Requirements Specification:

- A structured document that sets out the services the system is expected to provide.

- Should be precise so that it can act as a contract between the system procurer and software developer. Needs to be understandable by both.

- Describes what the system will do but not how it will do it (objectives but not how objectives will be achieved.

Design Specification:

- An abstract description of the software that serves as a basis for (or describes) detailed design and implementation.

- Describes how the requirements will be achieved.

- Primary readers will be software designers and implementers rather than users or management.

- Goals and constraints specified in requirements document should be traceable to the design specification (and from there to the code.)
Contents of Requirements Documents

**Introduction:** Describes the need for the system and places it in context, briefly describing its functions and presenting a rationale for the software. Describes how the system fits into the overall business or strategic objectives of the organization commissioning the software.

**System Model:** Shows the relationships between the system components and the system and its environment. An abstract data model should also be described if appropriate to the type of system.

**System Evolution:** Fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.

**Functional Requirements:** The services provided for the user. This includes timing and accuracy requirements.
Contents of Requirements Documents (2)

**Constraints:** Constraints on how the goals can be achieved (restrictions on behavior of software and freedom of designer), e.g., safety, hardware, programming languages, standards that must be followed. Includes quality requirements such as maintainability, availability, etc.

**Priorities:** Guides tradeoffs and design decisions if all requirements and constraints cannot be completely achieved.

**Interfaces to the Environment:** Input or output interfaces and relevant assumptions about environmental components with which the software will be interacting.

**Glossary:** Definitions of technical terms used in the document.

**Indexes:** Various types of indexes may be provided.
Attributes of a good requirements document:

- Readable and understandable by customers, users, and designers.
- Specifies only external system behavior (black box)
- Structured to be easy to change.
- Specifies both goals and constraints.
- Able to serve as a reference for system maintainers.
- Consistent, complete, unambiguous, realistic, and testable
- Specified acceptable responses to undesired events.
- Specifies what should not do as well as what should do.
- Specified incremental subsets if desired or minimum and maximum functionality
- Specifies changes anticipated in the future (for both environment and software)
Requirements must be testable

An untestable requirement:

The system shall be easy to use by experienced controllers and shall be organized in such a way that user errors are minimized.

A testable requirement:

Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.
Types of Specifications

• Informal
  – Free form, natural language
  – Ambiguity and lack of organization can lead to incompleteness, inconsistency, and misunderstandings

• Formatted
  – Standardized syntax (e.g., UML)
  – Basic consistency and completeness checks
  – Imprecise semantics implies other sources of error may still be present.
Types of Specifications (2)

- Formal
  - Syntax and semantics rigorously defined.
  - Precise form, perhaps mathematical.
  - Eliminate imprecision and ambiguity.
  - Provide basis for mathematically verifying equivalence between specification and implementation.
  - May be hard to read without training.
  - Semantic distance too great?
A program is a mathematical object

A programming language is a mathematical language.

Therefore, we can prove properties about the program.

e.g. does it do what it is supposed to do does it not do anything harmful

Building a model like engineers do, but need discrete rather than continuous mathematics.
Input–Output Assertions

\[ S \{P\} Q \]

If \( S \) holds before execution of \( S \), then \( Q \) holds afterward.

Examples:
1. \( \text{sum} = 0 \) \{ for \( i=1 \) to \( n \) do \( \text{sum}:=\text{sum}+\text{a}(i) \) \} \( \text{sum} = \sum_{j=1}^{n} a_j \)

2. \textbf{proc} \( \text{search(A,n,x)} \) \textbf{int;}

\begin{align*}
\textbf{pre} & \quad n \geq 0 \\
\textbf{post} & \quad (\text{result} = 0 \land \forall i \in \{1,\ldots,n\} : \text{A}[i] \neq x) \lor \\
& \quad (\text{result} = i \land 1 \leq i \leq n \land \text{A}[i] = x \land \\
& \quad \forall i \in \{1,\ldots,i-1\} : \text{A}[i] \neq x) \\
\end{align*}
Abstract Model Specifications

• Build an abstract model of required software behavior using mathematically defined (perhaps using axioms) types (e.g., sets, relations).

• Define operations by showing effects of that operation on the model.

• Specification includes:
  – Model
  – Invariant properties of model
  – For each operation:
    name
    parameters
    return values
  – Pre and post conditions on the operations
Z (pronounced Zed)

• Z specifications are made up of "schemas"

• A schema is a named, relatively short piece of specification with two parts:
  – Above the line: the definition of the data entities
  – Below the line: the definition of invariants that hold on those data entities
Z: Defining the Abstract Model

Library
books: \( P \) BOOK
status: BOOK \( \mapsto \) STATUS
books = dom status

• Declaration says library has two visible parts of its state:
  − books is a set of BOOKS, which are atomic elements.
  − status is a partial function that maps a BOOK into a STATUS
    (which is another atomic element that can take values In or Out)

• The invariant says the set of books is precisely the same as the domain of the function status.
  − Says every book in the Library has exactly one status
  − Two books may have the same status.

Example of a legal state for Library is:
books = \{Principia Mathematica, Safeware\}
status = (Principia Mathematica \( \mapsto \) In,
        Safeware \( \mapsto \) Out)
Z : Defining Operations

Borrow

Δ Library
book?: BOOK

status (book?) = In
status’ = status ⊕ (book? → Out)

• Δ Library declaration says operation modifies state of Library

• book? is the input

• A prime indicates the value after the operation

• The first invariant defines a pre-condition on the operation, i.e., the book to be borrowed must be currently checked in.

• The second invariant defines the semantics of borrowing, i.e., it overwrites the entry in the status function for the borrowed book.
Example: After a borrow operation, the set of books in the library remains unchanged.

\[
books' = books
\]

\[
books' = \text{dom} \quad \text{status}'
\]

\[
= \text{dom} \quad (\text{status} \oplus \{\text{book?} \mapsto \text{Out}\})
\]

\[
= \text{dom} \quad (\text{status} \cup \{\text{book?} \mapsto \text{Out}\})
\]

\[
= \text{dom} \quad \text{status} \cup \text{dom} \quad \{\text{book?} \mapsto \text{Out}\}
\]

\[
= \text{book} \cup \text{book}
\]

\[
= \text{book}
\]

[true because first invariant of Borrow implies that book? is an element of books]

Copyright © Nancy Leveson, Sept. 1999
Example of a State Machine Model

Water level low

Reading at set point / Close drain pipe

Water level at setpoint

High reading / Open drain pipe

Low reading / Activate pump

Reading at setpoint / Turn off pump
Cruise Control Off

- Cruise Control On and in Standby Mode
  - cruise control turned on / initialize cc
  - cruise control turned off

Increasing Speed

- set point reached / reduce throttle

Maintaining Speed

- read wheel turning rate / adjust throttle

- brake depressed or accelerator depressed / discontinue cruise control

Cruise Control On and in Standby Mode

- increase speed commanded / send command to throttle to increase at X rate
SpecTRM

- System engineering tools for software-intensive systems
  - A "CATIA" for the logical parts of the system
  - Requirements errors found early while cheaper to fix
  - Goal of enhancing communication and expert review

- Integrates hazard analysis into engineering decision-making environment
  - Information available when needed and in form that has maximum impact on decisions.

- Documentation of design rationale

- Complete traceability
  - For verification and validation
  - To support change and upgrade process

- Based on
  - Cognitive engineering research on how experts solve problems
  - Basic principles of system engineering
<table>
<thead>
<tr>
<th>Intent</th>
<th>Environment</th>
<th>Operator</th>
<th>System</th>
<th>Verification Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0: Program Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1: System Purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2: System Design Principles</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3: System Architecture</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 4: Design Representation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 5: Physical Representation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 6: System Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Part–Whole**
- **Refinement**
- **Environment Validation**
- **Verification**

- *Management View*
- *Customer View*
- *System Engineering View*
- *(Interface between System and Component Engineers)*
- *(Component Designer View)*
- *(Component Implementer View)*
- *(Operations View)*
<table>
<thead>
<tr>
<th>Level 0</th>
<th>Environment</th>
<th>Operator</th>
<th>System and components</th>
<th>V&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog. Mgmt.</td>
<td>Project management plans, status information, safety plan, etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>Assumptions [Constraints]</td>
<td>Responsibilities [Requirements I/F requirements]</td>
<td>System goals, high-level requirements, design constraints, limitations</td>
<td>Preliminary Hazard Analysis, Reviews</td>
</tr>
<tr>
<td>System Purpose</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2</td>
<td>External interfaces</td>
<td>Task analyses</td>
<td>Logic principles, control laws, functional decomposition and allocation</td>
<td>Validation plan and results, System Hazard Analysis</td>
</tr>
<tr>
<td>System Principles</td>
<td></td>
<td>Task allocation [Controls, displays]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3</td>
<td>Environment models</td>
<td>Operator Task models [HCI models]</td>
<td>Blackbox functional models [Interface specifications]</td>
<td>Analysis plans and results, Subsystem Hazard Analysis</td>
</tr>
<tr>
<td>System Architecture</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 4</td>
<td>HCl design</td>
<td></td>
<td>Software and hardware design specs</td>
<td>Test plans and results</td>
</tr>
<tr>
<td>Design Rep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 5</td>
<td>GUI design, physical controls design</td>
<td></td>
<td>Software code, hardware assembly instructions</td>
<td>Test plans and results</td>
</tr>
<tr>
<td>Physical Rep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 6</td>
<td>Audit procedures</td>
<td>Operator manuals [Maintenance Training materials]</td>
<td>Error reports, change requests, etc.</td>
<td>Performance monitoring and audits</td>
</tr>
<tr>
<td>Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Level 1: Environment

- Description of environment in which interacts
- Assumptions about environment
  
  EA–1: Altitude information is available from intruders with minimum precision of 100 feet
  
  EA–2: All aircraft will have legal identification numbers

Level 1: Operator

- Pilot Responsibilities and Tasks
- Operator requirements
  
  OP–5: TCAS advisories shall be executed in such a way as to minimize the aircraft’s deviation from it’s ATC clearance

Human–Machine Interface Requirements

HMI–3: A red visual alert shall be provided in the primary field of view for each pilot for resolution advisories.
Level 1 Functional Goals:

G1: Provide affordable and compatible collision avoidance system options for a broad spectrum of National Airspace users.

Level 1 Functional Requirements

FR–1: TCAS shall provide collision avoidance protection for any two aircraft closing horizontally at any rate up to 1200 knots and vertically up to 10,000 feet per minute.

Assumption: Commercial aircraft can operate up to 600 knots and 5000 fpm during vertical climb or controlled descent and therefore the planes can close horizontally up to 1200 knots and vertically up to 10,000 pfm.
TCAS does not display a resolution advisory.

- TCAS unit is not providing RAs.
  - <Self−monitor shuts down TCAS unit>
  - Sensitivity level set such that no RAs are displayed.
  - No RA inputs are provided to the display.
- No RA is generated by the logic
  - Inputs do not satisfy RA criteria
    - Surveillance puts threat outside corrective RA position.
      - Surveillance does not pass adequate track to the logic
        - <Threat is non−Mode C aircraft> \[→L.5\]
        - <Surveillance failure> \[→1.23.1\]
        - <Surveillance error causes incorrect range or range rate to be calculated>
      - Altitude reports put threat outside corrective RA position
        - Altitude errors put threat on ground
          - <Uneven terrain> \[→2.19\]
          - <Intruder altitude error>
          - <Own Mode C altitude error> \[→1.23.1\]
          - <Own radar altimeter error> \[→1.23.1\]
        - Altitude errors put threat in non−threat position.
          - ...
      - <Intruder maneuver causes logic to delay RA beyond CPA>
  - ...
- <Process/display connectors fail>
  - <Display is preempted by other functions> \[→SC4.8 \[→2.22\]
  - <Display hardware fails> \[→1.23.1\]
TCAS displays a resolution advisory that the pilot does not follow.

- Pilot does not execute RA at all.
  - Crew does not perceive RA alarm.
    - <Inadequate alarm design> ➞ 1.4 to 1.14  ▼ 2.74, 2.76
    - <Crew is preoccupied>
    - <Crew does not believe RA is correct.> ➞ OP.1
  - ...

- Pilot executes the RA but inadequately
  - <Pilot stops before RA is removed> ➞ OP.10
  - <Pilot continues beyond point RA is removed> ➞ OP.4
  - <Pilot delays execution beyond time allowed> ➞ OP.10
Level 1: System Limitations

L–5: TCAS provides no protection against aircraft with non-operational or non-Mode C transponders [FTA–370]
Level–1 Safety Constraints and Requirements

SC–5: The system must not disrupt the pilot and ATC operations during critical phases of flight nor disrupt aircraft operation. [H3]

SC–5.1: The pilot of a TCAS–equipped aircraft must have the option to switch to the Traffic–Advisory mode where traffic advisories are displayed but display of resolution advisories is prohibited [2.37]

Assumption: This feature will be used only during final approach to parallel runways when two aircraft are projected to come close to each other and TCAS would call for an evasive maneuver [6.17]
SC–7: TCAS must not create near misses (result in a hazardous level of vertical separation that would not have occurred had the aircraft not carried TCAS) [H1]

SC–7.1: Crossing maneuvers must be avoided if possible. [2.36, 2.38, 2.48, 2.49.2]

SC–7.2: The reversal of a displayed advisory must be extremely rare [2.51, 2.56.3, 2.65.3, 2.66]

SC–7.3: TCAS must not reverse an advisory if the pilot will have insufficient time to respond to the RA before the closest point of approach (four seconds or less) or if own and intruder aircraft are separated by less then 200 feet vertically when ten seconds or less remain to closest point of approach [2.52]
Example Level–2 System Design for TCAS

SENSE REVERSALS

2.51 In most encounter situations, the resolution advisory sense will be maintained for the duration of an encounter with a threat aircraft. [SC–7.2]

However, under certain circumstances, it may be necessary for that sense to be reversed. For example, a conflict between two TCAS–equipped aircraft will, with very high probability, result in selection of complementary advisory senses because of the coordination protocol between the two aircraft. However, if coordination communications between the two aircraft are disrupted at a critical time of sense selection, both aircraft may choose their advisories independently. [FTA–1300]

This could possibly result in selection of incompatible senses. [FTA–395]

2.51.1 [Information about how incompatibilities are handled]
Level 3 Specification (modeling) language goals

• Specify allocation of functionality to components

• Readable and reviewable
  - Minimize semantic distance
  - Minimal: blackbox behavior only (transfer function)
  - Easy to learn
  - Unambiguous and simple semantics
  - Visualization tools

• Complete (can specify everything need to specify)
  Analyzable (formal, mathematical foundation)
    - Executable (acts as a prototype)
      Animation and simulation
    - Tools to check completeness, consistency, nondeterminism
    - Includes human (operator) procedures and analysis

• Extensible (e.g., connecting to MATLAB, Simulink)
  API, built on Eclipse
SpecTRM–RL

- Combined requirements specification and modeling language
- A state machine with a domain–specific notation on top of it.
  - Includes a task modeling language
  - Can add other notations and visualizations of state machine
- Enforces or includes most of completeness criteria
- Supports specifying systems in terms of modes
  - Control modes
  - Operational modes
  - Supervisory modes
  - Display modes
SpecTRM Model of HETE Attitude Control System

HETE ACS

CONTROL MODES
- Wait
- Detumble
- Spinup
- Reorient
- Deploy Wheel
- Acquire
- Deploy Paddles
- Orbit Day
- Orbit Night
- Ground Command

Paddles
- Deployed
- Not deployed
- Unknown

Wheel
- Deployed
- Not deployed
- Unknown

Orbit
- Day
- Night
- Unknown

Optical System
- Tracking
- Not tracking
- Unknown

Sun Sensors
- Azimuth Angle
- Elevation Angle

Magnetometers
- Magnetic Fields (X,Y,Z)
- Bias

Mission Ops

Torque Coils

Momentum Wheel
= Detumble (Mode 1)

The purpose of detumble mode is to minimize the magnitude of body momentum vector in the X–Z plane. As soon as the magnitude falls below a threshold, the software should transition to spinup mode. The mode delay provides hysteresis in the mode transitions to prevent the software from jumping between modes too rapidly.

In detumble mode, the wheel actuator shall be controlled such that the wheel maintains the velocity it had upon entering the mode, and the magnetic moment along the Y axis shall be controlled to minimize the angular velocity about the X and Z axes.

<table>
<thead>
<tr>
<th>Control Mode</th>
<th>Wait</th>
<th>Detumble</th>
<th>Spinup</th>
<th>Ground Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Values</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time since entered wait &gt;= 10 sec</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time since entered detumble &lt; 100 sec</td>
<td>T</td>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xz momentum error &gt; xz momentum error threshold</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>Time since entered spinup &gt;= 100 sec</td>
<td></td>
<td></td>
<td></td>
<td>T T</td>
</tr>
<tr>
<td>Paddles in–state deployed</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Optical system in–state tracking</td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Time since entered ground control &gt;= 10 sec</td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
</tbody>
</table>

OR
<table>
<thead>
<tr>
<th>Name</th>
<th>Output Command</th>
</tr>
</thead>
</table>

Destination:
Acceptable Values:
- Units:
- Granularity:
- Exception Handling:
- Hazardous Values:

Timing Behavior:
- Initiation Delay:
- Completion Deadline:
- Output Capacity Assumptions:
  - Load:
  - Min time between outputs:
  - Max time between outputs:

Hazardous timing behavior:
- Exception Handling:

Feedback Information:
- Variables:
- Values:
- Relationship:
  - Min. time (latency):
  - Max. time:
- Exception Handling:

Reversed By:
Comments:
References:

DEFINITION

= ...
Requirements Analysis

- Model Execution, Animation, and Visualization
- Completeness
- State Machine Hazard Analysis (backwards reachability)
- Human Task Analysis
- Test Coverage Analysis and Test Case Generation
- Automatic code generation
Does It Work?

- It is being used for aerospace and automotive systems.
  - Have found important errors in requirements
  - Very complex systems modeled

- Level 3 models used to maintain TCAS II for past 10 years
  - All suggested changes and upgrades first modeled and evaluated through simulation.
Summary

- Integrate design rationale and safety information into specification and its structure
- Capture domain knowledge (reusable architectures)
- Provides traceability from high-level requirements to detailed design and code.
- Blackbox models at Level 3
  - Executable and analyzable
    - e.g., completeness, robustness, mode confusion, hazard analysis, test case generation, code generation
  - Specification acts as an executable prototype
    - Can interface with system simulation
  - Visualization tools
  - Interface to contractors