Tell me what you want, what you really really want —
A guide to effective requirements engineering

22 Oct 2012

Dr. William Bail
The MITRE Corporation

The authors’ affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or view points expressed by these authors.
Overview

- Requirements need to be managed carefully from the beginning of development to product delivery
  - And into operation (sustainment)
- Current practice falls short in many ways
  - Resulting in unnecessary rework and shortcomings in deployed systems
- This tutorial examines
  - Underlying principles beneath the idea of “software system requirements”
  - Practical guidance for defining and handling requirements
  - Common pitfalls and traps that plague current practice
Agenda

Motivation

- Terminology
- IEEE view
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
Motivation

- Why do we care so much about requirements?
  - Why not just start building, figure out what you want later
  - Use a Lean Rapid Agile $6\sigma$ Enterprise Spiral Prototyping process
  - Build – Test – Fix

- Yes, but...experience clearly shows that
  - Requirements form foundation of all system development
  - If we don’t handle them properly, we incur significant risk
  - You must respect them

- The longer defects fester in a system, the more expensive to repair
  - To repair a requirements defect during system I&T, cost can be as large as 130 times the cost of repair during requirements analysis

“You get what you spec, not what you expect”
Relative cost of repairing defects

- Sources: Davis, Basili, et al

<table>
<thead>
<tr>
<th>Phase When Defect Introduced into System</th>
<th>Req Anl</th>
<th>Design</th>
<th>Code</th>
<th>Test</th>
<th>Int Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req Anl</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>50</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td>130</td>
<td>26</td>
<td>13</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Operations</td>
<td>368</td>
<td>64</td>
<td>37</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

N.A.
Motivation – Defense Science Board

  - Requirements management viewed as being a fundamental problem
  - Requirements setting and management are still the hardest parts of SW development
  - Problem seen with overspecification of requirements
  - Underutilization of modern technical and management practices for requirements
Agenda

- Motivation

Terminology

- IEEE view
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
Basic terminology

What is a requirement?

Requirements...

Huh?

Threshold
Performance
Acceptability
Specifications
Objectives
Expectations
KPPs
Desires
Needs
System Specification
Mission
Capabilities
Necessities
Requirements
“Requirements”

- The word is generic and vague – many meanings
  - “This is what we require”
- Our industry has grabbed on this word and overloaded it
- But what do we really mean?
- Word used in many different ways:
  - The capabilities we need
  - The behaviors/functions we want
  - The programmatics we expect (cost, schedule, …)
  - What the system actually does
  - The system design we want
  - The systems it will interact with
- Pitfall - Lack of a consistent and accurate definition results in misunderstandings and inefficiencies
Clarification

- In general English usage, a requirement is something that is necessary or desired
  - With software, generally used to denote a system or development attribute that is to be realized when the system is built
- Systems have many different types of attributes, incl:
  - Behavior - “Alert fire department when smoke detected”
  - Schedule to develop - “System is ready for use on Sep 1”
  - Appearance - “User interfaces are easy to use”
  - Development location - “Development took place in Philadelphia”
  - Design/architecture - “System uses a layered architecture”
- Each of these can be defined as a “requirement” for the system and its development if this is what the customer/stakeholders want
System attributes

- A “requirement” is simply a statement about a system attribute that is needed / desired / envisioned / ...
- When we say
  - “the system must measure the current temperature in a room, and adjust the heater or air conditioner appropriately to maintain a temperature defined by the user”
- We are saying that we require the system to have this specific attribute
- Why is that important???
  - There is a temporal aspect to a “requirement”
  - The implication is that it is stated before the system is built
  - Once the system is built, the actual attributes can be referred to as existing characteristics and defined in a product specification
Time-scale nature of attributes

- Up-front, stakeholders define what they want the system to be
  - “Build-to” attributes that document what the software is supposed to do (and what it is supposed to be like)
- These guide the development of the software
  - And inform the developers about what needs to be built
- After the system is built, “As-built” attributes document what the software actually does
  - What was really built
  - To support maintenance, enhancements, user manuals, …
Build-to vs As-Built

- Often (usually), they differ
  - Requirements do not end up being implemented exactly as originally envisioned
  - Sometimes less than the attributes that were required
  - Sometimes more than …

What the SW actually does “as built”

What was not implemented

What is required “build to”

What needed to be changed
Agenda

- Motivation
- Terminology

**IEEE view**
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
IEEE definition of “requirement”

  - (1) A condition or capability needed by a user to solve a problem or achieve an objective.
  - (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
  - (3) A documented representation of a condition or capability as in (1) or (2).
  - See also: design requirement; functional requirement; implementation requirement; interface requirement; performance requirement; physical requirement.

Present – “as-built”

Future – “build-to”

Crucial distinction
A refinement from IEEE Std 830

- IEEE Std 830-1998 – IEEE Recommended Practice for Software Requirements Specifications:
  - “A requirement specifies an externally visible function or attribute of a system”
    - We can see inputs and the outputs, but not what happens inside

- For any product (SW, HW, total system), the behavioral requirements for that product specify its externally visible behavior
  - as seen by other systems outside

- Specifically, *behavioral attributes*
  - Important concept

- This brings in more detailed viewpoints
- Stay tuned
Agenda

- Motivation
- Terminology
- IEEE view

Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
System requirements and architectures

- There is a close relationship between a system’s requirements and a system’s architecture
  - Not just in the requirements driving the architecture
- In fact, developing an architecture is closely related to developing requirements
  - Hand-in-hand
- Failing to recognize this raises risk of inefficiencies
- Exploiting this relationship opportunities for efficiencies
Architecture

- From the Greek ἀρχιτέκτων
  - chief
  - Workman, builder

- "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." Bass, Clements, Kazman. Software Architecture in Practice (2nd edition). Addison-Wesley 2003

- “The fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.” IEEE-1471
Context of requirements

- All requirements are defined in context of a specific item – like a “black box”
  - E.g., component, module, system, unit, subsystem

- Any specific item may consist of additional internal components
  - Each of which has its own attributes/behaviors/interfaces

- Hence there are multiple levels of requirements based on level of component
  - System level, subsystem level, software configuration item (SCI) level, component level, software unit level,...
  - Requirements from higher levels are allocated to components at lower levels
Component design (its architecture) consists of:
- The requirements for behavior of each constituent component
- The interrelationships between the components

Interaction of components produces the behavior of parent component

Architectural design is the process of defining requirements for constituent components, down to the smallest unit
- AKA allocated requirements
Design

- **The system**
  - Something that provides services according to what we need
  - Services defined by the system’s behavioral attributes
    - The system’s input and output behavior (its stimuli and responses)
  - To build the system, we need to define what is inside of the box

![Diagram of system with stimuli and responses](image-url)
Designing the system
Levels of design

- At each stage, system/subsystem/components are decomposed into constituent parts
- The behavior of the system is determined by the aggregate behavior of its subsystems
  - The behavior of each subsystem is determined by the aggregate behavior of its components
- and so on “down” to the software code and the hardware
Key points

- "The software architecture of a program or computing system is the structure or structures of the system, which comprise:
  - software elements,
  - the externally visible properties of those elements, and
  - the relationships among them."

The behaviors of the elements

The interfaces and interactions among the elements
External versus internal requirements

- Some system components
  - are visible only internally to system
  - are visible both internally and externally to a system
  - have attributes that are visible only externally to system

- System behavioral attributes consist of the collection of behavioral attributes of its components that have external visibility
Why is this important?

- The practice of requirements engineering needs to extend into and be intertwined with the architectural design practices
- Advantages include the capability of an integrated modeling approach
- The rigor applied as a part of requirements engineering assists in clearly establishing the required attributes of the successive components and subcomponents
- The concepts (to be explained in this tutorial) will provide effective support to the design process
Agenda

- Motivation
- Terminology
- IEEE view
- Requirements and architecture/design

Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
Nature of system attributes

- **Terminology recap:**
  - “Requirements” is a term we use to refer to the set of system attributes that customer/users desire for a system to be acquired.
  - “Attributes” is a term we use to refer to the characteristics/conditions/capabilities of a system.

- **We will use “requirements” to denote desired attributes**
  - Once the product is built, the as-built attributes need to be documented in a product specification.
  - And are rarely the same as the initial requirements.

- **There are several different types/categories of system attributes that we deal with**
  - When expressing an acquirers requirements for a system, we must ensure that we cover all types to ensure completeness.
Major categories of system requirements

- **Four major categories**
  - *Behavioral* – address externally-visible behaviors, dynamic properties of the product as seen through the component’s interfaces
    - E.g., “shall turn on fan when temperature is greater than 90° within 2 secs of reaching that temperature zone”
  - *Supportability* – address working with the product as a product – static properties (AKA quality of construction)
    - E.g., maintainability, portability, extensibility
  - *Implementation constraints* – address design and construction attributes, constrain the internals of the component
    - E.g., specific architectural styles, specific algorithms
  - *Programmatic* – address the resources used and other non-system-specific elements used to develop and maintain the system
Types of software system attributes

System Attributes

Behavioral
- Functional
- Temporal
- Capacity
- Resource utilization
- Usability

Supportability
- Maintainability
- Portability
- Extensibility
- Reusability
- Integrity of construction

Implementation
- Delivery Schedule
- Cost

Programmatic

Reliability
- Safety
- Availability
- Integrity of operation
- Protection of information

Performance
- Ease of learning
- Efficient to use
- Easy to remember
- Forgiving

……
Behavioral attributes

System Attributes

- Behavioral
  - Functional
  - Trustworthiness
  - Interface
  - Temporal
  - Capacity
  - Resource utilization
  - Usability
- Supportability
  - Maintainability
  - Portability
  - Extensibility
  - Reusability
  - Integrity of construction
- Implementation
  - Delivery Schedule
  - Cost
  - ...
Behavioral

- Behavioral requirements consist of the product’s externally-visible behaviors, which can be caused by
  - Externally-supplied stimuli
  - Autonomous (self-generated once activated)
  - Mixed

- Stimuli provided by the component’s environment
  - Sometimes chaotic and disorganized
  - Sometimes organized and deliberate
  - Sometimes expected and planned

- Interfaces define what stimuli are “seen” and what responses are produced –
  - no interface → no input or output

- Sometimes unexpected
- Sometimes nefarious
Behavioral attributes

- Deal with the interactions between an item and the world outside
  - As seen through its interfaces
- Express desired externally-visible actions / attributes of the item
- Visibility provided via interfaces which provide the means for
  - Stimulating the item
  - Observing item’s responses
The world outside - the environment

No interface → component sees nothing from the plane

Interfaces
What the component “sees”

External Systems
Responses to the world outside

External Systems

INTERNET

Responses to the world outside.
Interfaces

- The interfaces allow interactions with the world outside and provide the means for receiving and exporting data.
- To define behaviors, must first define the environment(s) in which the component will operate.
  - If only part of the environment is defined, surprises will happen.
- Includes defining input data characteristics:
  - Format
  - Time characteristics:
    - Frequency of arrival
    - Periodicity of arrival
  - Volume
- Pitfall: failure to adequately describe the system’s operating environment will result in awkward moments.
  - “Where did that input come from?”
Interfaces

- Three major types
  - Peer-to-peer
  - User interface
  - Computing infrastructure
Defining interface requirements

- The data items that are input
- The source of the input data – where the data is obtained
- The format of the input data (signature)
- The information content of the input data (semantics)
  - Types of data
  - Data units
  - Data ranges and distributions
  - Precision / accuracy
- The occurrence patterns of the input data, i.e., the operational data environment
  - Arrival timing
  - Data distribution
  - The timing of the input data
    - i.e., the operational data environment
Functional attributes

- Input-output behavior in terms of responses to stimuli - maps data received to data transmitted
  - Output = f(input)

- Two types
  - Simple I/O (stateless) – this input produces this output, e.g.
    - $x \rightarrow \text{Element} \rightarrow x^2$
  
  - State-based – the history of inputs defines the output, e.g.
    - $x_1 \ x_2 \ x_3 \ x_4 \ x_5 \rightarrow \text{Element} \rightarrow (x_1+x_3)^2 / x_5$
Functional attributes

- Simple I/O (stateless) – can be defined using tables, functions,
  \[ f(a, b, c) = y \]
- State-based – can be defined using state machines

Can be complex

\[ \frac{\left( x_1 + x_3 \right)^2}{x_5} \]

\[ \frac{\left( x_1 + x_3 \right)^2}{x_5} \]

\[ \frac{\left( x_1 + x_3 \right)^2}{x_5} \]

\[ x^2 \]

\[ x^2 \]

\[ x^2 \]

\[ x^2 \]

\[ x^2 \]
Attributes of functional behavior (1 of 2)

- Attributes constrain and characterize the behavior
- Some associated within individual functional behaviors
  - Some associated with collective functional behaviors
- Six attributes
  - Inputs - source of the data, via the incoming interface
  - Outputs - the destination of the data, via the outgoing interface
  - Temporal – the timing of the responses - speed, latency, throughput
  - Capacity – the amount of processing that can be performed
  - Resource utilization – the computing resources required
  - Trustworthiness – the degree of dependability
  - Usability – the ease of use by operators
Attributes of functional behavior (2 of 2)

- All of these six must be addressed to ensure requirements are complete
  - Even if they are not “relevant”
  - Often, attributes are not important or not of particular concern interest to the users - when specifying requirements, a positive response to that fact is required

- Failure to evaluate them can lead to forgetting to address them in the system
  - “Oh. I forget to mention that”

- Late surprises are costly, often requiring rework
Temporal

- A performance-related requirement
- Addresses behavior of component regarding time
  - Establishing time characteristics of functional responses
- Such as:
  - Speed – rate at which response events occur
    - e.g., display refreshed screen every 0.5 sec,
  - Latency (delay) – the time between initiation of a function and its completion
    - e.g., time between user hitting key (reception of data) and appearance of key stroke effect on display (response)
    - Not the same as speed
  - Throughput – number of items processed (volume) per unit time
    - e.g., process 10,000 database requests per hour
Capacity

- A performance-related requirement
- Amount of information that can be handled
- Defined in terms of
  - System operation – e.g., 25 simultaneous users
  - System data objects
    - e.g., a minimum of 20,000 employee records
    - e.g., a minimum of 1000 tracks
Resource utilization

- A performance-related requirement
- Limitations on execution resources available to the component
- Defined in terms of hardware and other items that provide resources to allow the system to operate

Examples
- Memory usage – limits to the amount of memory that can be used, in all categories
  - RAM – system can use no more than 100 mb of RAM (volatile memory)
  - Disk – system can use no more than 2 tb of disk
  - Non-volatile – system can use no more than 5 mb of read-only memory
- Processor usage – limits to how many processor resources can be used
  - “no more than 80% CPU utilization”
A word on “performance” requirements

- **Term used to denote two different contexts**
- **Execution performance**
  - Relates to the behavior of the executing code relative to time, space, resources
  - e.g., latency from receipt of input to the time its presence appears on a user screen
- **Functional performance**
  - Relates to the ability of the system to meet operational objectives in terms of its delivery of functional behavior
  - e.g., ability to recognize words within spoken dialog at a specific success rate (“required to recognize 99% of all words spoken …”)
- **Both are behavioral**
  - Functional performance falls into the Trustworthiness category
  - See next slides
Trustworthiness (dependability)

- Degree of confidence in product’s delivery of functions
  - Inherently qualitative – cannot be definitively proven but can be inferred based on evidence and assurance cases

- Types
  - Reliability – probability of operation without failure for a specified time duration under specified operational environment (e.g., 0.001 failures/hr)
  - Availability – proportion of time a system is ready for use over a defined period of time (e.g., 0.9999999 over 1 year)
  - Safety – avoidance of actions that could lead to harm to humans or property
  - Integrity of operation – system features that protects against corruption during operation, unauthorized, intentional or unintentional
  - Protection of information – features that protect against unauthorized disclosure of information
  - Integrity of information – features that protect against unauthorized disclosure of information
Usability

- The ease of system use by an operator
- Two different flavors based interacting agent -- human or other systems
- When applied to system-to-system interfaces
  - Deals with the complexity of the interfaces, their ease of implementation, and their efficiency of operation
- When applied to human operators
  - Deals with the complexity of the interfaces relative to the how operators can operate with them, the ease of learning, and the efficiencies with which operators can exploit the services provided by the system.
- Usability requirements cannot be directly verified
  - Involve inherently subjective behaviors that often have to be observed over time (e.g., via a usability analysis)
Supportability

System Attributes

Behavioral

Functional
Trustworthiness
Interface
Temporal
Capacity
Resource utilization
Usability

Reliability
Safety
Availability
Integrity of operation
Protection of information
Performance
Ease of learning
Efficient to use
Easy to remember
Forgiving

Supportability

Maintainability
Portability
Extensibility
Reusability
Integrity of construction

Implementation

Delivery Schedule
Cost

Programmatic

......
Supportability requirements

- AKA Quality of construction requirements
- Attributes of the product itself and its construction
- Deals with how product can be handled, not its operation
- Inherently qualitative – cannot definitively verify
- Often not directly observable or measurable
  - Measures exist that provide insight into these qualities,
    - Help to infer level of quality based on related quantitative system attributes
  - But direct measures do not in general exist
- Examples:
  - Portability – ease with which component can be ported from one platform to another
  - Maintainability – ease with which product can be fixed when defects are discovered
  - Extensibility – ease with which product can be enhanced with new
Implementation

System Attributes

Behavioral
- Functional
- Trustworthiness
- Interface
- Temporal
- Capacity
- Resource utilization
- Usability

Reliability
- Safety
- Availability
- Integrity of operation
- Protection of information

Performance
- Ease of learning
- Efficient to use
- Easy to remember
- Forgiving
- ......

Supportability
- Maintainability
- Portability
- Extensibility
- Reusability
- Integrity of construction

Implementation
- Delivery Schedule
- Cost
- ......

Programmatic

NDIA System Engineering Conference 2012
Implementation requirements (1 of 2)

- Restrictions placed on developers that limit design space and development process (AKA implementation constraints, design constraints)
  - e.g., use of specific software components
  - e.g., imposition of specific algorithms
  - e.g., customer-mandated architectures
  - e.g., imposition of certain development techniques

- Two general types:
  - Product constraints – restrictions on the product construction
    - Design constraints – restrictions on design / architecture
    - Implementation constraints – restrictions on coding or construction
  - Process constraints – restrictions on how the product is built, constraints on the processes to be used
Implementation requirements (2 of 2)

- An implementation constraint to a system might be a requirement to a SW component within that system
- While these are required characteristics of development effort, they are not characteristics of the product’s behavior
  - But will likely affect behavior
- Examples
  - Use of specific software components
  - Imposition of specific algorithms
    - But sometimes algorithms can be used to define functionality
  - Required use of specific designs
    - Technical architectures
    - Certain internal standards (VME at system level)
  - Imposition of specific coding styles

Programmatic

System Attributes

Behavioral

- Functional
- Trustworthiness
- Interface
- Temporal
- Capacity
- Resource utilization
- Usability

Supportability

- Maintainability
- Portability
- Extensibility
- Reusability
- Integrity of construction

Implementation

- Delivery Schedule
- Cost
- ...

Reliability
- Safety
- Availability
- Integrity of operation
- Protection of information

Performance
- Ease of learning
- Efficient to use
- Easy to remember
- Forgiving
- …..
Programmatic (contractual) requirements

- Terms and conditions imposed as a part of a contract exclusive of behavioral requirements
  - aka contractual requirements
- Address project environment aspects of product development
- Examples
  - Cost
  - Schedule
  - Organizational structure
- While these are required characteristics of development effort, they are not characteristics of the product
  - But can directly affect ability to achieve product characteristics
    - (not enough time, not enough budget)
Interrelationships

- The categories and subcategories are interrelated
- e.g., Budget and schedule (programmatic) affect what functions can be implemented (behavioral)
  - With limited budget, some functional requirements might not be possible
- e.g., Portability (supportability) affects internal design (implementation)
  - Need to be able to rehost might negate some implementation constraints
    - Required use of a COTS product which does not run on VxWorks contradicts a requirement to be able to rehost system to a VxWorks environment
- Performance (behavioral) affects design (implementation)
  - Tight latency requirements might make a required infrastructure unusable
Quality requirements

- What about requirements that address the quality of the system
  - The quality of a system is directly related to how well it meets its requirements

- But you might say: “Some systems meet all of their requirements and are of poor quality”

- Response: “The requirements for that system are deficient and incomplete, ....”

- Better assertion;
  - The quality of a system is related to
    - The quality of its requirements
    - The degree of adherence of the system to its requirements
    - The suitability of the system to what the users need

- We sometimes refer to the -ilities
The “-ilities”

- Commonly, “requirements” that end with “-ility” are classified as quality requirements
  - Reliability, maintainability, portability, extensibility, availability, marketability, schedulability, authenticability, …. yada yada yada

- However → A lexical similarity does not mean that there is a semantic similarity
  - Just because a word ends with -ility does not mean that it has a similar effect on a system’s requirements

- Some deal with behavior, some deal with construction, some deal with manageability

- Should not be classified and handled together and alike
## Sampling of -ilities

<table>
<thead>
<tr>
<th>Ability</th>
<th>Abatability</th>
<th>Abbreviability</th>
<th>Abdicability</th>
<th>Abdominability</th>
<th>Aberrationability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
<tr>
<td>Ability</td>
<td>Abatability</td>
<td>Abbreviability</td>
<td>Abdicability</td>
<td>Abdominability</td>
<td>Aberrationability</td>
</tr>
</tbody>
</table>
Interrelationships

programmatic

behavioral

implementation

manageability
Agenda

- Motivation
- Terminology
- IEEE view
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
IEEE Qualities of requirements (1 of 2)

- IEEE Std 830-1993* defines nine qualities for requirements specifications
  - Complete – All external behaviors are defined
  - Unambiguous – Every requirement has one and only one interpretation
  - Correct – Every requirement stated is one that software shall meet
  - Consistent – No subset of requirements conflict with each other
  - Verifiable – A cost-effective finite process exists to show that each requirement has been successfully implemented
  - Modifiable – SRS structure and style are such that any changes to requirements can be made easily, completely, and consistently while retaining structure

* IEEE Recommended Practice for Software Requirements Specifications
IEEE Qualities of requirements (2 of 2)

- IEEE Std 830-1993 qualities of requirements (cont’d)
  - Traceable – Origin of each requirement is clear, structure facilitates referencing each requirement within lower-level documentation
  - Ranked for importance – Each requirement rated for criticality to system, based on negative impact should requirement not be implemented
  - Ranked for stability – Each requirement rated for likelihood to change, based on changing expectations or level of uncertainty in its description

- These qualities are associated with how the requirements are defined and documented
- Failure to assess requirements against these criteria raises risk of subsequent problems
Requirements maturity

- **Maturity level** – a measure of how well the requirement meets key qualities

- **Requirements generally mature over time**
  - As we learn more about what we want the system to do
  - Important to understand the maturation path as a system is developed

- **Key factors:**
  - **Correctness**
    - Does the requirement express a product attribute that is needed?
    - Are we certain, or are we guessing / approximating? Is it accurate?
  - **Unambiguity (specificity)**
    - Does the requirement express a precise attribute that cannot be interpreted in more than one way?
    - Would a programmer know exactly what code to write to satisfy the requirement?
Correctness / accuracy

- Not always possible to determine up-front
- Independent of ambiguity
- Criterion: does the requirement express an attribute that needed for the product?

- E.g.
  - “The elevator shall stop smoothly at each floor”
  - Clearly ambiguous and vague, but certainly desirable and correct

- E.g. 2
  - “Upon receipt of signal from the temperature sensor that indicates the temperature is greater than 110° C, the system shall initiate a shutdown of the burner unit within 0.1 sec.”
  - Precise and unambiguous, but it turns out that 140° C is the critical temperature and 1 sec is the shutdown tolerance.
  - Hence, incorrect.
Maturity path

- Requirements generally start at lower left and hopefully progress towards upper right of maturity graph.
- Far too often the ambiguity is decided by the programmers while writing code, resulting in less-than-optimal interpretation of requirement.
Derived requirements

- These are additional requirements that are created to
  - Support the baseline requirement(s) by providing necessary detail
  - Expand on the baseline requirements with additional behaviors

- Derived requirements provide a mechanism to achieve maturity

- Note that they refer to the same component as the baseline requirements
  - They are not allocated to lower-levels of the architecture
Concreteness property for requirements

- **Level of detail may vary**
  - Ranges from vague and general to very specific (& directly testable)
    - The system shall require authentication for all users to gain access (vague)
    - Users shall enter their username followed by their password to gain access to the system (more concrete)

- **At some point, the detail will be defined, either**
  - informally by the programmer and perhaps neither documented nor reviewed/approved, or
  - Formally and documented prior to and during implementation

- **Important note - if the detail addresses externally visible behavior, it will never be design**
  - Design is internal to the component
  - If it addresses externally visible behavior, it is always a “requirement”
Requirements tracing

- Each and every requirement defined at each level must be
  - Based on a requirement at next higher-level (else it has no reason to exist)
  - Supported by a requirement/design feature at next lower-level (else it will not be achieved)
  - Hence all requirements are derived

- Sometimes mapping is simple (1–1 or 1–few)
- Sometimes mapping is complex and/or indirect (1–to–many or many–to–1)
- Use of automated tools recommended
Agenda

- Motivation
- Terminology
- IEEE view
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements

Creating requirements
- Verifying requirements
- Challenges / pitfalls
Requirements engineering

- A discipline all its own
- Must consider all aspects of the system and its environment, such as:
  - Feasibility to effectively and efficiently build system
    - Requires knowledge of hardware, software, and people
  - Fitness for purpose
    - Requires knowledge of operational domain
  - Adoption of potentially reusable elements (including upgrades to existing systems)
    - Requires knowledge of precedent assets and how their specs will affect requirements
  - Relationship to other interacting elements in the environment
    - Requires knowledge of existing and emerging systems
  - Ability to elicit, listen, and interpret
In particular...

- Defining requirements must involve all affiliated subject-matter experts (SMEs) – risk otherwise

- Cannot allocate to and define requirements for system elements unless you consider what the engineers who will build these elements need
  - Hardware, software, operator interactions
  - Incl. domain experts (radar, thermal, legal, sonar, guidance, accountants, …)

- Hence, requirements engineers must be involved from the beginning
  - Once requirements allocated to SW components, SW requirements analysis (SRA) needed to derive specific SW requirements and place into form suitable for implementation
Requirements analysis techniques

- Different techniques/processes are used
  - ad hoc techniques
  - Functional techniques
  - Object-oriented techniques

- New processes arrive every day
  - Agile Unified Process
  - Extreme Programming
  - Cleanroom Software Engineering

- Many tools exist
  - DOORS, Analyst Pro, Rational Rose

- All should produce the same result – a description of behavior of the system and all other desired attributes

- Important to select technique to be appropriate to system
A note on Use cases and requirements

- Booch, Rumbaugh, Jacobson. The Unified Modeling Language Users Guide:
  - “A use case specifies the behavior of a system or part of a system and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor”
  - “A use case describes what a system ... does but it does not specify how it does it.”
- Can be misleading
- Provide some information about behavioral requirements but not at level of detail sufficient for development
- Use cases are important to requirements definition
  - Especially in the form of scenarios
  - Help to describe how system will be used
  - How various capabilities and functions will be used together (“Threads”, scenarios)
Scenarios to functional attributes

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Scenarios</th>
<th>Scenarios</th>
<th>Scenarios</th>
<th>Scenarios</th>
<th>Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use cases</td>
<td>Use cases</td>
<td>Use cases</td>
<td>Use cases</td>
<td>Use cases</td>
<td>Use cases</td>
</tr>
<tr>
<td>Threads</td>
<td>Threads</td>
<td>Threads</td>
<td>Threads</td>
<td>Threads</td>
<td>Threads</td>
</tr>
</tbody>
</table>

Functional attributes
Key Information needed for requirements

- Need – why is the system needed? What role will the system play?
- Operational environment – what is the environment in which the system will operate?
- Strategy for satisfying need – given the environment, how is it expected that the system will support the needs?
- System capabilities to implement strategy – what does the system need to do to satisfy need?
- Usage profile – how will the system be used?
- Technical requirements – what specific behaviors will the system provide to deliver desired capabilities?
Key factors in defining requirements

Need

Operational environment

Strategy

Capabilities

Usage/scenarios

Technical requirements

Crucial steps

- Understand user needs via active requirements elicitation
- Characterize the operational environment
- Define a strategy for fulfilling user needs
- If strategy involves building a system:
  - Derive necessary system capabilities based on needs and strategy
    - Involve users, develop scenarios that cover needs
  - Define scenarios depicting how users will employ the system to fulfill needs
    - Include low likelihood as well as common situations
  - Use scenarios to derive specific required attributes
    - Incl behavioral, implementation, programmatic, and manageability
  - Apply same techniques throughout system design
  - Continue to revisit required attributes as understanding matures
  - Document attributes throughout, until completed

**Requirements “life cycle”**

- Important to understand how we get to the requirements

```
Mission needs

Overall system concept

Operational Requirements

System Requirements

System Design

Software development

Why system is needed, what the overall problem is

Role of the system(s) in solving the problem

Capabilities system needs to solve the problem

Functions necessary to achieve capabilities

Overall design of system, including CIs that are used to build system
```
Each step involves deciding between alternative approaches
Continual refinement of system behavior
Decisions based on many constraints

- Schedule/budget, staffing, risk, operator preferences, alignment to mission, reused components, ...

Requirements selected represent one possible approach to delivering capabilities, not unique.
As the requirements mature through their life cycle, they cross the spectrum from

- Mission and operational context
- To
- Specific, detailed behaviors

<table>
<thead>
<tr>
<th>Mission And Roles</th>
<th>Capabilities</th>
<th>General Requirements</th>
<th>Detailed Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>The kinds of things that the system needs to do in order to support the mission</td>
<td>The behaviors that the system needs to implement in order to deliver the capabilities</td>
<td>Very specific definition of input/output behaviors</td>
<td></td>
</tr>
</tbody>
</table>
“Capabilities”

- “The system shall have the capability of reading the room temperature sensor and adjusting the heat in the room to correspond to the desired setting”
- Just because it has the capability does not mean that it has to
  - Most people have the capability of adjusting the thermostat if the room is too cold. They may just decide not to do so.
- The system meets the requirement if:
  - It adjusts the heat 2 days later
  - It adjusts the heat to within $\pm 10^\circ$
  - It reads your mind and guesses what the desired setting is
- Defining capabilities useful in early stages
  - On way to defining more specific required attributes
  - Need to mature these to be more specific as development proceeds
Agenda

- Motivation
- Terminology
- IEEE view
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements
- Challenges / pitfalls
Verification / validation

- Systems are verified to determine overall acceptability
  - Is the system suitable for operation?
    - Despite any identified shortcomings
- While building a system, continual tradeoffs need to be performed to reconcile actual attributes to desired attributes
  - Systems rarely implement every requirement
- Trades to be based on acceptability framework
- All requirements to be evaluated against
  - How realized in developed product (complete, partial)
  - Priority (importance) of each requirement
  - Overall fitness for use
- Best practice is to create a requirements verification matrix (RVM)
  - Where each requirement is mapped to a verification approach
Acceptability framework

System Attributes

Behavioral
- Functional
- Trustworthiness
- Interface
- Temporal
- Capacity
- Resource utilization
- Usability

Supportability
- Maintainability
- Portability
- Extensibility
- Reusability
- Integrity of construction

Implementation
- Supportability
- Delivery Schedule
- Cost
- ....

Programmatic
- Reliability
- Safety
- Availability
- Integrity of operation
- Protection of information

Performance
- Ease of learning
- Efficient to use
- Easy to remember
- Forgiving
- ....

Fitness for purpose
Requirements and verification

- Each and every requirement needs to be verified
  - That is, need to be able to construct a valid argument that the requirement has been satisfied by the as-built system
  - Argument needs to be supported with sufficient objective evidence

- A requirement is verifiable if such an argument can be constructed

- There are multiple techniques to construct these arguments

- Each type of requirement may require the application of multiple techniques to provide a full, sufficient argument

- When defined, each requirement must be correlated to the approach(s) to be used to verify that requirement

- Note that ALL requirements need to be verified
  - Even if not behavioral
We define five types of verification techniques (note that this list is not necessarily the same as used elsewhere):

- Test
- Product analysis
- Inspection
- Demonstration
- Process analysis
Verifying requirements – test

- **With test, we execute the product, challenge with stimuli, and observe behavior (responses)**
  - Collect the responses
  - Compare responses to desired responses (oracle) to determine degree of adherence
  - Desired responses specified by the requirement statement

- **Execution environment may include actual operational environment of product**
  - May also include simulations of other systems in the environment
Categories of test

- Two types of test based on the ability to determine conformance to requirements:
  - **Definitive**
    - Results are quantitative
    - Can be compared directly to the requirements
    - Results can be stated as pass/fail
  - **Analytic**
    - For requirements that cannot be definitively verified
      - Mathematical and other forms of analysis must be used to make an argument for compliance.
    - Test results from one or more tests may support an argument for either pass or fail, but do not provide an absolute determination of conformance.
    - Such arguments serve to establish the levels of trust that can be placed on the system’s performance
Verifying requirements – product analysis

- **Product is not executed (not tested)**

- **System attributes evaluated analytically, often supported mathematically**
  - *e.g.*, RMA (Rate Monotonic Analysis)
  - *e.g.*, architecture analysis

- **Results used to create arguments of compliance for those requirements that are inherently non-deterministic**
  - dependability
  - to establish levels of trust
Verifying requirements – demonstration

- Product is manipulated to demonstrate that it satisfies a quality of construction requirement
- Such requirements express certain attributes of the product but not how these attributes are achieved
- e.g., portability
  - A portability requirement states a desire to be able to rehost a product to a different computational environment with minimal effort and cost
  - Usually achieved by imposing certain design constraints (modular architecture, low coupling, high cohesion)
    - Perhaps separately stated as a design constraint
  - To verify that the product is portable, a demonstration of rehosting the product from one computer to another may be performed.
Verifying requirements – inspection

- Visual examination of product, its documentation, and other associated artifacts to verify conformance to requirements
- Often used in conjunction with other techniques to complete argument
- Particularly useful for verifying adherence to design/implementation constraint requirements
  - *e.g.*, a software component may be inspected to verify that makes no operating calls other than to a POSIX-standard interface
Verifying requirements – process analysis

- Analysis of the techniques and processes used by developers to determine if they are adhering to any required project standards and plans
  - Particularly those that correlate to stakeholder requirements for specific processes
  - May involve examination of the various intermediate and final products as well as programmatic artifacts and records
  - E.g., adherence to safety standards (DO 178B/C, 882C)
## Verification approaches

<table>
<thead>
<tr>
<th>Type of Requirement</th>
<th>Verification Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Definitive Testing</td>
</tr>
<tr>
<td>Behavioral</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>✓</td>
</tr>
<tr>
<td>Interface</td>
<td>✓</td>
</tr>
<tr>
<td>Temporal</td>
<td>✓</td>
</tr>
<tr>
<td>Capacity</td>
<td>✓</td>
</tr>
<tr>
<td>Resource utilization</td>
<td>✓</td>
</tr>
<tr>
<td>Trustworthiness</td>
<td>✓</td>
</tr>
<tr>
<td>Usability</td>
<td>✓</td>
</tr>
<tr>
<td>Supportability</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Constraints</td>
<td></td>
</tr>
<tr>
<td>Product constraint</td>
<td></td>
</tr>
<tr>
<td>Process constraint</td>
<td></td>
</tr>
</tbody>
</table>
Limitations to verification

- Vague requirements cannot be effectively verified
  - Because valid responses not clearly defined
- Good, complete, and unambiguous requirements inherently contain the information necessary for verification
- Explicit, concrete, and detailed requirements generally can be directly verified
  - All input and output details fully defined
- A requirement could be very explicit and concrete yet not be directly verifiable
  - The system shall be available for 0.9999999 over a one year operating duration
Agenda

- Motivation
- Terminology
- IEEE view
- Requirements and architecture/design
- Types of requirements
- Qualities of requirements
- Creating requirements
- Verifying requirements

Challenges / pitfalls
Some typical challenges and pitfalls

- Fear of detail
- Human Interfaces (HMI)
- Failure to manage volatile and late-defined requirements
- Failure to recognize unknown “physics”
- Overspecified / over-constrained / unbounded
Levels of detail for requirements (1 of 3)

- When developing a system, requirements typically start at general level
  - List of capabilities – what users want to be able to do
    - “The product shall allow users to perform word processing”
  - Sometimes, detailed behaviors known up-front
    - e.g., preexisting external interfaces
    - e.g., required screen formats and display icons

- As development proceeds, general requirements are refined (should be) to become specific behaviors
  - More detail added as more is understood
  - E.g., “Pressing Ctrl and I at the same time results in the selected text being converted to an italics font within 0.5 sec”
Levels of detail for requirements (2 of 3)

- Regardless of level and amount of detail, as long as descriptions address external behavior, they are “requirements” and not design
  - Including GUI screens and formats, interface formats, and protocols
    - (Note that “interface design” is actually part of requirements definition)
  - Remember, practice tends to use the word “requirement” to refer to externally-visible behavior
    - Where design is internal to the item

- Why is this important?
  - Externally-visible behaviors tend to create dependencies on external entities (systems, people)
  - Design features tend to be created based on design tradeoffs made by the developers without dependencies on external systems
  - Although externally-visible, the latter are not necessarily the cause of all requirements
Levels of detail for requirements (3 of 3)

- When product complete, all system attributes implemented and should be known in full detail, to support
  - Training and operator manuals
  - Maintenance manuals
  - Reuse of system and its components

- Failing to recognize this results in
  - Immature requirements being built into software
  - Disconnects with external systems with dependencies

Don’t fear detail - no matter how much detail is provided, “requirements” are never design (for a specific component)
Human Machine Interfaces (1 of 2)

- Human Machine Interface (HCI, MMI, HIS, ...)
  - Refers to interface between the system and humans
  - How information is conveyed between humans and the system
  - How control is achieved by operators over system

- HMIs are externally visible – hence are part of system/software requirements
  - Failure to treat as requirements can lead to problems

- If HMI is awkward and ineffective, system will be a failure
  - If system cannot be effectively and efficiently used, its role will be diminished
  - Operators will ignore it
Human Machine Interfaces (2 of 2)

- **Recommendations:**
  - Where applicable, human performance must be a part of system performance
    - Define performance reqs which include human-in-the-loop
  - Involve users early with design of user interfaces
    - Exploit dynamic prototypes, avoid prolonged use of static displays
  - Perform usability analysis to determine how well users can learn and interact with system
    - Measure overall performance
  - Obtain formal agreement on HMI once defined as part of requirements
    - Avoids second-guessing during system acceptance
  - Defer some HMI features as run-time configuration option
    - If appropriate, to avoid code rework
  - Rely on standards and standard tools to help create common view
Volatile and late-defined requirements

- Requirements always change
  - Some don’t change, but are defined late
- Not necessarily bad but careful management necessary to avoid
  - Expensive rework (and cost and schedule impact)
  - Compromises to functionality

- Crucial to associate levels of risk to levels of change
  - Some changes are low-risk
  - Other may be high risk
  - Related to amount of rework required

- Developers better able to design defensively if they know
  - Which requirements are likely to change
  - Degree of change that could be expected
Volatile and late-defined requirements
(2 of 4)

- Depends on attributes of the requirement and its linkage to design
  - Some can be defined early or late
  - Some must be defined early
  - Some should be defined later

- Important attributes (i.e., how to decide...)
  - If level of understanding of desired behavior is low (exact behaviors not well understood or unknown) – delay in definition may reduce risk
    - If defined and frozen early, later changes may impact design and cause rework
  - If high likelihood that requirement will change – delay in definition may reduce risk
    - Avoids rework due to late changes
Volatile and late-defined requirements
(3 of 4)

- Important attributes (cont’d)
  - If a requirement has high or complex external component dependencies – early resolution may reduce risk
    - Late changes likely to affect external systems/components
  - If a requirement has strong internal design dependencies – early resolution may reduce risk
    - Late changes may force extensive rework due to design dependencies

<table>
<thead>
<tr>
<th></th>
<th>Early definition</th>
<th>Late definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of understanding of desired behavior</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Likelihood that requirement will change</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>External component dependencies</td>
<td>complex</td>
<td>simple</td>
</tr>
<tr>
<td>Internal design dependencies</td>
<td>strong</td>
<td>weak</td>
</tr>
</tbody>
</table>
Recommendations to address requirements volatility:

- Define requirements with priorities and likelihood to change
  - Allows designers to insulate themselves from unexpected change
- Ensure design accommodates expected changes
- Where possible, allow run-time reconfiguration to allow changing behavior without changing requirements
  - e.g., screen color options
- Assess dependencies between requirements and design
  - Some requirements deeply affect design globally
  - Others have limited design impact (GUI formats)
- Ensure inter-requirements dependencies are well understood
- Define and monitor requirements stability with metrics
  - Track immature requirements, undefined requirements, and changing requirements

Different types of changes have different impacts
Types of requirements changes

- **9 types of changes**
  - Addition – new requirement statement added to existing set, increasing functionality
    - Potentially increases work scope, can cause rework
  - Extension – existing requirement statement is changed to extend functionality
    - Potentially increases work scope, can cause rework, not as risky as addition
  - Correction - changes existing requirement statement to correct a defect in its description (thereby modifying the functionality described).
    - Beneficial but may contribute to a change in scope
  - Deletion – remove an existing requirement statement with corresponding removal of functionality
    - Might not require extensive changes, depending on work already performed
  - Deferred – defer an existing requirement from requirements
9 types of changes (cont’d)

- Clarification – change an existing requirement statement to clarify functionality
  - Common, should be expected and encouraged
  - Includes expansion of description, refinement of detail, and removal of ambiguity
- Split – an existing requirement statement is split into multiple statements
  - No change in behavior, little risk
- Merge – two or more requirement statements are merged into a single statement, without changing intended functionality
  - No change in behavior, little risk
- Editorial - an existing requirement statement is modified by correcting defects in language or notation, but with no change to functionality
  - No change in behavior, little risk
  - Common, should be expected and encouraged

Typical pattern of changes

![Diagram showing typical pattern of changes with columns for Added, Extended, Corrected, Clarified, Editorial, Unchanged, Deferred, Deleted, and Total. The diagram includes dates from Feb-10 to Jan-11 with changes ranging from -11 to 38.](image-url)
Unknown “physics” (1 of 2)

- Sometimes we don’t know what the behavior of the software should be
  - Due to lack of detailed/accurate knowledge of real world (typical for embedded systems)
- Results in changing or late-defined requirements
- Need to use software product to explore physics and mature product based on discoveries (like a prototype)
- Frequently, discoveries are at fine grain level (adjustments to constants, changes to algorithms)
- Requirements statements must explicitly recognize need for experimentation
  - Design (and process) must allow for experimentation / exploration
  - Need to incorporate provisions for data extraction and collection
Mitigation strategies include:

- Iterative development of system, and use of iterations to probe and explore the environment
- Use of executable models and prototypes
- Use of simulations to depict external environment
- Use of data logging functions to collect relevant data
Sometimes requirements are too ambitious, too restrictive, or too general

- Too ambitious – results in gold-plating
  - Unneeded capabilities created, unattainable functions defined
- Too restrictive – results in narrow, point solutions
  - System rapidly becomes outdated when mission changes
  - Sometimes you just don’t know up front the exact details (“is 5 sec OK or do I need 2 sec response time?”)
- Too general – results in inefficient system that does everything but not well

Result is wasted resources
Recommendations

- General but important
  - Be clear about what you know and what you don’t know
    - Don’t feel you have to define everything up front
  - Be willing to start general and incrementally refine
  - Focus on prioritization of requirements
    - Some are always more important than others
  - Ensure what is needed is emphasized
    - Avoid “gold plating” – advice often provided and often ignored
      - Do you really need a 0.1 sec response time?
  - Build system in a series of increments
    - Incrementally commit to requirements and apply early lessons-learned
End

- Any questions?....