Enumerated	Init, On, Failed		Status of the Thermostat Monitor Function
Display Temperature	Integer	[68..105]	°F	Displayed temperature of Isolette
Alarm	Enumerated	Off, On		Command to turn Alarm on or off

The following environmental assumptions are made:

- EA-OI-1: All temperatures will be entered and displayed in degrees Fahrenheit.

  Rationale: Minimize the complexity of this example. An actual system would probably support Celsius or perhaps both Fahrenheit and Celsius

- EA4-OI-2: All temperatures will be set and displayed by the operators in increments of 1°F.

  Rationale: Marketing studies have shown that customers prefer to set temperatures in 1 degree increments. A resolution 1°F is sufficient to be consistent with the functional and performance requirements specified in the rest of the document.

- EA-OI-3: The Lower Alarm Temperature will always be ≥93°F.

  Rationale: Exposure to temperatures less than 93°F will result in hypothermia, which can lead to death within a few minutes for severely ill preterm infants.

- EA-OI-4: The Lower Alarm Temperature will always be less than or equal to the Lower Desired Temperature of -1°F.

  Rationale: If the Lower Alarm Temperature is greater than or equal to the Lower Desired Temperature, the Alarm could be activated while the Current Temperature is still in the Desired Temperature Range.

- EA-OI-5: The Lower Desired Temperature will always be ≥97°F.

  Rationale: Exposing the Infant to temperatures lower than 97°F may result in excessive heat loss and drop in heart rate secondary to metabolic acidosis.

- EA-OI-6: The Lower Desired Temperature will always be less than or equal to the Upper Desired Temperature of -1°F.

  Rationale: If the Lower Desired Temperature is greater than or equal to the Upper Desired Temperature, it is unclear if the Heat Source should be on or off. This may result in excessive cycling of the Heat Source.

- EA-OI-7: The Upper Desired Temperature will always be ≤100°F.

  Rationale: Exposing the Infant to temperatures greater than 100°F may result in an incorrect diagnosis of fever resulting in aggressive evaluation (blood culture and lumbar puncture) and treatment for infection.

- EA-OI-8: The Upper Alarm Temperature will always be greater than or equal to the Upper Desired Temperature of 1°F.

  Rationale: If the Upper Alarm Temperature is less than or equal to the Upper Desired Temperature, the Alarm could be activated while the Current Temperature is still in the Desired Temperature Range.

- EA-OI-9: The Upper Alarm Temperature will always be ≤103°F.

Rationale: Exposure to temperatures greater than 103°F will result in hyperthermia, which can lead to cardiac arrhythmias and febrile seizures within a few minutes.

- EA-OI-9: The Display Temperature will cover the range of at least 68.0° to 105.0°F.

  Rationale: This is the specified range of operation of the Isolette. The lower end of this range is useful for monitoring an Isolette that is warming to the Desired Temperature Range. The upper end is set to be greater than the maximum Upper Alarm Temperature.

A.4 SAFETY REQUIREMENTS.

The following relevant hazards were identified through the safety assessment process:

- H1: Prolonged exposure of Infant to unsafe heat or cold
  Classification: catastrophic
  Probability: $<10^{-9}$ per hour of operation

To ensure that probability of hazard H1 is 10^{-9} per hour of operation, the following derived safety requirements are levied on the Isolette Thermostat:

- SR-1: The Isolette shall include an independent regulator function that maintains the Current Temperature inside the Isolette within the Desired Temperature Range.

  Rationale: The Desired Temperature Range will be set by the Nurse to the ideal range based on the Infant's weight and health. The regulator should maintain the Current Temperature within this range under normal operation.

  Allowed probability of failure: $<10^{-5}$ per hour

- SR-2: The Isolette shall include an independent monitor function that activates an Alarm within a maximum of 5 seconds whenever

  - the Current Temperature falls below or rises above the Alarm Temperature Range.
  - the Current Temperature or the Alarm Temperature Range is flagged as invalid.
  - an internal failure has been detected in the monitor function.

  Rationale: The Alarm Temperature Range will be set by the Nurse based on the Infant's weight and health. The Infant should be removed from the Isolette within 15 seconds after the Current Temperature falls below or rises above this range. With the normal monitoring provided by the Nurse, this can be accomplished within 10 seconds, leaving 5 seconds for the system to activate the Alarm. Activating the Alarm in less time is desirable.

  If the Current Temperature or the Alarm Temperature Range provided to the monitor function are flagged as invalid or if an internal failure is detected in the monitor function, the monitor function should not be trusted to perform correctly.

Allowed probability of failure: $<10^{-5}$ per hour.

A.5 THERMOSTAT SYSTEM FUNCTION.

The Thermostat performs two logically independent functions. The first regulates the Current Temperature in the Isolette so it is maintained within the Desired Temperature Range. The second monitors the Current Temperature in the Isolette and activates an Alarm if it falls below or rises above the Alarm Temperature Range.

The high-level requirements for the Thermostat Function are as follows:

- REQ-TH-1: The Thermostat shall set the value of the Heat Control.

  Rationale: A primary function of the Thermostat is to turn the Heat Control on and off to maintain the Current Temperature in the Isolette within the Desired Temperature Range, which is required by SR-1.

- REQ-TH-2: The Thermostat Function shall set the value of the Regulator Status.

  Rationale: SR-1 requires the Thermostat to provide an independent regulator function. The status of this function is provided to the Operator Interface by the Thermostat. The Operator Interface will use the Regulator Status and the Monitor Status to report the overall status of the Thermostat, which is required by SR-1.

- REQ-TH-3: The Thermostat shall set the value of the Display Temperature.

  Rationale: The Current Temperature is displayed on the Operator Interface to provide the operators with an additional means to confirm the Isolette is maintaining the temperature correctly. This value is provided by the Thermostat to the Operator Interface as the Display Temperature.

- REQ-TH-4: The Thermostat shall set the value of the Alarm Control.

  Rationale: A primary Thermostat Function is to activate the Alarm if the Isolette is unable to maintain the Current Temperature within the Alarm Temperature Range, which is required by SR-2.

- REQ-TH-5: The Thermostat shall set the value of the Monitor Status.

  Rationale: SR-2 requires the Thermostat to provide an independent monitor function. The status of this function must be provided to the Operator Interface, which will use it and the status of the regulator function to report the overall status of the thermostat.

The Thermostat Function is allocated into subfunctions as shown in figure A-2.

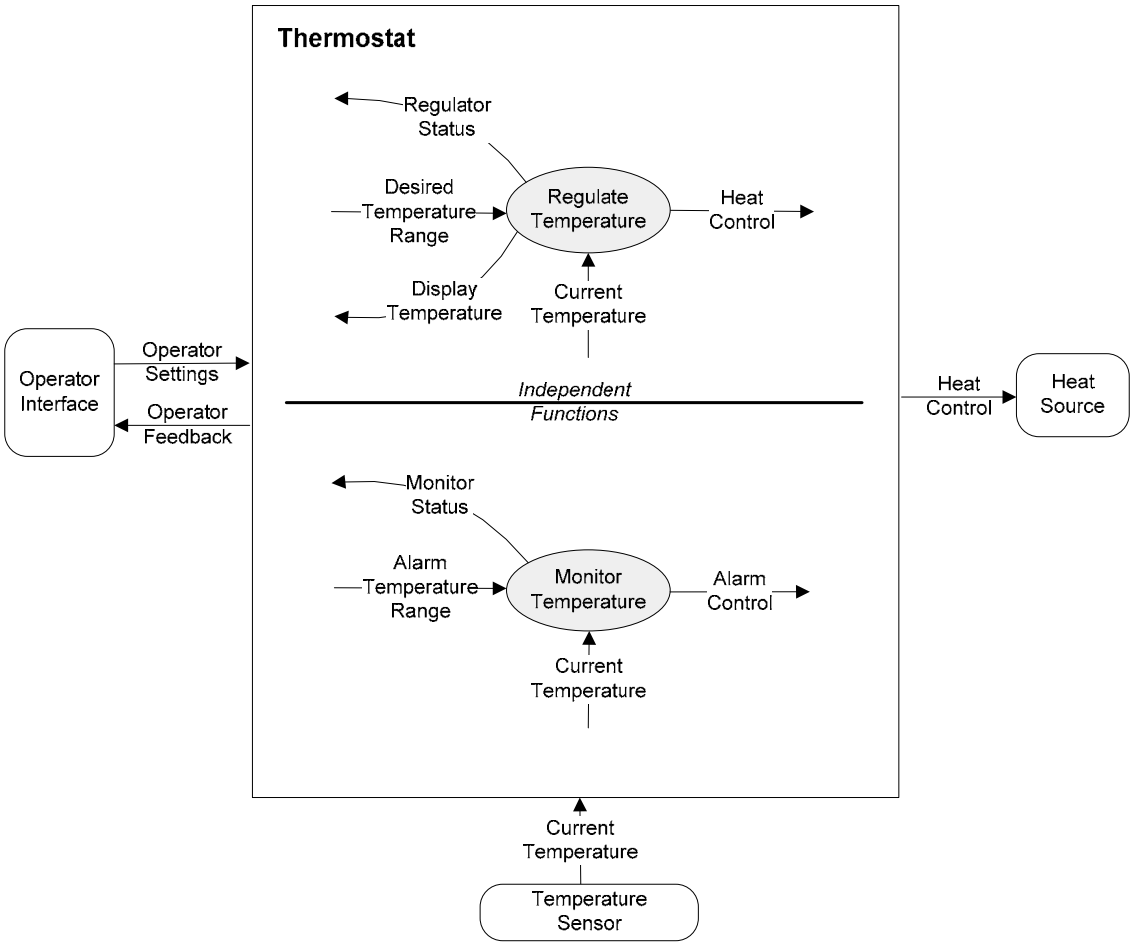


Figure A-2. Thermostat Dependency Diagram

A.5.1 REGULATE TEMPERATURE FUNCTION.

The Regulate Temperature Function compares the Current Temperature from the Temperature Sensor with the Desired Temperature Range provided by the Operator Interface and turns the Heat Source on or off to keep the Current Temperature within the Desired Temperature Range. It also provides the Display Temperature and the Regulator Status back to the Operator Interface.

The high-level requirements for the Regulate Temperature Function are as follows:

- REQ-RT-1: The Regulate Temperature Function shall set the value of the Heat Control.

  Rationale: The primary function of the Regulate Temperature Function is to turn the Heat Control on and off to maintain the Current Temperature in the Isolette within the Desired Temperature Range, as required by SR-1.

- REQ-RT-2: The Regulate Temperature Function shall set the value of the Regulator Status.

  Rationale: The status of the Regulate Temperature Function is provided to the Operator Interface so it can use the status of the Regulate Temperature and Monitor Temperature Functions to report the overall status of the Thermostat, as required by SR-1.

- REQ-RT-3: The Regulate Temperature Function shall set the value of the Display Temperature.

  Rationale: The Current Temperature of the Isolette is displayed on the Operator Interface to provide the operators with an additional means to confirm that the Isolette is maintaining the temperature correctly. This value is provided by the Regulate Temperature Function to the Operator Interface as the Display Temperature.

The Regulate Temperature Function is allocated into subfunctions in figure A-3.

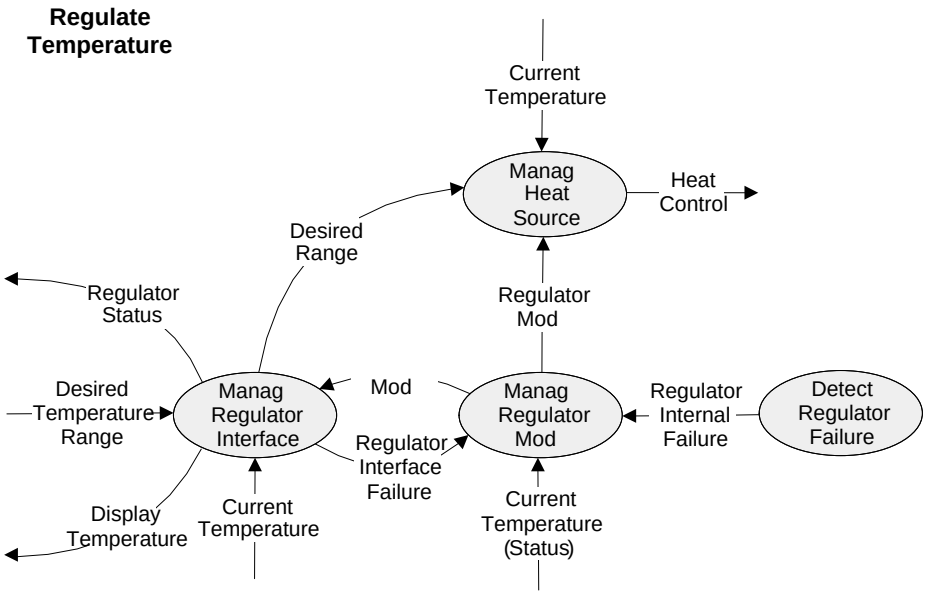


Figure A-3. Regulate Temperature Dependency Diagram

The internal variables for the Regulate Temperature Function are shown in table A-7.

Table A-7. The Regulate Temperature Internal Variables

Name	Type	Range	Units	Physical Interpretation
Desired Range				Desired range of Isolette temperature
Lower Desired Temp	Integer	[96..101]	°F	Lower value of desired range
Upper Desired Temp	Integer	[97..102]	°F	Upper value of desired range
Regulator Interface Failure	Boolean	False, True		Indicates an operator interface
Regulator Internal Failure	Boolean	False, True		Indicates an internal failure
Regulator Mode	Enumerated	Init		Initializing following power-up
		NORMAL		Normal mode of operation
		FAILED		Internal failure detected

A.5.1.1 Manage Regulator Interface Function.

The Manage Regulator Interface Function defines the interaction with the Operator Interface external entity. These include obtaining the Desired Range, reporting back the status of the Regulate Temperature Function, and reporting back the Display Temperature. The constants are shown in table A-8.

Table A-8. Manage Regulator Interface Function Constants

Name	Type	Value	Units	Physical Interpretation
Max Operator Response Time	Real	0.5	Sec	The time an operator will tolerate between an operator request or a change in the Thermostat state and the visible response
Rationale: A trade study has shown that this lag should be no more than 0.5 second.				

The requirements for the Regulator Status controlled variable are as follows:

- REQ-MRI-1: If the Regulator Mode is INIT, the Regulator Status shall be set to Init.
- REQ-MRI-2: If the Regulator Mode is NORMAL, the Regulator Status shall be set to On.
- REQ-MRI-3: If the Regulator Mode is FAILED, the Regulator Status shall be set to Failed.

  Latency: < Max Operator Response Time
  Tolerance: N/A

The requirements for the Display Temperature controlled variable are as follows:

- REQ-MRI-4: If the Regulator Mode is NORMAL, the Display Temperature shall be set to the value of the Current Temperature rounded to the nearest integer.

  Rationale: Displaying the rounded value of the Current Temperature provides the the most accurate display of the Current Temperature possible using an integer display. When combined with the accuracy of the Temperature Sensor (EA-TS-2), the Display Temperature should be within 0.6°F of the actual value.

- REQ-MRI-5: If the Regulator Mode is not NORMAL, the value of the Display Temperature is UNSPECIFIED.

  Rationale: In modes other than NORMAL, the value of Display Temperature is not meaningful and should not be used.

  Latency: < Max Operator Response Time
  Tolerance: ±0.6°F

The requirements for the Regulator Interface Failure internal variable are as follows:

- REQ-MRI-6: If the Status attribute of the Lower Desired Temperature or the Upper Desired Temperature is Invalid, the Regulator Interface Failure shall be set to True.

- REQ-MRI-7: If the Status attribute of the Lower Desired Temperature and the Upper Desired Temperature is Valid, the Regulator Interface Failure shall be set to False.

  Rationale: The Regulator Interface Failure internal variable indicates if any errors have occurred in sensing the Operator Interface monitored variables needed by the Regulate Temperature Function. Note that its initial value on power-up will always be True since the Status of the Lower Desired Temperature and the Upper Desired Temperature are initially Invalid.

The requirements for the Desired Range internal variable are as follows:

- REQ-MRI-8: If the Regulator Interface Failure is False, the Desired Range shall be set to the Desired Temperature Range.

- REQ-MRI-9: If the Regulator Interface Failure is True, the Desired Range is UNSPECIFIED.

  Rationale: The Desired Range is only meaningful when there is not a Regulator Interface Failure. If there is, its value should not be used, and it can be set to any value.

A.5.1.2 Manage Regulator Mode Function.

The Manage Regulator Mode Function determines the mode of the Regulate Temperature Function. The constants and definitions are shown in tables A-9 and A-10, respectively.

Table A-9. The Manage Regulator Mode Function Constants

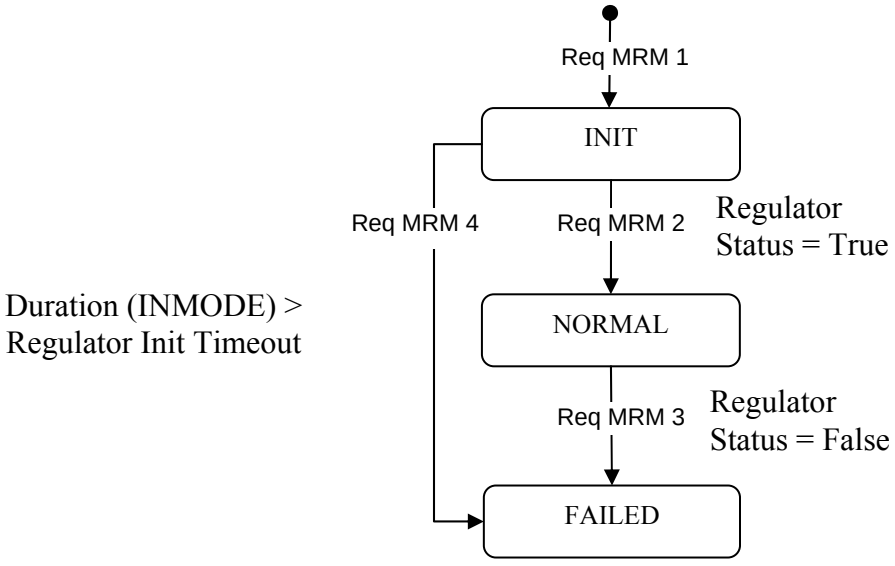
Name	Type	Value	Units	Physical Interpretation
Regulator Init Timeout	Real	1.0	Sec	The time allowed for initialization of the Regulate Temperature Function before declaring failure
Rationale: A trade study has shown that users become impatient if the Thermostat requires more than one second to initialize.				

Table A-10. The Manage Regulator Mode Function Definitions

Name	Type	Definition
Regulator Status	Boolean	NOT (Regulator Interface Failure OR Regulator Internal Failure) AND Current Temperature.Status = Valid

The requirements for the Regulator Mode internal variable are as follows:

- The modes and transitions of the Manage Regulator Mode Function are specified in the state transition diagram shown in figure A-4. Each transition is a separate requirement and is assigned a unique identifier (e.g., Req MRM 1). All transitions are assumed to occur in negligible time.



MRM = Manage Regulator Mode

Figure A-4. Regulate Temperature Mode Transition Diagram

Rationale: (Req MRM 3 and Req MRM 4) Once the regulator has failed, the only way for it to re-enter normal operation is for it to be powered off and on. This ensures that the operators are made aware of any transient failures that the regulator may be experiencing.

A.5.1.3 Manage Heat Source Function.

The Manage Heat Source Function turns the Heat Source on and off to maintain the Current Temperature of the Isolette within the Desired Temperature Range. The constants are shown in table A-11.

Table A-11. The Manage Heat Source Function Constants

Name	Type	Value	Units	Physical Interpretation
Allowed Heat Source Latency	Real	6.0	Sec	The maximum time by which the Heat Source must be turned on or off to ensure acceptable operation of the Isolette system
Rationale: Since a closed Isolette will warm or cool at a maximum rate of 1°F per minute (EA-IS1 and EA-IS2), turning the Heat Source on or off within 6 seconds ensures that the Current Temperature will not have changed by more than 0.1°F, the required accuracy and resolution of the Temperature Sensor (EA-TS2).				

The requirements for the Heat Control controlled variable are as follows:

- REQ-MHS-1: If the Regulator Mode is INIT, the Heat Control shall be set to Off.

  Rationale: A regulator that is initializing cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

- REQ-MHS-2: If the Regulator Mode is NORMAL and the Current Temperature is less than the Lower Desired Temperature, the Heat Control shall be set to On.

- REQ-MHS-3: If the Regulator Mode is NORMAL and the Current Temperature is greater than the Upper Desired Temperature, the Heat Control shall be set to Off.

- REQ-MHS-4: If the Regulator Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Desired Temperature and less than or equal to the Upper Desired Temperature, the value of the Heat Control shall not be changed.

  Rationale: When the Isolette is warming towards the Upper Desired Temperature, the Heat Source should be left on until the Upper Desired Temperature is reached. In a similar fashion, if the Isolette is cooling towards the Lower Desired Temperature, the Heat Source should be left off until the Lower Desired Temperature is reached.

- REQ-MHS-5: If the Regulator Mode is FAILED, the Heat Control shall be set to Off.

  Rationale: In failed mode, the regulator cannot regulate the Current Temperature of the Isolette and the Heat Control should be turned off.

  Latency: < Allowed Heat Source Latency
  Tolerance: N/A

A.5.1.4 Detect Regulator Failure Function.

The Detect Regulator Failure Function identifies internal failures, (e.g., a memory check failure) in the Regulate Temperature Function. It defines a single Boolean-valued internal variable, Regulator Internal Failure, which is set to True if an internal failure is detected.

The requirements for Regulator Internal Failure variable are implementation-specific and cannot be specified until an implementation platform is chosen.

A.5.2 Monitor Temperature Function.

The Monitor Temperature Function compares the Current Temperature from the Temperature Sensor with the Alarm Temperature Range provided by the Operator Interface and turns the Alarm Control on or off to Alert The Nurse if the Current Temperature falls below or rises above the safe range. It also provides the Monitor Status back to the Operator Interface.

The high-level requirements for the Monitor Temperature Function are as follows:

- REQ-MT-1: The Monitor Temperature Function shall set the value of the Alarm Control.

  Rationale: The primary function of the Monitor Temperature Function is to raise an alarm if the Isolette is unable to maintain the Current Temperature within the Alarm Temperature Range, as required by safety requirement SR-2.

- REQ-MT-2: The Monitor Temperature Function shall set the value of the Monitor Status.

  Rationale: Safety requirement SR-2 requires the Thermostat to provide an independent monitor function. The status of this function must be provided to the Operator Interface, which will use it and the status of the Regulate Temperature Function to report the overall status of the Thermostat, as required by safety requirement SR-2.

The Monitor Temperature Function is allocated into subfunctions as shown in figure A-5.

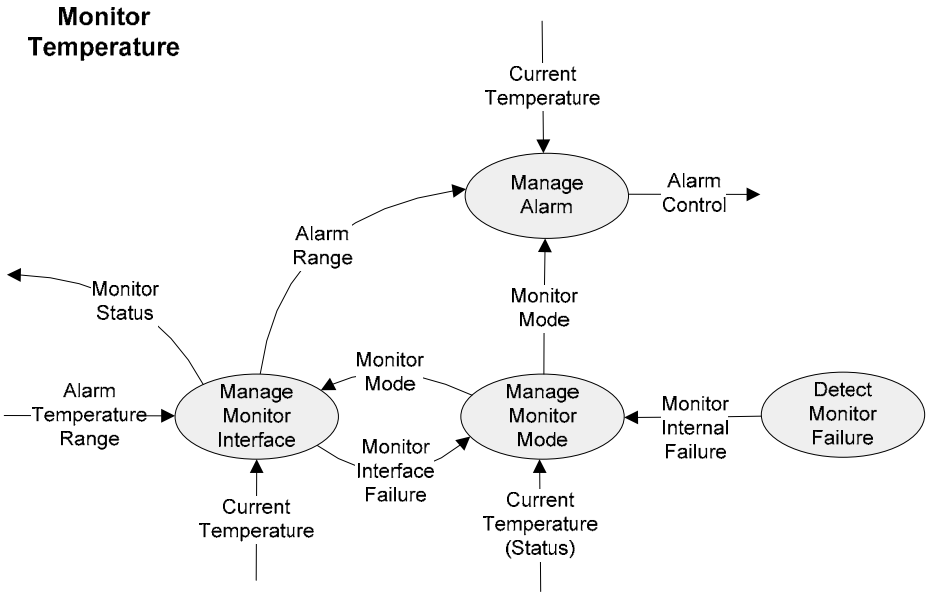


Figure A-5. Monitor Temperature Dependency Diagram

The Monitor Temperature internal variables are shown in table A-12.

Table A-12. Monitor Temperature Internal Variables

Name	Type	Range	Units	Physical Interpretation
Alarm Range				Safe range of Isolette temperature
Lower Alarm Temp	Integer	[96..101]	°F	Lower value of alarm range
Upper Alarm Temp	Integer	[97..102]	°F	Upper value of alarm range
Monitor Interface Failure	Boolean	False, True		Indicates an operator interface failure
Monitor Internal Failure	Boolean	False, True		Indicates an internal failure
Monitor Mode	Enumerated	INIT		Initializing following power-up
		NORMAL		Normal mode of operation
		FAILED		Internal failure detected

A.5.2.1 Manage Monitor Interface Function.

The Manage Monitor Interface function defines the interaction with the Operator Interface external entity. These include obtaining the Alarm Range and reporting back the status of the Monitor Temperature Function. The constants are shown in table A-13.

Table A-13. The Manage Monitor Interface Function Constants

Name	Type	Value	Units	Physical Interpretation
Max Operator Response Time	Real	0.5	Sec	The time an operator will tolerate between an operator request or a change in the Thermostat state and the visible response
Rationale: A trade study has shown that this lag should be no more than 0.5 second.				

The requirements for the Monitor Status controlled variable are as follows:

- REQ-MMI-1: If the Manage Monitor Interface mode is INIT, the Monitor Status shall be set to Init.

- REQ-MMI-2: If the Manage Monitor Interface mode is NORMAL, the Monitor Status shall be set to On.

- REQ-MMI-3: If the Manage Monitor Interface mode is FAILED, the Monitor Status shall be set to Failed.

  Latency: < Max Operator Response Time
  Tolerance: N/A

The requirements for Monitor Interface Failure internal variable are as follows:

- REQ-MMI-4: If the Status attribute of the Lower Alarm Temperature or the Upper Alarm Temperature is Invalid, the Monitor Interface Failure shall be set to True.

- REQ-MMI-5: If the Status attribute of the Lower Alarm Temperature and the Upper Alarm Temperature is Valid, the Monitor Interface Failure shall be set to False.

  Rationale: The Monitor Interface Failure internal variable indicates if any errors have occurred in sensing the Operator Interface monitored variables needed by the Manage Temperature Function. Note that its initial value on power-up will always be True since the Status attribute of the Lower Alarm Temperature and the Upper Alarm Temperature will initially be Invalid.

The requirements for Alarm Range Internal variable are as follows:

- REQ-MMI-6: If the Monitor Interface Failure is False, the Alarm Range variable shall be set to the Desired Temperature Range.

- REQ-MMI-7: If the Monitor Interface Failure is True, the Alarm Range variable is UNSPECIFIED.

  Rationale: The Alarm Range variable is only meaningful when there is not a Monitor Interface Failure. If there is, its value should not used and it can be set to any value.

A.5.2.2 Manage Monitor Mode Function.

The Manage Monitor Mode Function determines the mode of the Monitor Temperature Function. The constants and definitions are shown in tables A-14 and A-15, respectively.

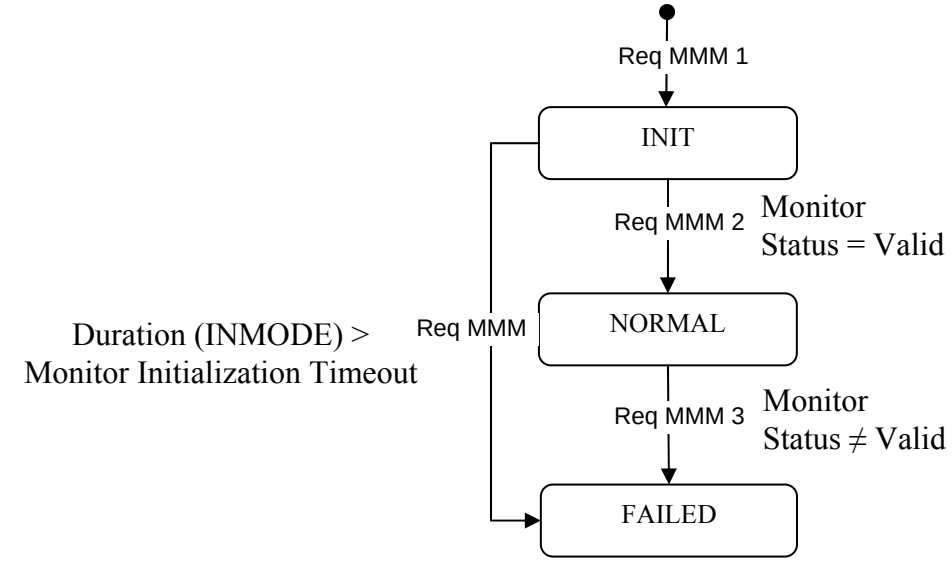
Table A-14. The Manage Monitor Mode Function Constants

Name	Type	Value	Units	Physical Interpretation
Monitor Initialization Timeout	Real	1.0	Sec	The time allowed for initialization of the Monitor Temperature Function before declaring failure.
Rationale: A trade study has shown that users become impatient if the Thermostat requires more than one second to initialize.				

Table A-15. The Manage Monitor Mode Function Definitions

Name	Type	Definition
Monitor Status	Boolean	NOT (Monitor Interface Failure OR Monitor Internal Failure) AND Current Temperature.Status = Valid

The modes and transitions of the Manage Monitor Mode Function are specified in the state transition diagram shown in figure A-6. Each transition is a separate requirement and is assigned a unique identifier (e.g., Req MMM 1). All transitions are assumed to occur in negligible time.



MMM = Manage Monitor Mode

Figure A-6. Monitor Temperature Mode Transition Diagram

Rationale: (Req MMM 3 and Req MMM 4) Once the monitor has failed, the only way for it to re-enter normal operation is for it to be powered off and on. This ensures that the operators are made aware of any transient failures that the monitor may be experiencing.

A.5.2.3 Manage Alarm Function.

The Manage Alarm Function turns the Alarm Control on when the Current Temperature of the Isolette falls below or rises above the Alarm Temperature Range.

The requirements for the Alarm Control controlled variable are as follows:

- REQ-MA-1: If the Monitor Mode is INIT, the Alarm Control shall be set to Off.

  Rationale: A monitor that is initializing should not activate the alarm unless it enters the FAILED mode.

- REQ-MA-2: If the Monitor Mode is NORMAL and the Current Temperature is less than the Lower Alarm Temperature or greater than the Upper Alarm Temperature, the Alarm Control shall be set to On.

- REQ-MA-3: If the Monitor Mode is NORMAL and the Current Temperature is greater than or equal to the Lower Alarm Temperature and less than the Lower Alarm Temperature +0.5°, or the Current Temperature is greater than the Upper Alarm Temperature -0.5° and less than or equal to the Upper Alarm Temperature, the value of the Alarm Control shall not be changed.

  Rationale: This provides a hysteresis that prevents transient alarms, see figure A-7.

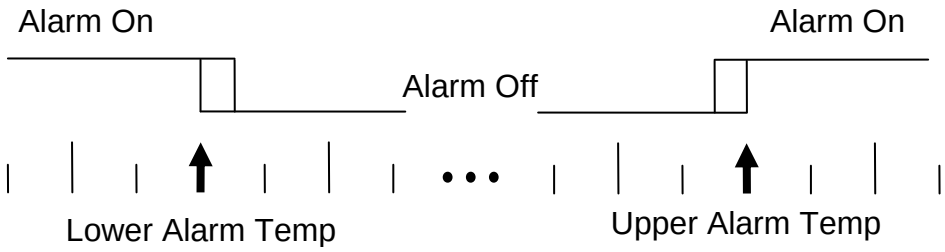


Figure A-7. Transient Alarm Hysteresis

- REQ-MA-4: If the Monitor Mode is NORMAL and the value of the Current Temperature is greater than or equal to the Lower Alarm Temperature +0.5° and less than or equal to the Upper Alarm Temperature -0.5°, the Alarm Control shall be set to Off.

  Rationale: This turns the alarm off at the same moment that the Displayed Temperature shows a value greater than the Lower Alarm Temperature and less than the Upper Alarm Temperature.

- REQ-MA-5: If the Monitor Mode is FAILED, the Alarm Control shall be set to On.

  Rationale: A failed monitor cannot monitor the Current Temperature of the Isolette and the Alarm should be turned on.

  Latency: <5 seconds
  Tolerance: N/A

  Rationale: Required by SR-2.

A.5.2.4 Detect Monitor Failure Function.

The Detect Monitor Failure Function identifies internal failures, (e.g., a memory check failure) in the Monitor Temperature Function. It defines a single Boolean-valued internal variable, Monitor Internal Failure, which is set to True if an internal failure is detected.

The requirements for Monitor Internal Failure variable are implementation-specific and cannot be specified until an implementation platform is chosen.

APPENDIX B—FLIGHT CONTROL SYSTEM EXAMPLE

This appendix contains a high-level specification for a simplified Flight Control System (FCS). The purpose of this example is to illustrate how a specification could be allocated to separate subsystems as discussed in section 2.10. For this reason, the example is only decomposed down to the Flight Guidance (FG) and Autopilot (AP) Functions. These functions are then further developed as separate subsystems specifications, which are shown in appendices C and D. However, detailed behavior and performance requirements are not included.

B.1 SYSTEM OVERVIEW.

The system being specified is a portion of an FCS. The FCS compares the measured Aircraft Attitude to a Reference Attitude and generates Flight Director (FD) Guidance commands that are displayed as visible cues, i.e., the FD, on the left and right Primary Flight Displays (PFD). The Pilot or Copilot can manually fly the aircraft to follow the FD to achieve the Reference Attitude. The Pilot or Copilot can clear the FD from the PFDs, turn the FD back on, and synchronize the Reference Attitude to the current Aircraft Attitude.

The FCS also provides an AP Function that the Pilot or Copilot can request. When the AP Function is engaged, the FCS generates Actuator Commands that will prompt the aircraft control surfaces to fly the aircraft to the Reference Attitude. While the AP Function is engaged, the Pilot or Copilot can initiate control wheel steering, which suspends the AP Function, allowing the Pilot or Copilot to manually fly the aircraft to a new attitude, and then resume the AP Function using the new attitude as the Reference Attitude.

In addition to the FD, the PFDs also display whether the AP Function is engaged, if the AP Function has failed, and if the FCS has failed.

B.1.1 SYSTEM CONTEXT.

The operational context of the FCS is shown in figure B-1.

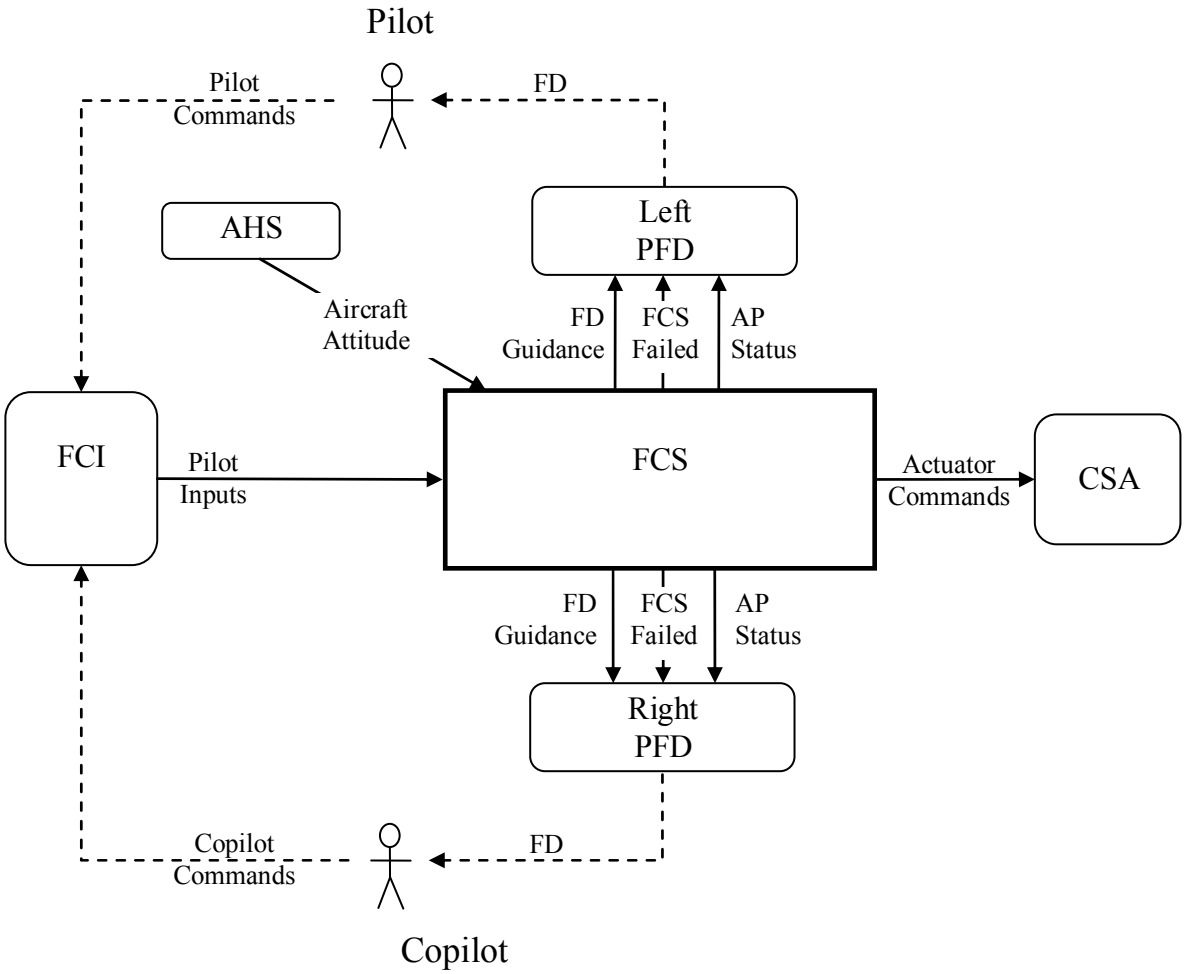


Figure B-1. Context Diagram for the Flight Control System

The FCS interacts directly with

- the Flight Crew Interface (FCI) that supplies the inputs provided by the Pilot and Copilot, such as the Display FD, Sync FD, and Engage AP requests.
- the Altitude Heading System (AHS) that provides the current Aircraft Attitude.
- the Right and Left PFDs that show the FD Guidance, the AP Status, and the FCS Failed indication.
- the Control Surface Actuators (CSA) that position the aircraft control surfaces based on the Actuator Commands.

The FCS also interacts indirectly with

- the Pilot and the Copilot, who view the FD, the AP Status, the FCS Failed indication on the PFD and provide the Pilot Inputs via the FCI.

B.1.2 SYSTEM GOALS.

The high-level goals of this system are as follows:

- G1—Provide guidance to the Pilot and Copilot to allow them to manually fly the aircraft to a desired attitude
- G2—Automatically fly the aircraft to hold at a desired attitude when the AP is engaged
- G3—Ensure that unsafe aircraft maneuvers are not commanded by the FCS
- G4—Ensure that aircraft maneuvers that might cause passenger discomfort are not commanded by the FCS
- G5—Keep the cost of manufacturing the FCS as low as possible
- G6—Keep the cost of operating the FCS as low as possible

B.2 OPERATIONAL CONCEPTS.

The use cases in tables B-1 through B-5 describe how the operators (the Pilot and Copilot) interact with the FCS.

Refer to table B-1 for use case 1: Pilot or Copilot activates and flies the FD.

Table B-1. Activating and Flying the FD

Purpose	Describes normal use of the FCS by the Pilot or Copilot when manually flying the aircraft to follow the FD command bars		
Primary Actor		Related Goals	
Pilot or Copilot		G1, G3, and G4	
Precondition		Postcondition	
Both FD are cleared AP Function is disengaged		Both FD are displayed AP Function is disengaged	
Main Success Scenario (Use Case 1)		Exception	Function
1. Pilot or Copilot requests display of FD.[1]			
2. FCS synchronizes Reference Attitude to current Aircraft Attitude		EC 1	B.4.1
3. Each PFD shows its FD at the Reference Attitude			
4. Pilot or Copilot flies the aircraft to desired attitude			
5. Pilot or Copilot requests synchronization of references[2]			
6. FCS synchronizes Reference Attitude to current Aircraft Attitude		EC 2	B.4.1
7. Each PFD gradually positions its FD to the new Reference Attitude			
8. Pilot or Copilot manually flies aircraft to follow the FD			
Exception Case EC 1			
1. FD fails to display due to unsafe attitude of aircraft			
2. Each PFD annunciates unsafe attitude to alert Pilot and Copilot			
3. Pilot or Copilot flies aircraft to safe attitude			
4. Return to step 1 of main success scenario			
Exception Case EC 2			
1. FCS fails to synchronize due to unsafe attitude			B.4.1
2. Each PFD annunciates unsafe attitude to alert Pilot and Copilot			
3. Pilot or Copilot flies the aircraft to safe attitude			
4. Return to step 4 of main success scenario			

[1]For example, by pressing the FD switch on the FCP
[2]For example, by pressing the SYNC button on the control yoke

Refer to table B-2 for use case 2: Pilot or Copilot clears the FD.

Table B-2. Clearing the FD

Purpose	Describes how Pilot or Copilot turns the FD off		
Primary Actor		Related Goals	
Pilot or Copilot		G1, G2, and G3	
Precondition		Postcondition	
Both FD are displayed		Both FD are cleared AP Function is disengaged	
Main Success Scenario (Use Case 2)		Exception	Function
1. Pilot or Copilot requests clearing of the FD[1]			B.4.1
2. Each PFD clears its FD		EC 1	
Exception Case 1 (EC 1)			
1. FD fails to clear because AP Function is engaged			B.4.2
2. Pilot or Copilot requests disengagement of the AP Function[2]			
3. AP disengages			B.4.2
4. Return to step 1 of main success scenario			

[1]For example, by pressing the FD switch on the FCP
[2]For example, by pressing the AP switch on the FCP

Refer to table B-3 for use case 3: Pilot or Copilot engages AP Function.

Table B-3. Engaging the AP Function

Purpose	Describes the engagement of the AP Function by the Pilot or Copilot when the FD is displayed		
Primary Actor		Related Goals	
Pilot or Copilot		G2 and G3	
Precondition		Postcondition	
Both FD are displayed AP Function is disengaged		Both FD are displayed AP Function is engaged	
Main Success Scenario (Use Case 3)		Exception	Function
1. Pilot or Copilot flies aircraft to the desired attitude			
2. Pilot or Copilot requests AP engagement*			
3. FCS synchronizes Reference Attitude with Aircraft Attitude		EC 1	B.4.1
4. Each PFD positions its FD to the new Reference Attitude			
5. FCS engages AP Function		EC 2	B.4.2
6. Each PFD annunciates AP engagement to Pilot and Copilot			
7. AP Function generates Actuator Commands to hold aircraft at Reference Attitude			B.4.2
Exception Case 1 (EC 1)			
1. FCS fails to synchronize due to unsafe attitude			B.4.1
2. Each PFD annunciates unsafe attitude to alert Pilot and Copilot			
3. Pilot or Copilot flies aircraft to safe attitude			
4. Return to step 2 of main success scenario			
Exception Case 2 (EC 2)			
1. AP Function fails to engage due to AP Function failure			B.4.2
2. Each PFD annunciates AP failure to alert Pilot and Copilot			
3. Pilot or Copilot flies aircraft manually			

*For example, by pressing the AP switch on the FCP

Refer to table B-4 for use case 4: Pilot or Copilot initiates control wheel steering.

Table B-4. Initiating Control Wheel Steering

Purpose	Describes how the Pilot or Copilot flies the aircraft to a new attitude when the AP is engaged		
Primary Actor		Related Goals	
Pilot or Copilot		G2, G3, and G4	
Precondition		Postcondition	
Both FD are displayed AP Function is engaged		Both FDs are displayed AP Function is engaged	
Main Success Scenario (Use Case 4)		Exception	Function
Pilot or Copilot requests control wheel steering[1]			
AP Function decouples[2] from CSA			B.4.2
Each PFD annunciates AP Function is decoupled			
FCS starts continuous synchronization of Reference Attitude to Aircraft Attitude			B.4.1
Pilot or Copilot flies the aircraft to desired attitude		EC 1	
Pilot or Copilot cancels control wheel steering[3]			
FCS stops synchronization of Reference Attitude			B.4.1
AP Function couples to CSA			B.4.2
Each PFD annunciates AP Function is coupled			
AP Function gradually steers aircraft to new Reference Attitude			
Exception Case 1 (EC 1)			
AP Function disengages due to unsafe attitude of aircraft			B.4.2
Each PFD annunciates unsafe attitude to alert Pilot and Copilot			
FCS stops synchronization of Reference Attitude			B.4.1
Pilot or Copilot flies the aircraft to safe attitude			
FCS resumes synchronization of Reference Attitude			B.4.1
Pilot or Copilot cancels control wheel steering			
FCS stops synchronization of Reference Attitude			B.4.1

[1]For example, by pressing and holding the SYNC button on the control yoke
[2]Temporarily disconnects AP from the CSA without disengaging the AP
[3]For example, by releasing the SYNC button on the control yoke

Refer to table B-5 for use case 5: pilot or copilot disengages the AP Function.

Table B-5. Disengaging the AP Function

Purpose	Describes how Pilot or Copilot disengages the AP Function		
Primary Actor		Related Goals	
Pilot or Copilot		G2	
Precondition		Postcondition	
Both FD are displayed AP Function is engaged		Both FD are displayed AP Function is disengaged	
Main Success Scenario (Use Case 5)		Exception	Function
Pilot or Copilot requests disengagement of the AP Function*			B.4.2
FCS disengages the AP Function		EC1	
Each PFD annunciates disengagement of the AP Function			

*For example, by pressing the AP switch on the FCP

B.3 EXTERNAL ENTITIES.

The following sections describe the external entities with which the FCS directly interacts—the FCI, the AHS, the CSA, and the left and right PFD. The monitored and controlled variables associated with each entity are listed along with any environmental assumptions made about the entity.

B.3.1 FLIGHT CREW INTERFACE.

The FCI provides the inputs from the Pilot and the Copilot that affect the behavior of the FCS. The monitored variables are shown in table B-6. No environmental assumptions are made.

Table B-6. Flight Control System Monitored Variables for FCI

Name	Type	Range	Units	Physical Interpretation
Pilot Inputs				Commands provided by Pilot or Copilot
Display FD	Boolean	False, True		Command to display or clear the FD to the Pilot and Copilot
	Status	●Invalid, Valid		
Sync FD	Boolean	False, True		Command to set Reference Attitude to current Aircraft Attitude
	Status	●Invalid, Valid		
Engage AP	Boolean	False, True		Command to activate or deactivate the AP Function
	Status	●Invalid, Valid		

● denotes initial value

B.3.2 ATTITUDE HEADING SYSTEM.

The AHS provides the current Aircraft Attitude to the FCS. The monitored variables are shown in table B-7.

Table B-7. Flight Control System Monitored Variables for Altitude Heading System

Name	Type	Range	Units	Physical Interpretation
Aircraft Attitude				Current attitude of the aircraft
Aircraft Roll	Real	[-180.0..179.9]	Degrees	Current roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
	Status	●Invalid, Valid		
Aircraft Pitch	Real	[-180.0..179.9]	Degrees	Current pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
	Status	●Invalid, Valid		

● denotes initial value

The following environmental assumptions are made:

- EA-AHS-1: The Aircraft Roll will range between -180.0° to +179.9°, inclusive.

  Rationale: The AHS will provide true roll of the aircraft, which can take on any value in the full range of motion of the aircraft.

- EA-AHS-2: The Aircraft Roll will be sensed to an accuracy of ±0.1°.

  Rationale: This accuracy is needed to ensure that the displayed aircraft attitude moves smoothly on the PFD and to ensure the FCS computes the Actuator Commands and FD Guidance commands with the necessary accuracy.

- EA-AHS-3: The Aircraft Pitch will range between -180.0° to +179.9°, inclusive.

  Rationale: The AHS will provide true pitch of the aircraft, which can take on any value in the full range of motion of the aircraft.

- EA-AHS-4: The Aircraft Pitch will be sensed to an accuracy of ±0.1°.

  Rationale: This accuracy is needed to ensure that displayed aircraft attitude moves smoothly on the PFD and to ensure the FCS computes the Actuator Commands and FD Guidance commands with the necessary accuracy.

B.3.3 CONTROL SURFACE ACTUATORS.

The CSA positions the aircraft control surfaces based on the Actuator Commands generated by the FCS to hold the aircraft to the Reference Attitude. The controlled variables are shown in table B-8.

Table B-8. Flight Control System Controlled Variables for Controlled Surface Actuators

Name	Type	Range	Units	Physical Interpretation
Actuator Commands				Commanded Actuator Rates
Roll Actuator	Real	[-20.0..20.0]	° surface/ second	Commanded rate of roll actuator: • 0 → no change of control surfaces • -X → left wing down. • +X → right wing down
Pitch Actuator	Real	[-20.0..20.0]	° surface/ second	Commanded rate of pitch actuator: • 0 → no change of control surfaces • -X → nose up • +X → nose down
AP On	Boolean	False, True		Indication whether the AP Function is on and the Actuator Commands will be used

The following environmental assumptions are made:

- EA-CSA-1: The Roll Actuator rate will range between -20.0 to +20.0° surface/second, inclusive.

  Rationale: Specified by aircraft manufacturer as the range necessary to adequately control the aircraft.

- EA-CSA-2: The Roll Actuator rate will be set to a resolution of 0.1° surface/second.

  Rationale: Controllability analysis shows that a resolution of 0.1° surface/second is necessary to maintain control of the aircraft.

- EA-CSA-3: The Pitch Actuator rate will range between -20.0 to +20.0° surface/second, inclusive.

  Rationale: Specified by aircraft manufacturer as the range necessary to adequately control the aircraft.

- EA-CSA-4: The Pitch Actuator rate will be set to a resolution of 0.1° surface/second.

  Rationale: Controllability analysis shows that a resolution of 0.1° surface/second is necessary to maintain aircraft control.

B.3.4 PRIMARY FLIGHT DISPLAY.

The Left and Right PFD shows the FD, FCS Failed indication, and the AP Status. The controlled variables are shown in table B-9.

Table B-9. Flight Control System Controlled Variables for PFD

Name	Type	Range	Units	Physical Interpretation
FD Guidance				Guidance Commands for FD
Roll Guidance	Real	[-45.0..45.0]	Degrees	Desired roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
Pitch Guidance	Real	[-45.0..45.0]	Degrees	Desired pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
FD On	Boolean	False, True		Indication if FD is to be displayed: • False → Do not display FD • True → Display FD
FCS Failed	Boolean	False, True		Indication if FCS is failed: • False → FCS is functioning • True → FCS is failed
AP Status	Enumerated	Failed, Off, On		Status of AP Function: • Failed → AP Function is failed • Off → AP Function is off • On → AP Function is on

The following environmental assumptions are made:

- EA-PFD-1: The Roll Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: Specified by aircraft manufacturer as the maximum range for the Roll Guidance.

- EA-PFD-2: The Roll Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the FD during control wheel steering.

- EA-PFD-3: The Pitch Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: Specified by aircraft manufacturer as the maximum range for the Pitch Guidance.

- EA-PFD-4: The Pitch Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the FD during control wheel steering.

B.4 FLIGHT CONTROL SYSTEM FUNCTIONS.

This section describes the main functions provided by the FCS.

The high-level requirements for the FCS Function are:

- REQ-FCS-1: The FCS shall generate the FD Guidance commands.

  Rationale: This is a primary function of the FCS. The PFD uses the FD Guidance commands to position the FD to show the Pilot and Copilot how to fly the aircraft to the Reference Attitude.

- REQ-FCS-2: The FCS shall set the Reference Attitude.

  Rationale: The FCS sets the Reference Attitude when the Pilot or Copilot requests the FCS to synchronize the Reference Attitude to the current attitude or to engage the AP Function.

- REQ-FCS-3: The FCS shall set FCS Failed status to indicate if the FCS Function has failed.

  Rationale: If the FCS has failed, the FD must be cleared and the Pilot and Copilot notified that the FCS Function has failed. This is typically indicated by the PFD.

- REQ-FCS-4: The FCS shall generate the Actuator Commands.

  Rationale: This is a primary function of the FCS. The CSA use the Actuator Commands to fly the aircraft to the Reference Attitude when the AP Function is on.

- REQ-FCS-5: The FCS shall set the AP Status to indicate the current status of the AP Function.

  Rationale: The current status of the AP Function must be displayed to the Pilot and Copilot to ensure they know when the AP Function is controlling the aircraft.

Refer to figure B-2 for the FCS functions.

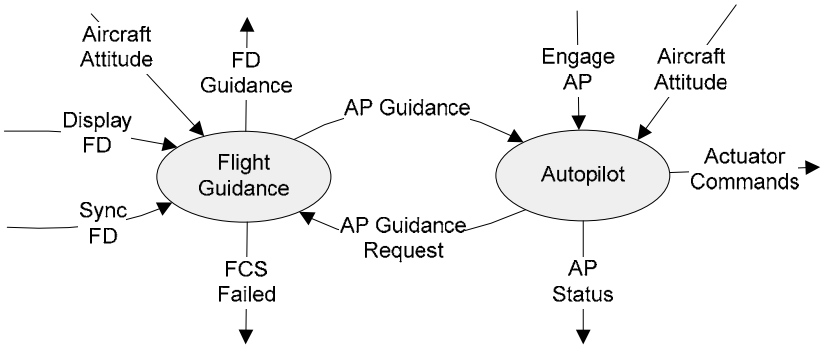


Figure B-2. Flight Control System Function Diagram

For the internal variables for the FCS refer to table B-10.

Table B-10. Flight Control System Internal Variables

Name	Type	Range	Units	Physical Interpretation
AP Guidance				Guidance Commands for AP
Roll Guidance	Real	[-45.0..45.0]	Degrees	Desired roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
Pitch Guidance	Real	[-45.0..45.0]	Degrees	Desired pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
Guidance Valid	Boolean	False, True		Indicates if the AP guidance is valid and can be used
AP Guidance Request	Boolean	False, True		Request for valid AP guidance

B.4.1 FLIGHT GUIDANCE FUNCTION.

The FG Function compares the measured Aircraft Attitude to a Reference Attitude and generates FD Guidance commands that are displayed as visible cues on the FD and AP Guidance commands that are used by the AP Function.

The high-level requirements for the FG Function are:

- REQ-FG-1: The FG Function shall generate the FD Guidance commands.

  Rationale: A primary function of the FG is to provide the FD Guidance commands for the FCS Function.

- REQ-FG-2: The FG Function shall set the Reference Attitude.

  Rationale: The FG Function sets the Reference Attitude when it is requested to synchronize the Reference Attitude to the current attitude (either by the Pilot, Copilot, or the AP Function).

- REQ-FG-3: The FG Function shall generate the AP Guidance commands.

  Rationale: A primary function of the FG Function is to provide the AP Guidance commands for the FCS Function.

- REQ-FG-4: The FG Function shall set the FCS Failed indication.

  Rationale: The FCS Function is split between the FG Function and the AP Function. The AP Function status is indicated independently to the Pilot and Copilot on the PFD. As a result, the status of the FCS Function (FCS Failed) is set by the FG Function.

B.4.2 AUTOPILOT FUNCTION.

The AP Function generates Actuator Commands from the AP Guidance commands provided by the FG Function. The Actuator Commands are used by CSA to fly the aircraft to the attitude specified by the AP Guidance commands.

The high-level requirements for the AP Function are:

- REQ-AP-1: The AP Function shall generate the Actuator Commands.

  Rationale: The primary function of the AP is to generate the Actuator Commands for the FCS Function.

- REQ-AP-2: The AP Function shall set the AP Status to indicate its status.

  Rationale: The AP Function is responsible for providing its current status to the FCS Function.

- REQ-AP-3: The AP Function shall generate the AP Guidance Request.

  Rationale: The AP Guidance Request calls for the FG Function to synchronize its Reference Attitude to produce an acceptable AP Guidance command.

APPENDIX C—FLIGHT GUIDANCE SYSTEM EXAMPLE

This appendix contains a high-level specification for a simplified Flight Guidance System (FGS). The purpose of this example is to illustrate how a specification could be allocated to separate subsystems, as discussed in section 2.10. It expands the Flight Guidance (FG) Function of the Flight Control System (FCS) specified in appendix B into a separate subsystem specification that could be given to a subcontractor. As such, it contains only the subset of the information from the FCS specification relevant to the FGS subsystem. This specification would be used as a starting point to be completed by the FCS contractor and the FGS subcontractor prior to developing the FGS. Further functional decomposition and definition of the detailed behavior and performance requirements would be needed to complete this specification.

C.1 SYSTEM OVERVIEW.

The system being specified is a portion of a FGS. The FGS compares the measured Aircraft Attitude to a Reference Attitude and generates Flight Director (FD) Guidance commands that are displayed as visible cues, i.e., the FD, on the Left and Right Primary Flight Displays (PFD). The Pilot or Copilot can manually fly the aircraft to follow the FD to hold the Reference Attitude. The Pilot or Copilot can also clear the FD from the PFDs, turn the FD back on, and synchronize the Reference Attitude to the current Aircraft Attitude.

The FGS also generates Autopilot (AP) Guidance commands that are used by an AP system to move the aircraft control surfaces to follow the AP guidance. At any time, the Pilot or Copilot can initiate control wheel steering, which instructs the AP to decouple from the Control Surface Actuators (CSA), allowing the Pilot or Copilot to manually fly the aircraft to a new attitude that is then followed by the AP.

C.1.1 SYSTEM CONTEXT.

The operational context of the FGS is shown in figure C-1.

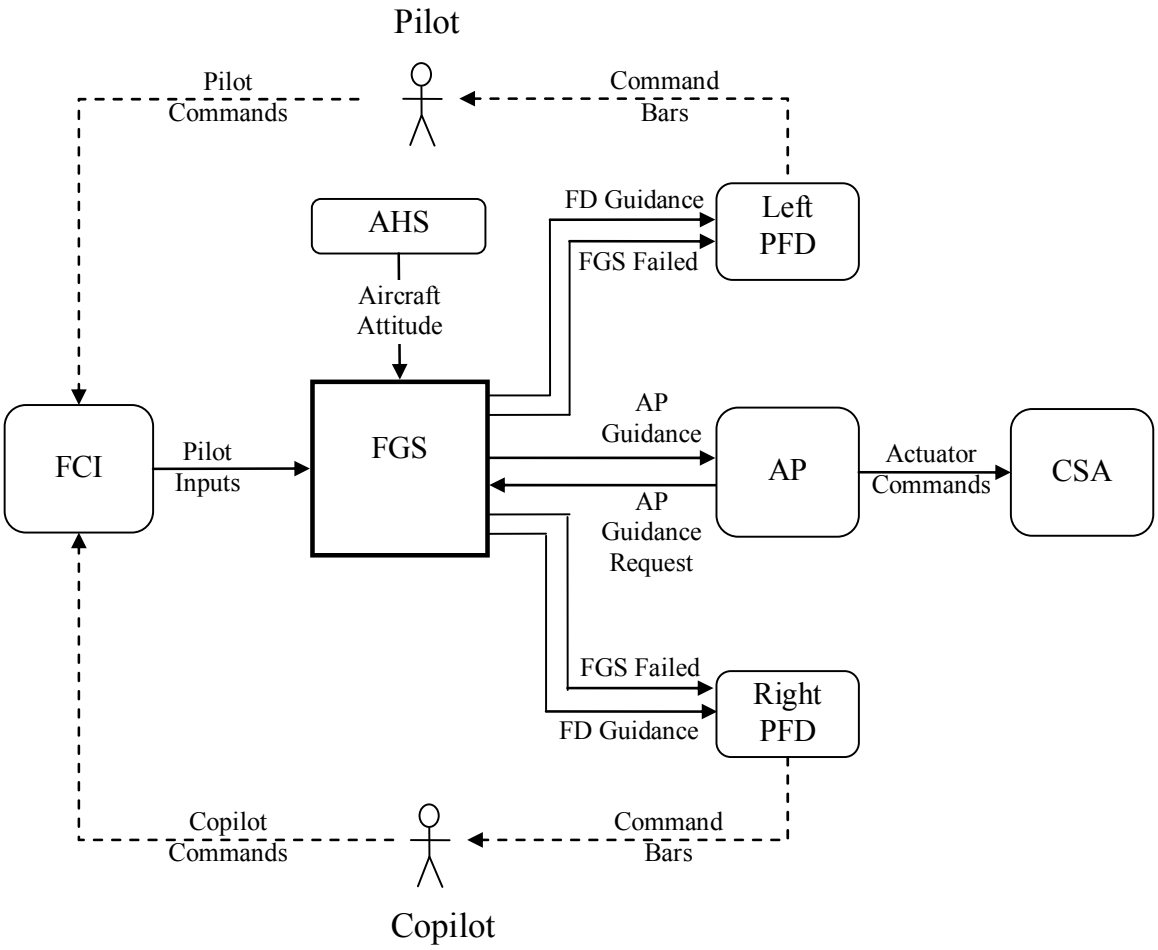


Figure C-1. Context Diagram for the FGS

The FGS interacts directly with

- the Flight Crew Interface (FCI) that supplies the inputs provided by the Pilot and Copilot, such as the Display FD and Sync FD commands.
- the Attitude Heading System (AHS) that provides the current Aircraft Attitude.
- the AP that translates the AP Guidance commands generated by the FGS to the Actuator Commands. The AP also sends AP Guidance Request commands to the FGS to request valid AP Guidance commands.
- the Right and Left PFDs that use the FD Guidance commands to position the command bars and display the FGS Failed indicator to the Pilot and Copilot.

The FGS also interacts indirectly with:

- the aircraft CSA that are moved by the Actuator Commands generated by the AP.
- the Pilot and the Copilot, who view the command bars displayed on the FD, the FGS Failed status on the PFD, and provide the Pilot Inputs via the FCI.

C.1.2 SYSTEM GOALS.

The high-level goals of this system are as follows:

- G1—Provide guidance to the Pilot and Copilot to allow them to manually fly the aircraft to a desired attitude
- G2—Provide guidance to the AP to allow it to automatically fly the aircraft to hold to a desired attitude
- G3—Ensure that unsafe aircraft maneuvers are not commanded by the FGS
- G4—Ensure that aircraft maneuvers that might cause passenger discomfort are not commanded by the FGS
- G5—Keep the cost of manufacturing the FGS as low as possible
- G6—Keep the cost of operating the FGS as low as possible

C.2 OPERATION/AL CONCEPTS.

The operational concepts of the FGS are provided here. This may consist of use cases that describe how the FGS interacts with the flight crew and other systems, such as the AP, or whatever other information the contractor and subcontractor agree on.

C.3 EXTERN/AL ENTITIES.

The following sections describe the external entities with which the FGS directly interacts—the FCI, the AHS, the AP, and the Left and Right PFD.

C.3.1 FLIGHT CREW INTERFACE.

The FCI provides the inputs from the Pilot and the Copilot that affect the behavior of the FGS. The monitored variables are shown in table C-1.

Table C-1. The FGS Monitored Variables for FCI

Name	Type	Range	Units	Physical Interpretation
Pilot Inputs				Commands Provided by Pilot or Copilot
Display FD	Boolean	False, True		Command to display or clear the FD
	Status	●Invalid, Valid		
Sync FD	Boolean	False, True		Command to set the Reference Attitude to current Aircraft Attitude
	Status	●Invalid, Valid		

● denotes initial value

No environmental assumptions are made.

C.3.2 ATTITUDE HEADING SYSTEM.

The AHS provides the current Aircraft Attitude to the FGS. The monitored variables are shown in table C-2.

Table C-2. The FGS Monitored Variables for AHS

Name	Type	Range	Units	Physical Interpretation
Aircraft Attitude				Current Attitude of the Aircraft
Aircraft Roll	Real	[-180.0..179.9]	Degrees	Current roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
	Status	●Invalid, Valid		
Aircraft Pitch	Real	[-180.0..179.0]	Degrees	Current pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
	Status	●Invalid, Valid		

● denotes initial value

The following environmental assumptions are made:

- EA-AHS-1: The Aircraft Roll will range between -180.0° to +179.9°, inclusive.

  Rationale: The AHS will provide true roll of the aircraft, which can take on any value in the full range of motion of the aircraft.

- EA-AHS-2: The Aircraft Roll will be sensed to an accuracy of ±0.1°.

  Rationale: This accuracy is needed to ensure that the displayed Aircraft Attitude moves smoothly on the PFD and to ensure the FGS computes the AP Guidance and FD Guidance commands with the necessary accuracy.

- EA-AHS-3: The Aircraft Pitch will range between -180.0° to +179.9°, inclusive.

  Rationale: The AHS will provide true pitch of the aircraft, which can take on any value in the full range of motion of the aircraft.

- EA-AHS-4: The Aircraft Pitch will be sensed to an accuracy of ±0.1°.

  Rationale: This accuracy is needed to ensure that the displayed Aircraft Attitude moves smoothly on the PFD and to ensure the FGS computes the AP Guidance and FD Guidance commands with the necessary accuracy.

C.3.3 AUTOPILOT.

The AP translates the guidance commands generated by the FGS into Actuator Commands that will move the aircraft's control surfaces to achieve the commanded changes about the lateral and vertical axes. The monitored and controlled variables are shown in tables C-3 and C-4, respectively.

Table C-3. The FGS Monitored Variables for AP

Name	Type	Range	Units	Physical Interpretation
AP Guidance Request	Boolean	False, True		Indication from the AP requesting the FGS resynchronize the Reference Attitude and provide valid guidance to the AP
	Status	●Invalid, Valid		

● denotes initial value

Table C-4. The FGS Controlled Variables for AP

Name	Type	Range	Units	Physical Interpretation
AP Guidance				Guidance commands for AP
Roll Guidance	Real	[-45.0..45.0]	Degrees	Desired roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
Pitch Guidance	Real	[-45.0..45.0]	Degrees	Desired pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
Guidance Valid	Boolean	False, True		Indicates if the Roll Guidance and Pitch Guidance are valid for use by the AP: • False → Invalid, do not use • True → Valid

The following environmental assumptions are made:

- EA-AP-1: The Roll Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the maximum range for the Roll Guidance.

- EA-AP-2: The Roll Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the aircraft during control wheel steering.

- EA-PFD-3: The Pitch Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the maximum range for the Pitch Guidance.

- EA-AP-4: The Pitch Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the aircraft during control wheel steering.

C.3.4 PRIMARY FLIGHT DISPLAY.

The Left and Right PFD show the FD and the FGS Failed status. The controlled variables are shown in table C-5.

Table C-5. The FGS Controlled Variables for PFD

Name	Type	Range	Units	Physical Interpretation
FD Guidance				Guidance Commands for FD
Roll Guidance	Real	[-45.0..45.0]	Degrees	Desired roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
Pitch Guidance	Real	[-45.0..45.0]	Degrees	Desired pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
FD On	Boolean	False, True		Indication if FD is to be displayed: • False → Do not display FD • True → Display FD
FGS Failed	Boolean	False, True		Indication if FGS is failed: • False → FGS is functioning • True → FGS is failed

The following environmental assumptions are made:

- EA-PFD-1: The Roll Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the maximum range for the Roll Guidance.

- EA-PFD-2: The Roll Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the FD during control wheel steering.

- EA-PFD-3: The Pitch Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the maximum range for the Pitch Guidance.

- EA-PFD-4: The Pitch Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the FD during control wheel steering.

C.4 FLIGHT GUIDANCE SYSTEM FUNCTIONS.

The FGS compares the measured Aircraft Attitude to a Reference Attitude and generates FD guidance commands that are displayed as visible cues on the FD and AP Guidance commands that are used by the AP Function.

The high-level requirements for the FGS Function are as follows:

- REQ-FGS-1: The FGS shall generate the FD Guidance commands.

  Rationale: A primary function of the FGS is to generate the FD guidance commands for the FCS.

- REQ-FGS-2: The FGS shall set the Reference Attitude.

  Rationale: The FGS sets the Reference Attitude when it is requested to synchronize the Reference Attitude to the current attitude (either by the pilot, copilot, or AP).

- REQ-FGS-3: The FGS shall generate the AP Guidance commands.

  Rationale: A primary function of the FGS is to generate the AP Guidance commands for the FCS.

- REQ-FGS-4: The FGS Function shall set the FCS Failed status.

  Rationale: The FCS is split between the FGS and the AP. The status of the AP is indicated independently to the Pilot and Copilot on the PFD. As a result, the status of the FCS (FCS Failed) is set by the FGS.

APPENDIX D—AUTOPILOT EXAMPLE

This appendix contains a high-level specification for a simplified Autopilot (AP). The purpose of this example is to illustrate how a specification could be allocated to separate subsystems, as discussed in section 2.10. It expands the AP Function of the Flight Control System (FCS) specified in appendix B.4.2 into a separate subsystem specification that could be given to a subcontractor. As such, it contains only the subset of the information from the FCS specification relevant to the AP subsystem. This specification would be used as a starting point to be completed by the FCS contractor and the AP subcontractor prior to development of the AP system. Further functional decomposition and definition of the detailed requirements would be needed to complete this specification.

D.1 SYSTEM OVERVIEW.

The system being specified is a simple AP. The AP accepts AP Guidance commands from the Flight Guidance System (FGS) and generates Actuator Commands to move the aircraft control surfaces to track the reference attitude maintained by the FGS. The Pilot or Copilot can engage the AP any time the aircraft is in a safe attitude. Prior to engagement, the AP will request the FGS to resynchronize the AP Guidance to the current Aircraft Attitude. While the AP is engaged, the Pilot or Copilot can initiate control wheel steering, which will decouple the AP from the Control Surface Actuators (CSA), allowing the Pilot or Copilot to manually fly the aircraft to a new attitude. Upon completion of control wheel steering, the AP will recouple to the CSA and track the new Reference Attitude. The Pilot or Copilot can request the AP to disengage at anytime.

The current status of the AP is displayed on the Left and Right primary flight display (PFD).

D.1.1 SYSTEM CONTEXT.

The operational context of the AP is shown in figure D-1.

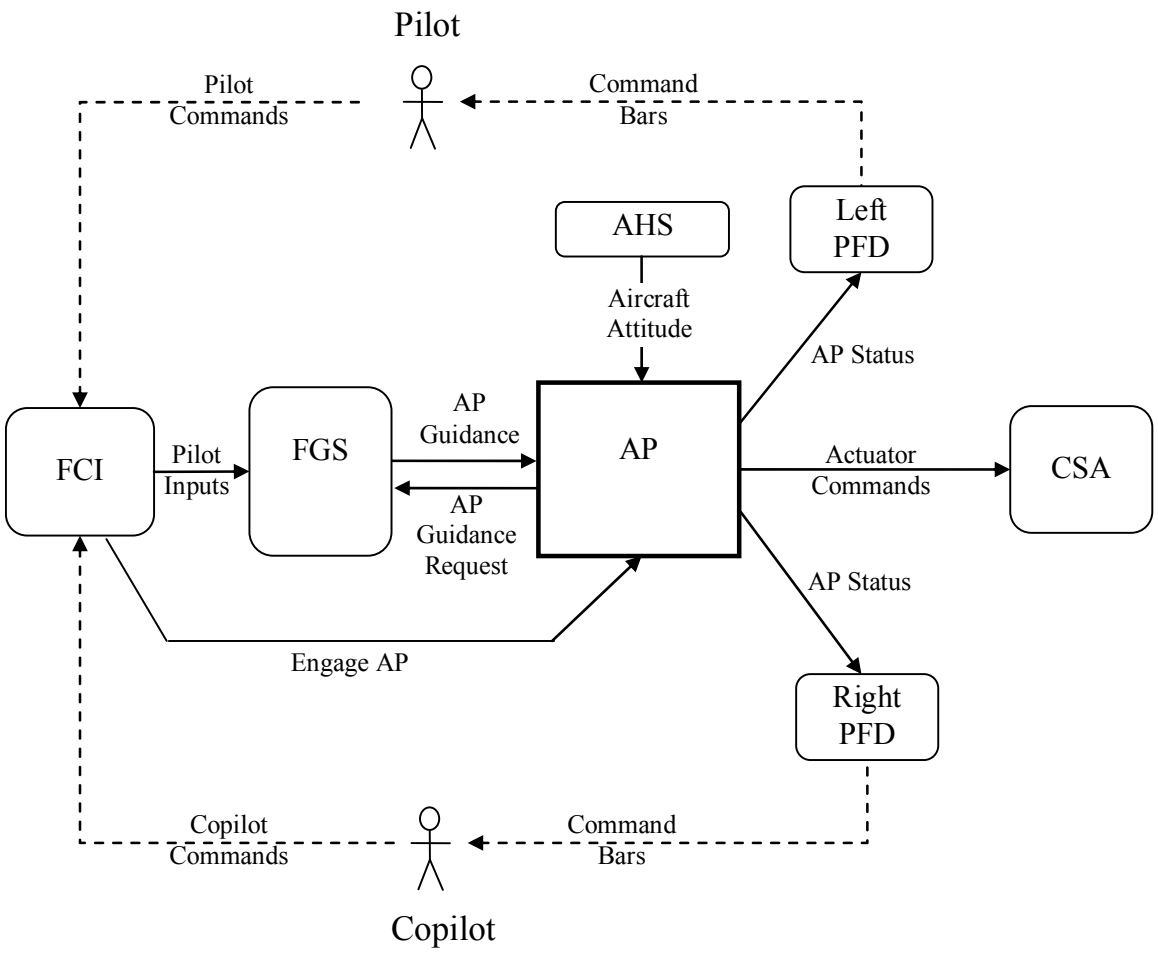


Figure D-1. Context Diagram for the Autopilot System

The AP interacts directly with

- the Flight Crew Interface (FCI), which provides the Engage AP command initiated by the Pilot or Copilot.
- the Attitude Heading System (AHS), which provides current attitude of the aircraft.
- the FGS, which provides the AP Guidance commands used by the AP to generate the Actuator Commands. The AP also sends AP Guidance Request commands to the FGS to request the AP Guidance be synchronized to the current Aircraft Attitude.
- the Right and Left PFDs, which display the AP Status.

The AP also interacts indirectly with

- the Pilot and the Copilot, who initiate the Engage AP command and view the AP Status displayed on the PFD.

D.1.2 SYSTEM GOALS.

The high-level goals of this system are as follows:

- G1—Generate the Actuator Commands to hold the aircraft to the attitude provided by the FGS
- G2—Ensure unsafe aircraft maneuvers are not commanded by the AP
- G3—Ensure aircraft maneuvers that could cause passenger discomfort are not commanded by the AP
- G4—Keep the cost of manufacturing the AP as low as possible
- G5—Keep the cost of operating the AP as low as possible

D.2. OPERATION/AL CONCEPTS.

The operational concepts of the AP are provided here. This could consist of use cases that describe how the AP system interacts with the flight crew and other systems, such as the FGS, or whatever other information the contractor and subcontractor agree on.

D.3 EXTERN/AL ENTITIES.

The following sections describe the external entities with which the AP directly interacts—the FCI, AHS, FGS, the Left and Right PFD, and the CSA.

D.3.1 FLIGHT CREW INTERFACE.

The FCI provides the inputs from the Pilot and the Copilot that affect the behavior of the AP. The monitored variables are shown in table D-1.

Table D-1. The AP Monitored Variables for FCI

Name	Type	Range	Units	Physical Interpretation
Engage AP	Boolean	False, True		Command to engage or disengage the AP
	Status	●Invalid, Valid		

● denotes initial value

No environmental assumptions are made.

D.3.2 ATTITUDE HEADING SYSTEM.

The AHS provides the current Aircraft Attitude to the AP. The monitored variables are shown in table D-2.

Table D-2. Autopilot Monitored Variables for AHS

Name	Type	Range	Units	Physical Interpretation
Aircraft Attitude				Current Attitude of the Aircraft
Aircraft Roll	Real	[-180.0..179.9]	Degrees	Current roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
	Status	●Invalid, Valid		
Aircraft Pitch	Real	[-180.0..179.9]	Degrees	Current pitch angle of the aircraft: • 0° indicates level fligh • -X° indicates X° nose up • +X° indicates X° nose down
	Status	●Invalid, Valid		

● denotes initial value

The following environmental assumptions are made:

- EA-AHS-1: The Aircraft Roll will range between -180.0° to +179.9°, inclusive.

  Rationale: The AHS will provide true roll of the aircraft, which can take on any value in the full range of motion of the aircraft.

- EA-AHS-2: The aircraft roll will be sensed to an accuracy of ±0.1°.

  Rationale: This accuracy is needed to ensure that the displayed Aircraft Attitude moves smoothly on the PFD and to ensure the FGS computes the AP Guidance and FD Guidance commands with the necessary accuracy.

- EA-AHS-3: The Aircraft Pitch will range between -180.0° to +179.9°, inclusive.

  Rationale: The AHS will provide true pitch of the aircraft, which can take on any value in the full range of motion of the aircraft.

- EA-AHS-4: The Aircraft Pitch will be sensed to an accuracy of ±0.1°.

  Rationale: This accuracy is needed to ensure that displayed Aircraft Attitude moves smoothly on the PFD and to ensure the FGS computes the AP Guidance and FD Guidance commands with the necessary accuracy.

D.3.3 FLIGHT GUIDANCE SYSTEM.

The FGS provides the guidance commands that the AP uses to generate actuator commands. The monitored and controlled variables are shown in tables D-3 and D-4, respectively.

Table D-3. Autopilot Monitored Variables for FGS

Name	Type	Range	Units	Physical Interpretation
AP Guidance				Guidance Commands for AP
Roll Guidance	Real	[-45.0..45.0]	Degrees	Desired roll angle of the aircraft: • 0° indicates wings level • -X° indicates X° bank to the left • +X° indicates X° bank to the right
	Status	●Invalid, Valid		
Pitch Guidance	Real	[-45.0..45.0]	Degrees	Desired pitch angle of the aircraft: • 0° indicates level flight • -X° indicates X° nose up • +X° indicates X° nose down
	Status	●Invalid, Valid		
Guidance Valid	Boolean	False, True		Indicates if the Roll Guidance and Pitch Guidance are valid for use by the AP: • False → Invalid, do not use • True → Valid
	Status	●Invalid, Valid		

● denotes initial value

Table D-4. Autopilot Controlled Variables for FGS

Name	Type	Range	Units	Physical Interpretation
AP Guidance Request	Boolean	False, True		Request to the FGS to resynchronize the Reference Attitude and provide valid guidance to the AP.

The following environmental assumptions are made:

- EA-FGS-1: The Roll Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the maximum range for the Roll Guidance.

- EA-FGS-2: The Roll Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the aircraft during control wheel steering.

- EA-FGS-3: The Pitch Guidance will range between -45.0° to +45.0°, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the maximum range for the Pitch Guidance.

- EA-FGS-4: The Pitch Guidance will be set in tenths of a degree.

  Rationale: This resolution is necessary to achieve smooth movement of the aircraft during control wheel steering.

D.3.4 PRIMARY FLIGHT DISPLAY.

The Left and Right PFD shows the AP Status. The controlled variables are shown in table D-5.

Table D-5. Autopilot Controlled Variables for PFD

Name	Type	Range	Units	Physical Interpretation
AP Status	Enumerated	Failed, Off, On		Status of AP Function: • Failed → AP Function is failed • Off → AP Function is off • On → AP Function is on
	Latency	0.1	sec	
	Tolerance	N/A		

N/A = nonapplicable

No environmental assumptions are made.

D.3.5 CONTROL SURFACE ACTUATORS.

The CSA position the aircraft control surfaces, based on the Actuator Commands generated by the FCS to hold the aircraft to the Reference Attitude. The controlled variables are shown in table D-6.

Table D-6. Autopilot Controlled Variables for CSA

Name	Type	Range	Units	Physical Interpretation
Actuator Commands				Commanded Actuator Rates
Roll Actuator	Real	[-20.0..20.0]	degree surface per second	Commanded rate of Roll Actuator: • 0 → no change of control surfaces • -X → left wing down • +X → right wing down
Pitch Actuator	Real	[-20.0..20.0]	degree surface per second	Commanded rate of Pitch Actuator: • 0 → no change of control surfaces • -X → nose up • +X → nose down
AP On	Boolean	False, True		Indication whether the AP Function is on and the Actuator Commands will be used

The following environmental assumptions are made:

- EA-CSA-1: The Roll Actuator rate will range between -20.0 to +20.0° surface/second, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the range necessary to adequately control the aircraft.

- EA-CSA-2: The Roll Actuator rate will be set to a resolution of 0.1° surface/second.

  Rationale: A controllability analysis shows that a resolution of 0.1° surface/second is necessary to maintain control of the aircraft.

- EA-CSA-3: The Pitch Actuator rate will range between -20.0 to +20.0° surface/second, inclusive.

  Rationale: This is specified by the aircraft manufacturer as the range necessary to adequately control the aircraft.

- EA-CSA-4: The Pitch Actuator rate will be set to a resolution of 0.1° surface/second.

  Rationale: A controllability analysis shows that a resolution of 0.1° surface/second is necessary to maintain control of the aircraft.

D.4 AUTOPILOT FUNCTIONS.

The AP generates Actuator Commands from the AP Guidance commands provided by the FG Function. The actuator commands are used by CSA to fly the aircraft to the attitude specified by the AP Guidance commands.

The high-level requirements for the AP system are as follows:

- REQ-APS-1: The AP system shall generate the Actuator Commands.

  Rationale: The primary function of the AP is to generate the Actuator Commands for the FCS.

- REQ-APS-2: The AP shall set the AP Status to indicate its status.

  Rationale: The AP is responsible for providing its current status to the FCS.

- REQ-APS-3: The AP shall generate the AP Guidance Request.

  Rationale: The AP Guidance Request calls for the FGS to synchronize its reference Attitude to produce an acceptable AP Guidance command.