

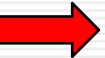
**8th Annual
NDIA Systems Conference
October, 2008
San Diego, California**

Correlation of Types of Requirements to Verification Methods

**Dr. William G. Bail
The MITRE Corporation**

wbail@mitre.org

Agenda

- 
- ◆ Introduction
 - ◆ Types of requirements
 - ◆ Verification techniques
 - ◆ Selecting techniques for requirements
 - ◆ Examples
 - ◆ Summary

Introduction

- ◆ When developing systems, it is a good idea to be able to show the customer that the system works
 - » Especially if you want to get paid
- ◆ Involves demonstration of compliance
 - » To all requirements, individually or in batches
 - » To the entire system, in operational environment with real-life operational scenarios
- ◆ Overall system “quality” needs to fit with customer’s range of acceptability, recognizing that trade-offs are usually made
- ◆ Need to construct a valid argument that system satisfies customer’s requirements, supported with sufficient objective evidence
 - » A requirement is verifiable if such an argument can be constructed
- ◆ This presentation examines some techniques for this proof

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 and style.

* IEEE Recommended Practice for
Software Requirements Specifications

Qualities of requirements (2 of 2)

- ◆ IEEE Std 830-1993 qualities of requirements (cont'd)
 - » *Traceable* – Origin of each requirement is clear, and 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

Agenda

- ◆ Introduction
- ◆ Types of requirements
- ◆ Verification techniques
- ◆ Selecting techniques for requirements
- ◆ Examples
- ◆ Summary



Requirement by any other name...

- ◆ We use many different words to refer to what we want to see in a system

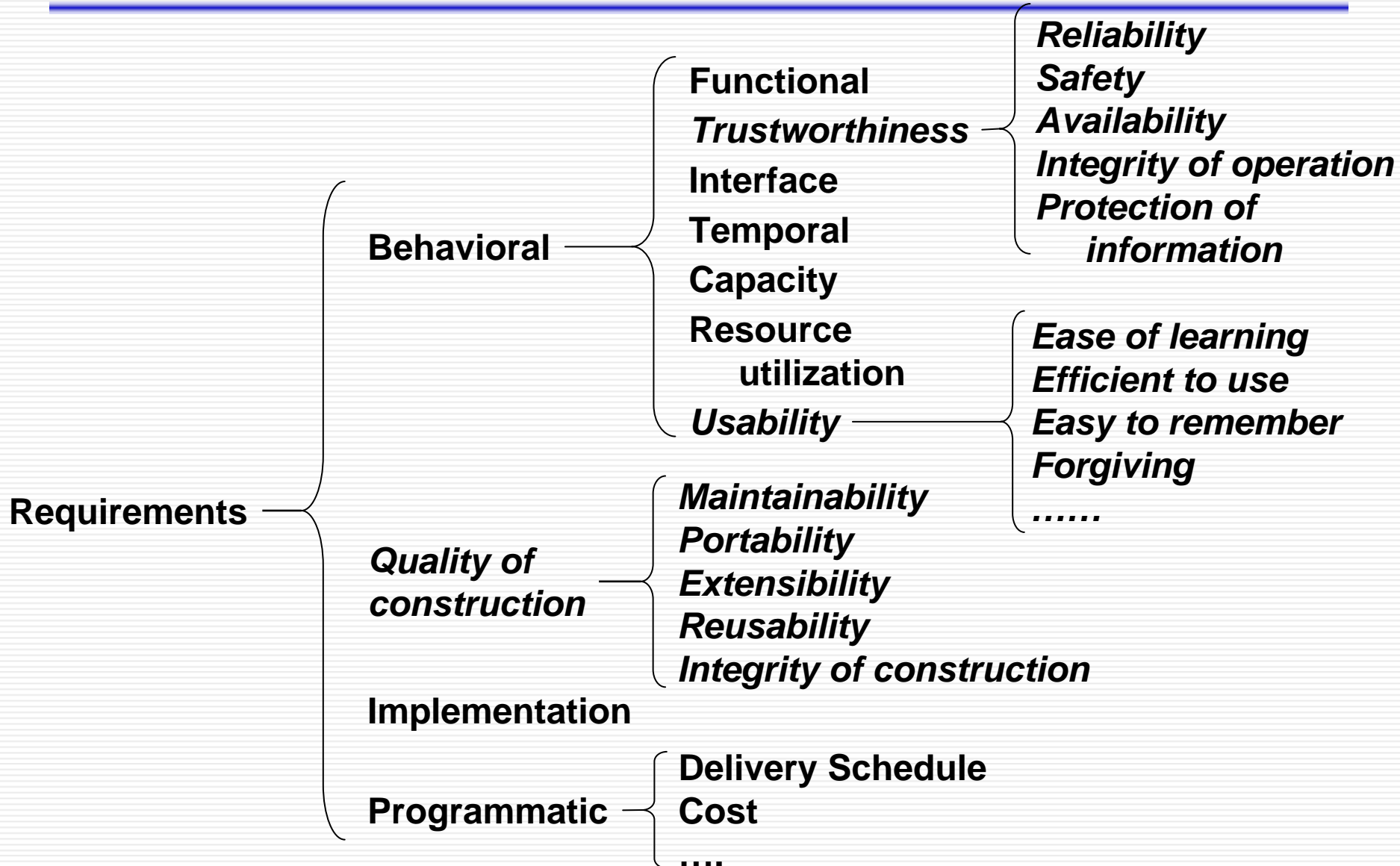


- ◆ They all describe some desired attribute of the to-be-built system
- ◆ When developing a system for a customer, we need to prove that the system has the customer's desired attributes
- ◆ We have various verification techniques to provide this proof

Differences among requirements

- ◆ There are many different types of requirements
 - » Each type has different verification techniques that are suitable
- ◆ Planning for verification starts with defining the requirements
 - » Important to define requirements such that they can be verified
 - » A key IEEE quality attribute
- ◆ As requirements mature and acquire detail, more detail about how to verify them can be added
- ◆ Important to map requirements to the feasible verification techniques early
 - » And mature these as development proceeds
- ◆ Good, complete, and unambiguous requirements inherently contain the information necessary for verification

Types of requirements

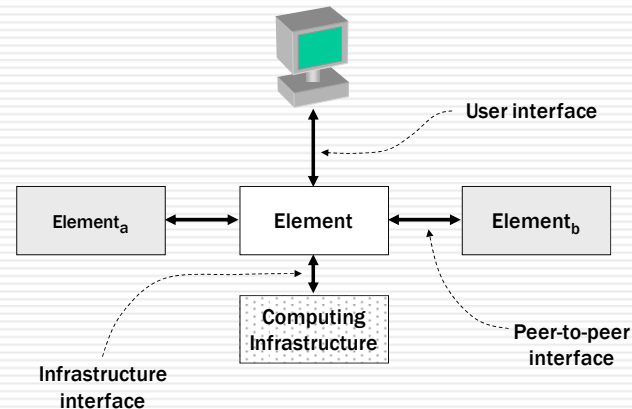


Behavioral requirements *(page 1 of 5)*

- ◆ Those that express externally-visible actions / attributes / behaviors of the entity (component, subsystem, system, unit,...)
 - » Defined by *functional requirements / functional specifications*
- ◆ Verifiable by observing externally-visible responses from externally-applied stimuli
 - » (Potentially) measurable by testing
- ◆ Seven types
 - » Functional » Resource utilization
 - » Interface » Trustworthiness
 - » Temporal » Usability
 - » Capacity

Behavioral requirements *(page 2 of 5)*

- ◆ *Functional* - Input-output behavior in terms of responses to stimuli
 - > Simple I/O (stateless) – this input produces this output
 - > State-based – the history of inputs defines the output
- ◆ *Interface* - characteristics of component's interfaces
 - > Peer-to-peer
 - > User interface
 - > Computing infrastructure



Behavioral requirements *(page 3 of 5)*

- ◆ *Temporal* - establishing time characteristics of behaviors
 - » *Speed* – rate at which events occur
 - » *Latency* – aka delay – the time between initiation of a function and its completion
 - » *Throughput* – number of items processed (volume) per unit time
- ◆ *Capacity* - amount of information that can be handled
 - » System operation – e.g., 25 simultaneous users
 - » System data objects - e.g., a minimum of 20,000 employee records
- ◆ *Resource utilization* - limitations on resources available
 - » Defined in terms of hardware and other items that provide resources to allow the system to operate
 - » e.g., memory usage (RAM, disk, flash,...), processor usage, communication line usage

Behavioral requirements *(page 4 of 5)*

- ◆ *Trustworthiness (dependability)* - degree of confidence in product's delivery of functions
 - » Inherently qualitative – cannot be definitively proven but can be inferred based on evidence
 - » 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* – features that protect against actions that could lead to harm to humans or property
 - > *Integrity of operation* – system features that protects against corruption during operation
 - > *Protection of information – (confidentiality)* – features that protect against unauthorized disclosure of information

Behavioral requirements *(page 5 of 5)*

- ◆ *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)

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 quantitative system attributes
 - » Direct measures generally do not 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 functionality

Implementation requirements *(page 1 of 2)*

- ◆ Restrictions placed on developers that limit design space and 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., Joint Technical Architecture)
 - » e.g., imposition of certain development techniques
- ◆ Two general types:
 - » *Product constraints* – restrictions on the product construction
 - > *Design constraints* – restrictions on design styles that can be used
 - > *Implementation constraints* – restrictions on coding or construction
 - » *Process constraints* – restrictions on how the product is built

Implementation requirements *(page 2 of 2)*

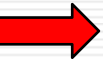
- ◆ An implementation constraint to a system may be a requirement to a SW component within that system
- ◆ While these are required characteristics of development effort, they are not characteristics of the product's behavior
 - » But will likely affect behavior
- ◆ Examples
 - » Use of specific software components
 - » Imposition of specific algorithms
 - » Required use of specific designs (e.g., open systems)
 - > Technical architectures
 - » Imposition of specific coding styles
 - » Required application of specific techniques (e.g., RMA)
 - » Required application of specific unit test coverage criteria

Programmatic requirements

- ◆ Terms and conditions imposed as a part of a contract exclusive of behavioral requirements
- ◆ Address development aspects of product
- ◆ Examples
 - » Costs
 - » Schedules
 - » Organizational structures
 - » Key people
 - » Locations
- ◆ While these are required characteristics of development effort, they are not characteristics of the product
 - » But they can directly affect the ability to achieve product characteristics (not enough time, not enough budget)

Agenda

- ◆ Introduction
- ◆ Types of requirements
- ◆ Verification techniques
- ◆ Selecting techniques for requirements
- ◆ Examples
- ◆ Summary

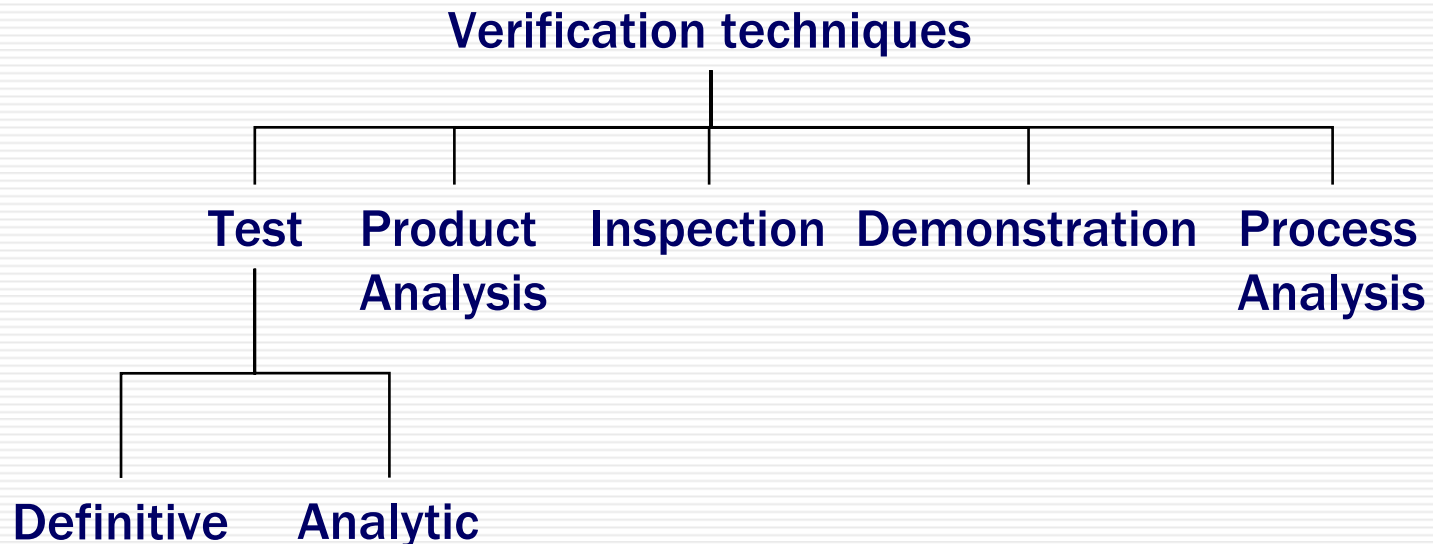


Requirements and verification

- ◆ 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

Verification techniques

- ◆ We define five types of verification techniques
 - » 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 (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

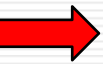
- ◆ Visual examination of product, its documentation, and other associated artifacts to verify conformance to requirements
- ◆ Often used in conjunction with other techniques to complete argument
- ◆ Particularly useful for verifying adherence to design/implementation constraint requirements
 - » e.g., 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
 - » May involve examination of the various intermediate and final products as well as programmatic artifacts and records.

Agenda

- ◆ Introduction
- ◆ Types of requirements
- ◆ Verification techniques
- ◆ Selecting techniques for requirements
- ◆ Examples
- ◆ Wrap-up

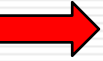


Verification approaches

Type of Requirement	Verification Approach					
	Definitive Testing	Analytic Testing	Analysis	Demonstration	Inspection	Process Analysis
Behavioral						
Functional	√	√	√		√	
Interface	√				√	
Temporal	√	√	√		√	
Capacity	√	√	√		√	
Resource utilization	√	√	√		√	
Trustworthiness		√	√	√	√	
Usability		√	√			
Quality of construction				√	√	√
Implementation Constraints						
Product constraint					√	
Process constraint					√	√

Agenda

- ◆ Introduction
- ◆ Types of requirements
- ◆ Verification techniques
- ◆ Selecting techniques for requirements
- ◆ Examples
- ◆ Wrap-up



Example 1 - Reliability

- ◆ Requirement – “The system shall have a reliability of 60 days MTBF”
- ◆ Cannot verify definitively
- ◆ A test can suggest failure to comply but not compliance
- ◆ Techniques to be applied:
 - » Analytic testing – to observe failure rates
 - » Inspection - to verify built-in fault tolerance
 - » Analysis – to examine failure modes and their effects
- ◆ Steps for creating argument of compliance
 - » Define appropriate operational scenarios, agreed to by customer
 - » Define analysis technique for predicting reliability based on testing, including confidence level

Example 2 – Anti-tamper

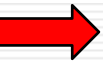
- ◆ Requirement – “The system shall incorporate anti-tamper features”
 - » Vague – requirement needs to be clarified
- ◆ “The system shall be resistant to attacks on code integrity”
 - » Better...
- ◆ “The system shall detect, resist, and create a log of all attempts to change the code.”
- ◆ Potential techniques to apply
 - » Definitive and analytic test – test altered code to verify detection
 - » Demonstration – show that code changes are detected at system load time
 - » Inspection – ensure that code check-sum is valid
 - » Process analysis – verify that safe processes being applied

Example 3 – Open modular system

- ◆ Requirement – “The system shall be an open, highly-modular system”
- ◆ Vague – requirement needs to be clarified
- ◆ “The system shall be designed with internal modules each of which is no larger than 50 KSLOC in size. The interfaces to these modules shall be documented and visible outside the system, and shall be easily replaceable.”
- ◆ Potential techniques to apply
 - » Inspection – verify that the modules are appropriately sized, and that their documentation is published
 - » Demonstration – show that each module can be replaced by alternate modules with same interfaces with less than 1 week of effort.

Agenda

- ◆ Introduction
- ◆ Types of requirements
- ◆ Verification techniques
- ◆ Selecting techniques for requirements
- ◆ Examples
- ◆ Wrap-up



Wrap-up

- ◆ Planning for requirement verification must start early, at same time as requirements are defined
- ◆ Requirements must be written with the goal of ensuring that they can be verified effectively and efficiently
- ◆ Verification must be planned for all types of requirements, not just behavioral
- ◆ Techniques need to be selected appropriate to the type of requirement
- ◆ The quality of the requirement statement usually drives the effectiveness of the verification
 - » Too vague results in loss of confidence