Requirements Development and Management

2011 NDIA System Engineering Conference
San Diego, California

Presented by
Al Florence
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 this presenter.

Tutorial overview

- Tutorial is in two major sections
  - Part I provides a conceptual overview of the nature of requirements
    - Much credit to Dr. Bill Bail - MITRE
  - Part II provides an overview of some practice of writing, validating and verifying effective requirements illustrated with real project examples
    - Developed by Al Florence - MITRE

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

- Conclusion
- References
- Contact Information

Motivation - *Why do we care?*

- Requirements form foundation of all (software) system development
- If we don't handle them properly, we incur significant risk
  - Many historical examples demonstrate this
- Requirements will change over development period
  - Planning ahead will mitigate resulting extra work (and risk)

"You get what you spec, not what you expect"

Motivation – Defense Science Board

Defense Science Board (SDB) Task Force on Defense Software

- Requirements setting and management are hardest parts of system development
- Requirements management viewed as being a fundamental problem
- Problem seen with over-specification of requirements
- Under-utilization of modern technical and management practices for requirements
- Need for advanced technology and tools

Backup – full quotations (1 of 2)

BACKUP

- "The troubled DoD programs reviewed by this team exhibited fundamental problems that were readily identifiable, at least in hindsight. Too often, programs lacked well thought-out, disciplined program management and/or software development processes. Meaningful cost, schedule, and *requirements baselines were lacking*, making it virtually impossible to track progress against them."   Sect 1.4, p. ES-2
- "Requirements trade off. In general, *systems are over-specified*, and in most cases there is no flexibility to adjust the specifications. The acquisition/development team must have latitude to trade requirements for cost, schedule, and risk. This does not mean that overall systems integrity can be compromised."   Sect 1.4, p. ES-4
- "The 1987 DSB Task Force observed that *requirements-setting and management are the hardest part of the software task* and advocated the use of evolutionary practices. This is still true today" Sect 2.5, p. 4

Backup – full quotations (2 of 2)

BACKUP

- "The study also recognized that modern software architecture methods and product lines could improve cost and cycle time. Technical and management practices for *better requirements management* were described and recommended long ago, as was the importance of team experience and technical practices related to architecture reuse. These practices and qualities are hallmarks of commercial best practice, but they remain *largely underutilized* in the acquisition and development of defense software." Sect 2.5 p. 5
- "As complexity and combinatorial difficulty increases, the need for more advanced technology will increase. The gap between system complexity and our abilities is increasing, *exacerbated by difficult requirements* for distributed, embedded, real-time, life-critical, survivable systems. Technology solutions can reduce both the development complexity and the apparent complexity by providing automation to tame the increasing intrinsic or inherent complexity of software." Sect 4.7, P. 32

Motivation - Software Engineering Institute

Capability Maturity Model Integration (CMMI®) v1.3 for:

» Development

» Acquisition

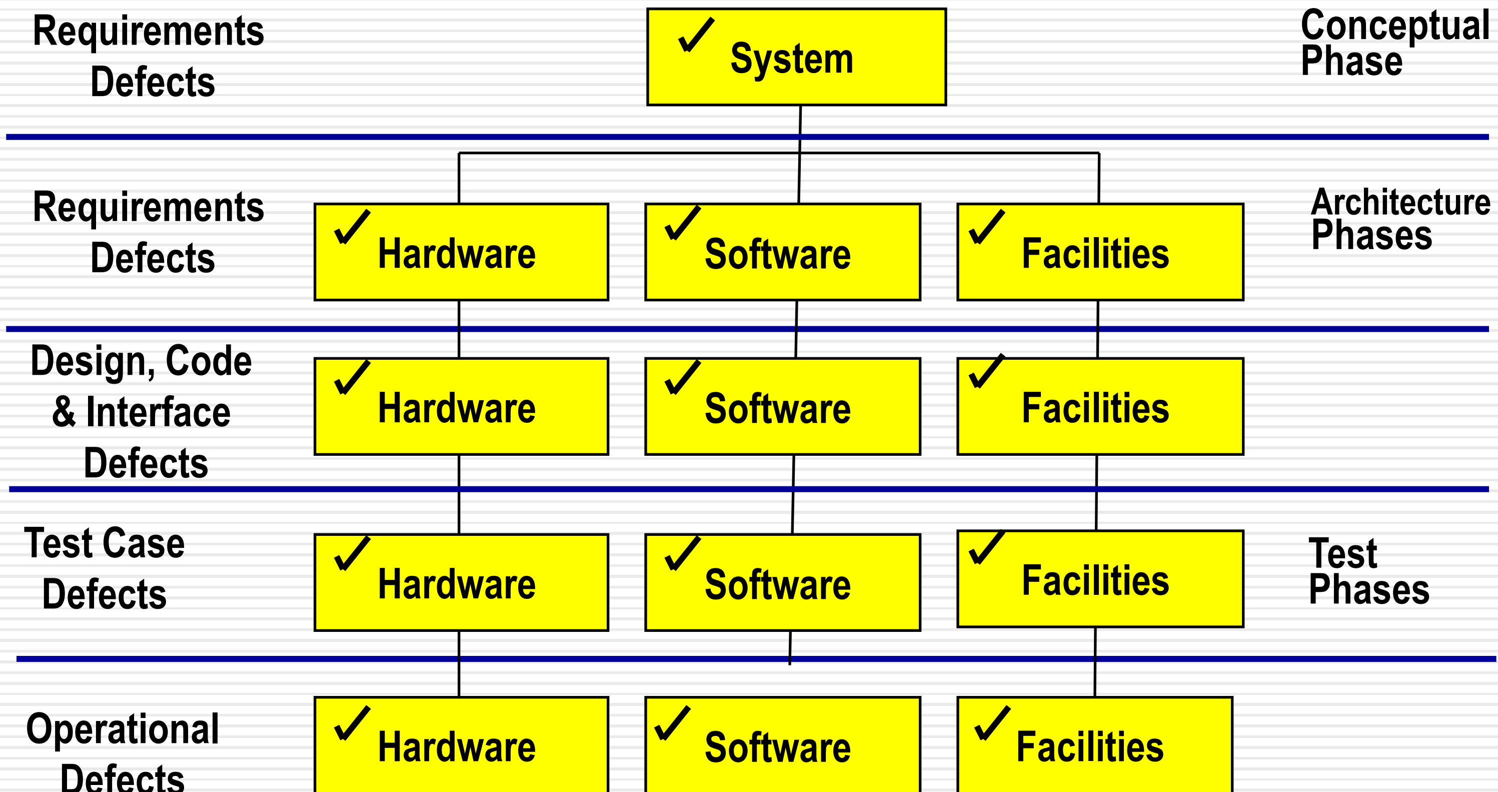
» Services

All these constellations have for process areas for Requirements Development and Management

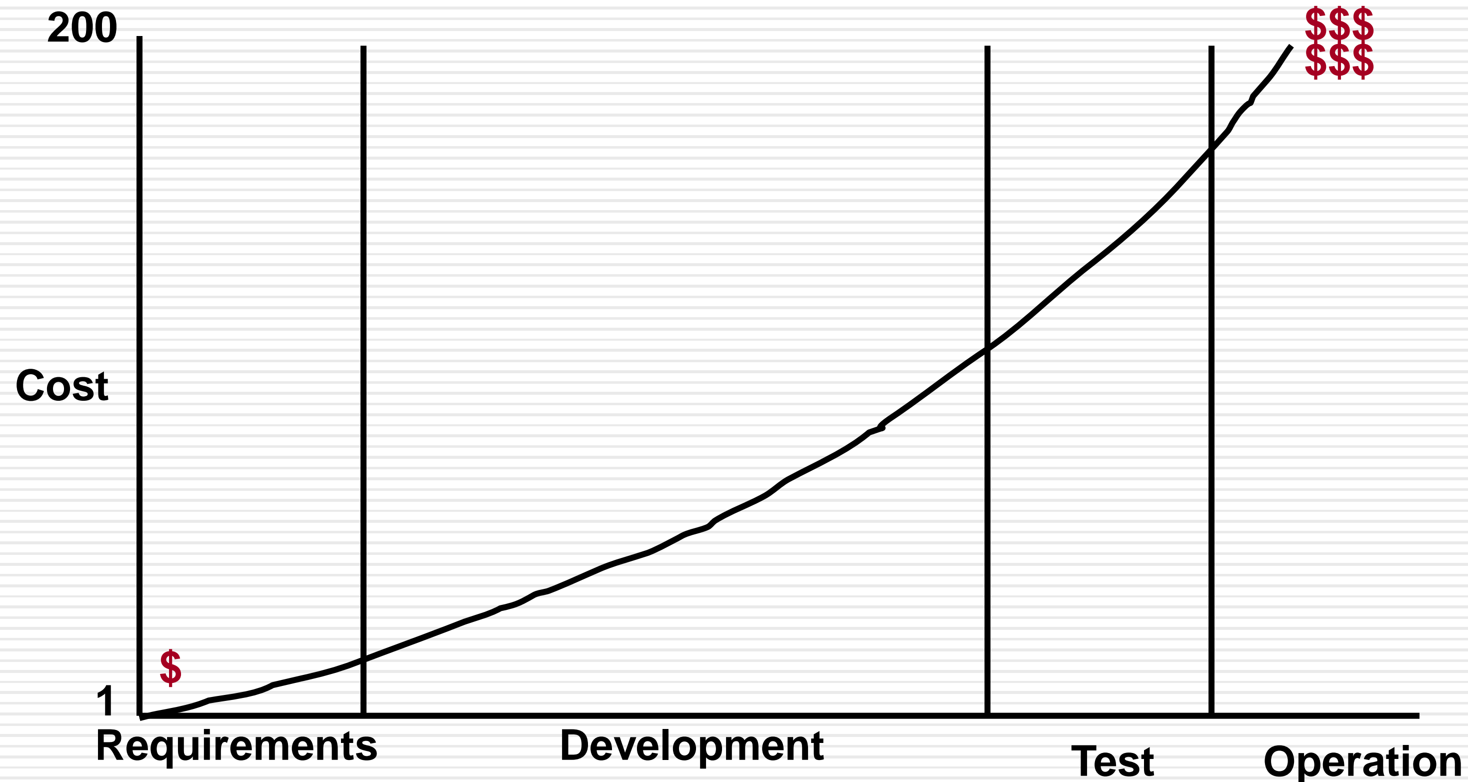
To be compliment with the CMMI organizations need to implement and execute Requirements Development and Management process

CMMI is a registered trademark of the SEI

Motivation – Requirements Defects Propagate



Motivation – Cost of Correcting Defects



Research suggests that correcting software defects can require nearly two hundred times the effort if the correction is implemented in the maintenance phase versus the requirements specification phase of a software lifecycle.
Davis, Alan M. Software Requirements: Objects, Functions, and States. Englewood Cliffs, NJ: Prentice-Hall, *1993.

***This has not changed**

Al Florence

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Conclusion
- References
- Contact Information

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

Nature of requirements - *what are they?* (1 of 7)

So, which word is “right”? (2 of 7)

- Many different words and terms are used
- Many different interpretations of what a “requirement” is
- Which is “right”, “wrong”, “correct”, “best”, ….?
- …………
- Depends…
  - » on what is meant
- *Now, that is helpful…*

- Let’s look at a definition from the IEEE

Nature of requirements - *what are they?* (3 of 7)

- IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology

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.

As-built

Build-to

Many different types - we will explore these in a moment

Nature of requirements - *what are they?*(4 of 7)

- IEEE definition is broad – term often used carelessly
- Popular use often ignores this definition
  - » *Anything that is "required"*
  - » Ranges from hopes to dreams to budgets to schedules...
  - » "This schedule is required"…"This budget is required"…"These software components must be used"…"These algorithms must be used"…
- Different types of requirements need to be handled differently
  - » because they affect system/SW/HW development in different ways
- This section of the tutorial focuses on differentiating different types of requirements
  - » and on recommending how to handle them appropriately
- Remember that all types of requirements are important in some way

Just because something is a requirement, does not mean that it is a requirement

Huh?

Al Florence

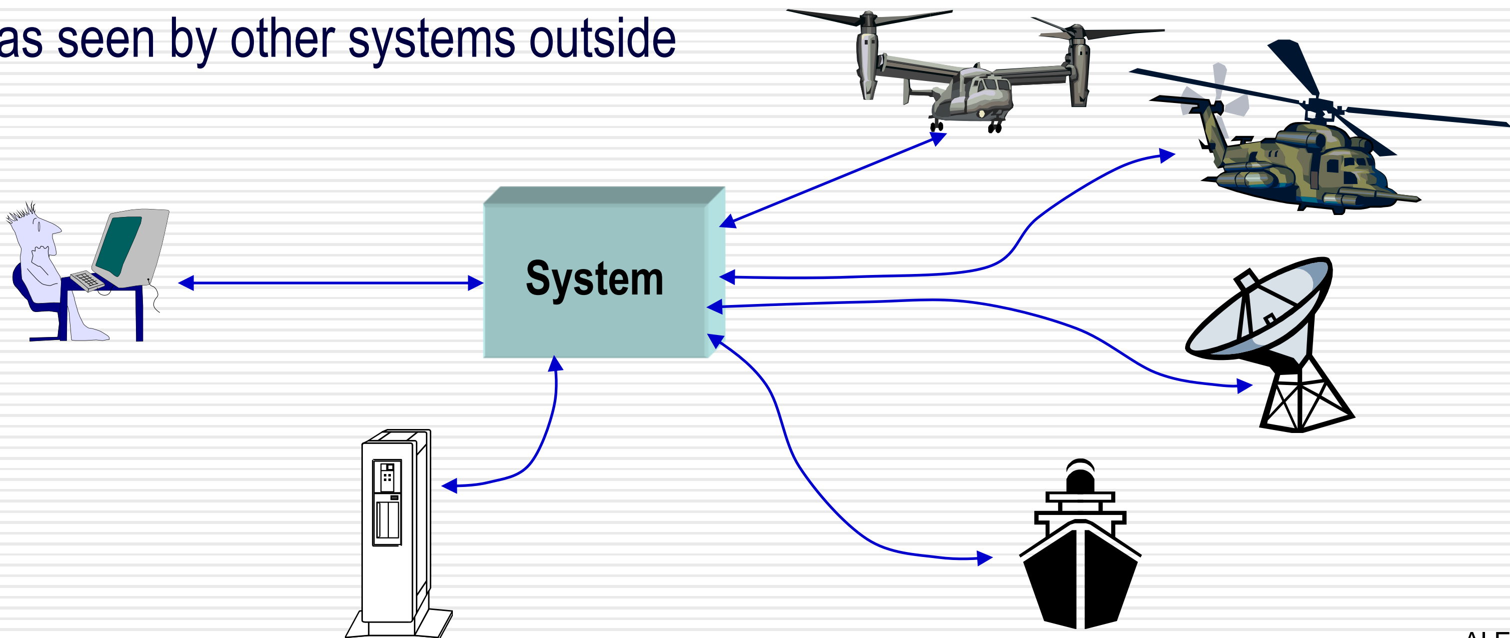
Nature of requirements - *what are they?* (6 of 7)

- IEEE Std 830-1998 – IEEE Recommended Practice for System Requirements Specifications:

"A requirement specifies an externally visible function or attribute of a system"

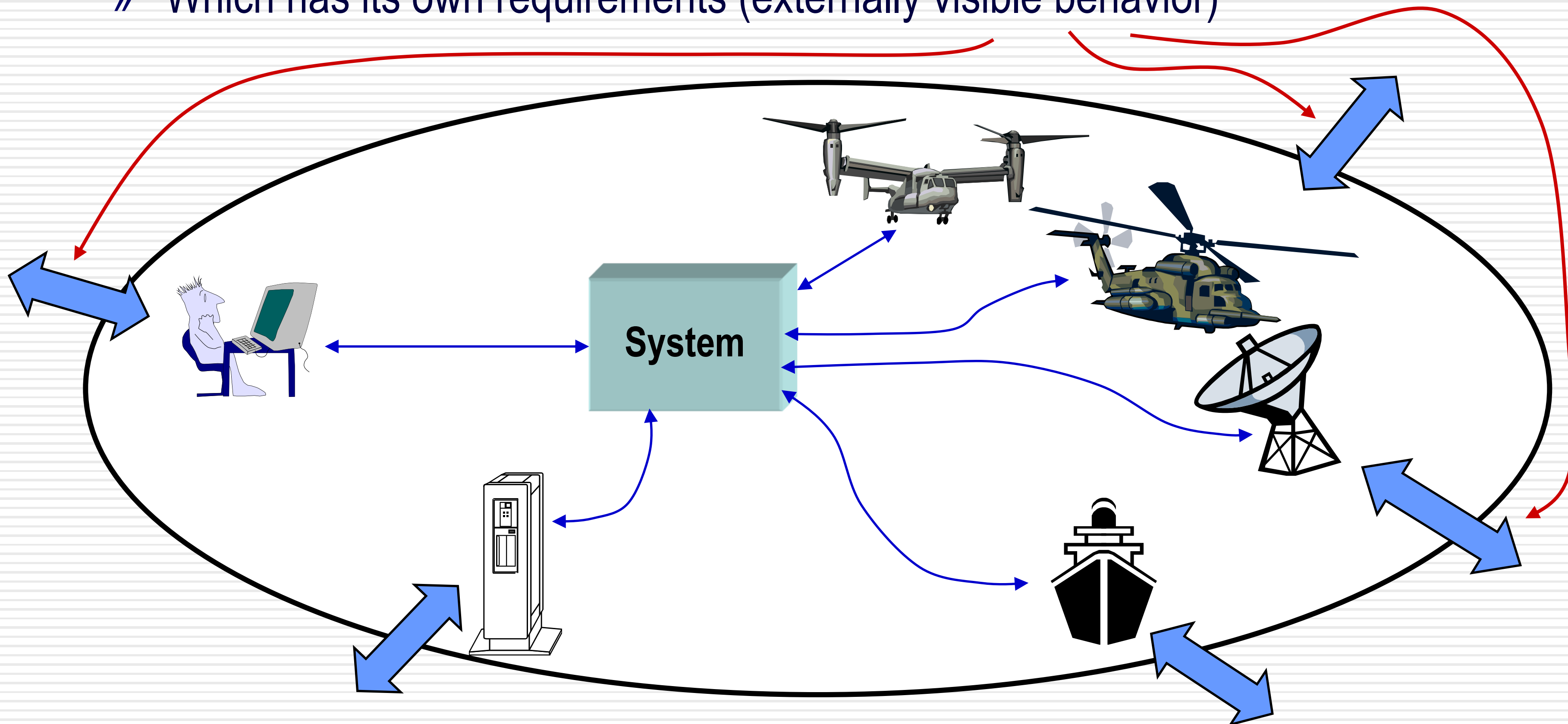
  - » We can see inputs and the outputs, but not what happens inside

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



Nature of requirements - *what are they?*(7 7)

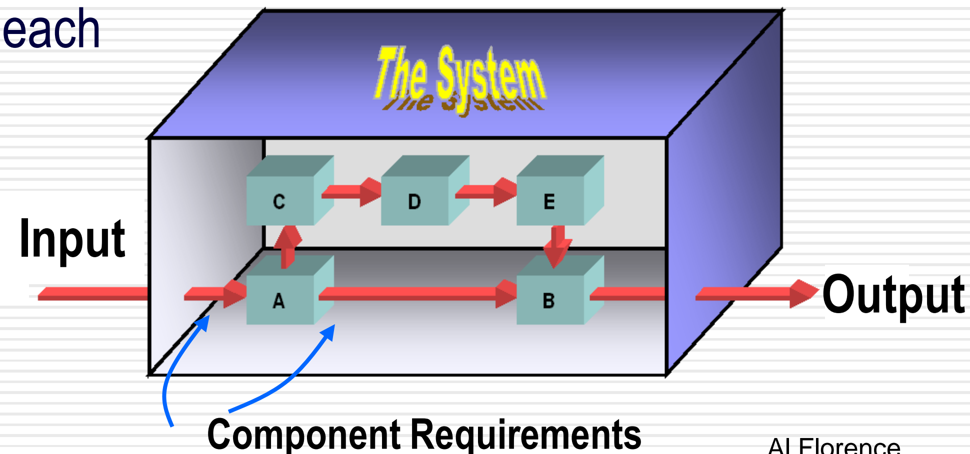
- But each such system could be part of a larger system
  - » Which has its own requirements (externally visible behavior)



For much of this briefing, “requirement” denotes <u>externally visible behavior</u>

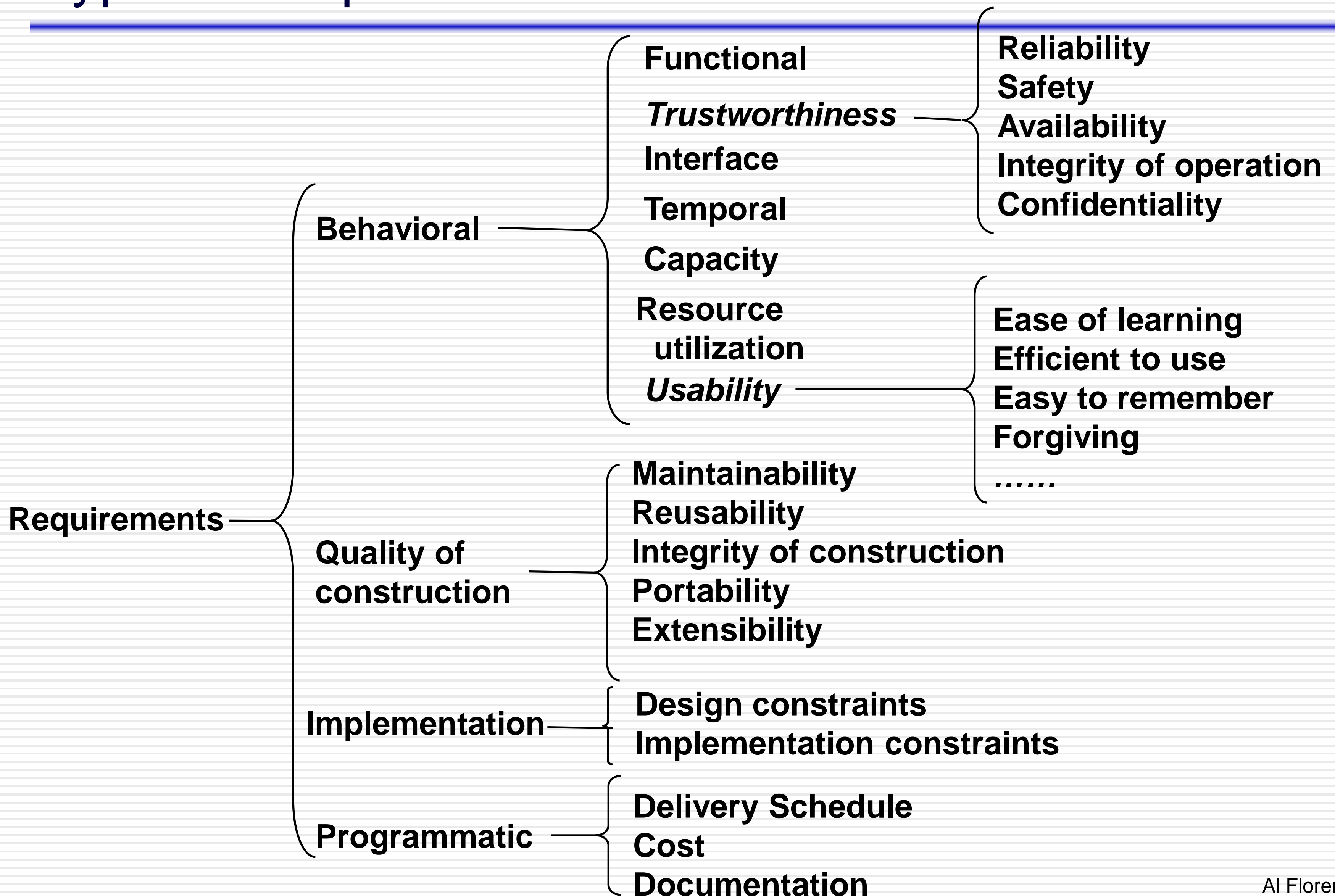
Context of requirements

- All requirements are defined in context of a specific component (e.g., black box)
  - » Which may consist of additional constituent components (e.g., subsystem, modules,...)
  - » 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,...
- 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



Al Florence

Types of requirements



Al Florence

MITRE

Some types of requirements

- ***Behavioral requirements*** - externally visible behaviors of an item (aka *functional specifications, functional requirements*)
- ***Quality of construction requirements*** - qualitative attributes of an item, such as maintainability and portability
  - » Often not directly externally observable – need to examine design and code
  - » Usually deal with how product can be handled
- ***Programmatic requirements*** - terms and conditions imposed as a part of a contract exclusive of behavioral requirements (e.g., costs, schedules, organizational structures) aka *contractual*
  - » Addresses development of product
- ***Implementation requirements*** - aka *implementation constraints, design constraints* – restrictions placed on developers that limit design space
  - » e.g., Use of specific software components
  - » e.g., Imposition of specific algorithms
  - » e.g., Customer-mandated design patterns (e.g., fault tolerance)

Behavioral requirements (1 of 2)

- Externally visible behaviors of an item (component, subsystem, system, unit,…) – all (potentially) measurable by testing
  - » ***Functional*** - input-output behavior in terms of responses to stimuli
    - > Output = fn(input), e.g., x → ☐ → x^2
  - » ***Interface*** **-** characteristics of component's interfaces
    - > e.g., appearance of operator screens (*user*)
    - > e.g., interfaces with other systems/components (*peer-to-peer*)
    - > e.g., computing infrastructure / APIs (*infrastructure*)
  - » ***Temporal*** - speed, latency, and throughput of functional behaviors
    - > e.g., display refreshed screen every 0.5 sec, e.g., x → ☐ → x^2 in 0.5 sec
    - > e.g., process 10,000 database requests per hour
  - » ***Capacity*** - amount of information that can be handled
    - > e.g., 25 simultaneous users
    - > e.g., 20,000 employee records
  - » ***Resource utilization*** - limitations on computer resources that can be used
    - > Memory and processor usage

Behavioral requirements (2 of 2)

- » ***Trustworthiness*** - degree of confidence in product's delivery of functions
  - Reliability - MTTF = 30 hrs
  - Availability - 99% over 30 days
  - Safety (e.g., actions to avoid) – "don't do this!"
  - Confidentiality – avoid unauthorized release of information
  - Integrity – ability of system to avoid being corrupted
- » ***Usability*** - how easy it is for an operator/user to make use of the system
  - For both system to system interfaces and user interfaces

- However, being externally visible does not result in testability
  - » "System shall run for 100 years without failure"
  - » "System shall be able to handle 1000 users simultaneously"
- In general, *trustworthiness* and *usability* requirements cannot be tested directly
  - » Validation based on body of evidence to provide basis for trust

Quality of construction requirements

- Qualitative attributes of an item
  - Deal with how product can be handled
  - Not usually directly measurable or observable
  - We have measures that can give us insight into these qualities, help us to *infer* level of quality
    - Based on related quantitative attributes of systems
  - 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 functionality
  - **Dependability**—degree of confidence in product's delivery of functions
    - Reliability—MTTF = 30 hrs
    - Availability—99% over 30 days

Programmatic (contractual) 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
- However, they directly affect ability to develop a system
  - e.g., not enough money or time

Implementation requirements (1 of 2)

- Restrictions placed on developers that limit design space
- Two important types:
  - » ***Design and implementation constraints*** – restrictions on design styles and coding
  - » ***Process and development approach constraints*** – restrictions on processes and techniques
- Examples
  - » Use of specific software components
  - » Imposition of specific algorithms
  - » Required use of specific designs
  - » Imposition of specific coding styles
  - » Requiring use of a specific language

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 system 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** – 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
- Absence of these qualities is strongly correlated to subsequent problems in development, *i.e.*, ignore at your own risk

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Conclusion
- References
- Contact Information

PART II

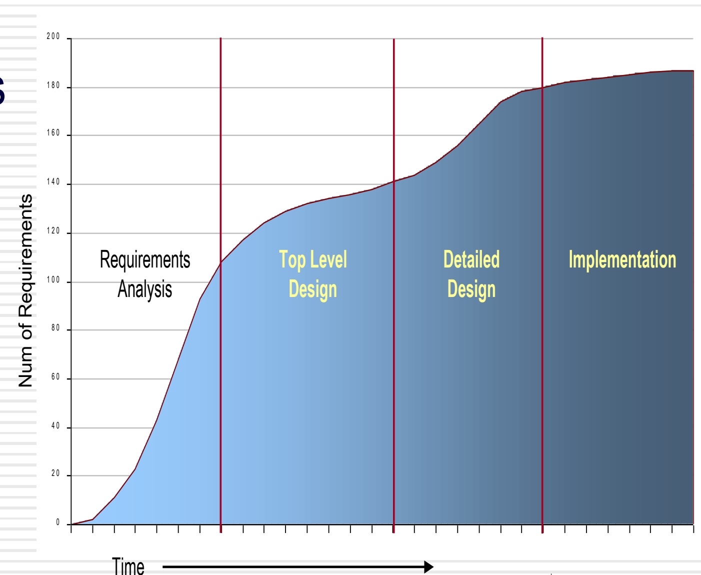
- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

Historic Abstraction

- Multiple levels of components are defined to minimize complexity during construction
  - Systems decomposed into subsystems
  - Subsystems decomposed into hardware/software
  - These are decomposed into lower components
- Each decomposition helps to manage complexity
  - by breaking problem down into smaller problems
- System behavior achieved by interactions between components
- System has requirements for its behavior
  - Each subsystem has requirements for its own behavior
  - Functional composition of subsystems produces system behavior

When Requirements are Defined

- Requirements definition continues throughout development
- During requirements gathering
  - » Gathering user needs / high level system definition
- During systems definition phase
  - » Usually at a general level
- During system design phase
  - » Detail added, capabilities become functions
- During Software/Hardware requirements analysis
  - » More detail added, more specific behaviors and formats
- During software design/implementation
  - » Refinement / clarification / more detail
    - > e.g., GUI details often deferred until later
  - » Changes / adaptations
  - » Additions (e.g., evolutionary development)

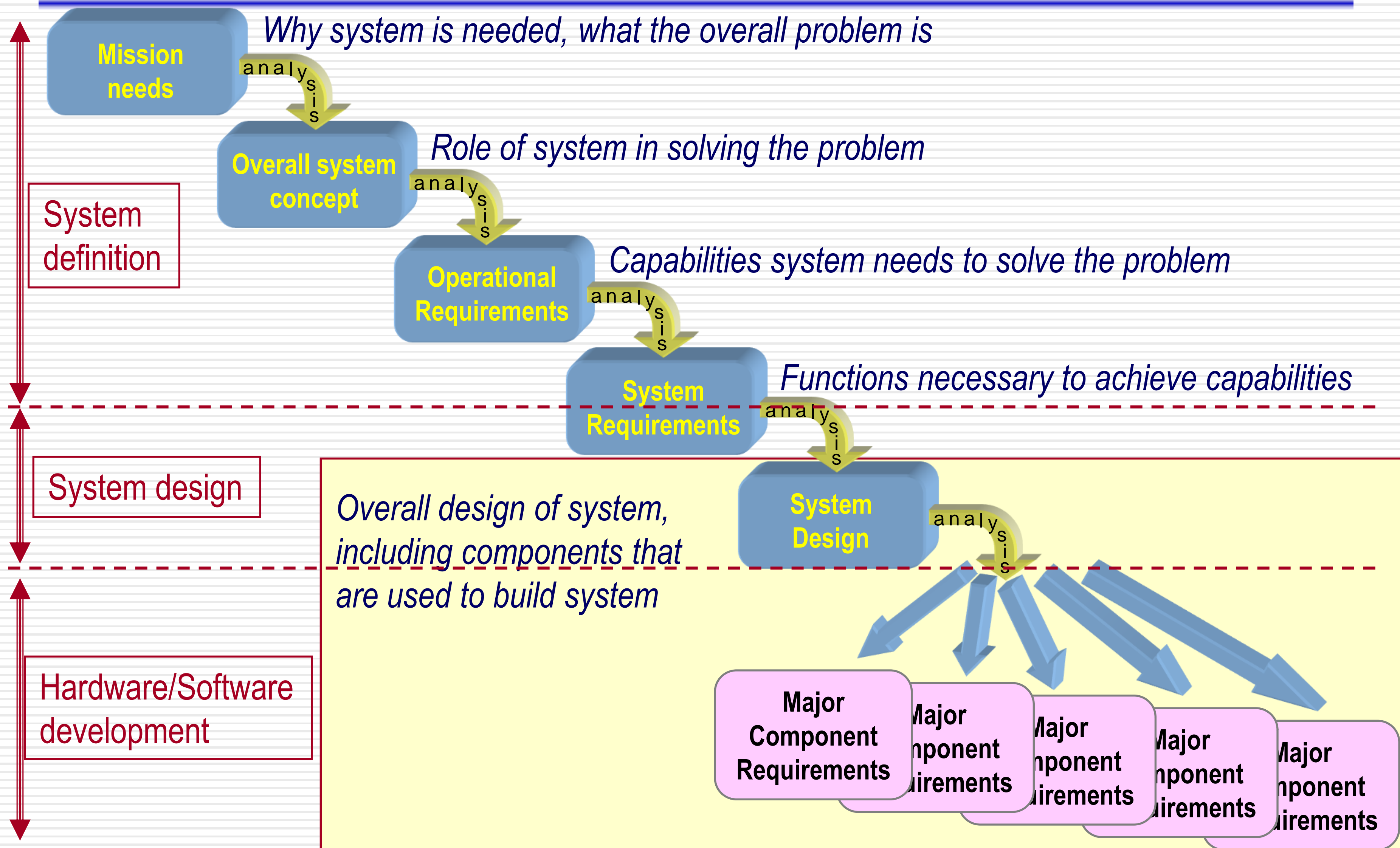


Al Florence

Historic Abstraction

- Multiple levels of components are defined to minimize complexity during construction
  - Systems decomposed into subsystems
  - Subsystems decomposed into hardware/software
  - These are decomposed into lower components
- Each decomposition helps to manage complexity
  - by breaking problem down into smaller problems
- System behavior achieved by interactions between components
- System has requirements for its behavior
  - Each subsystem has requirements for its own behavior
  - Functional composition of subsystems produces system behavior

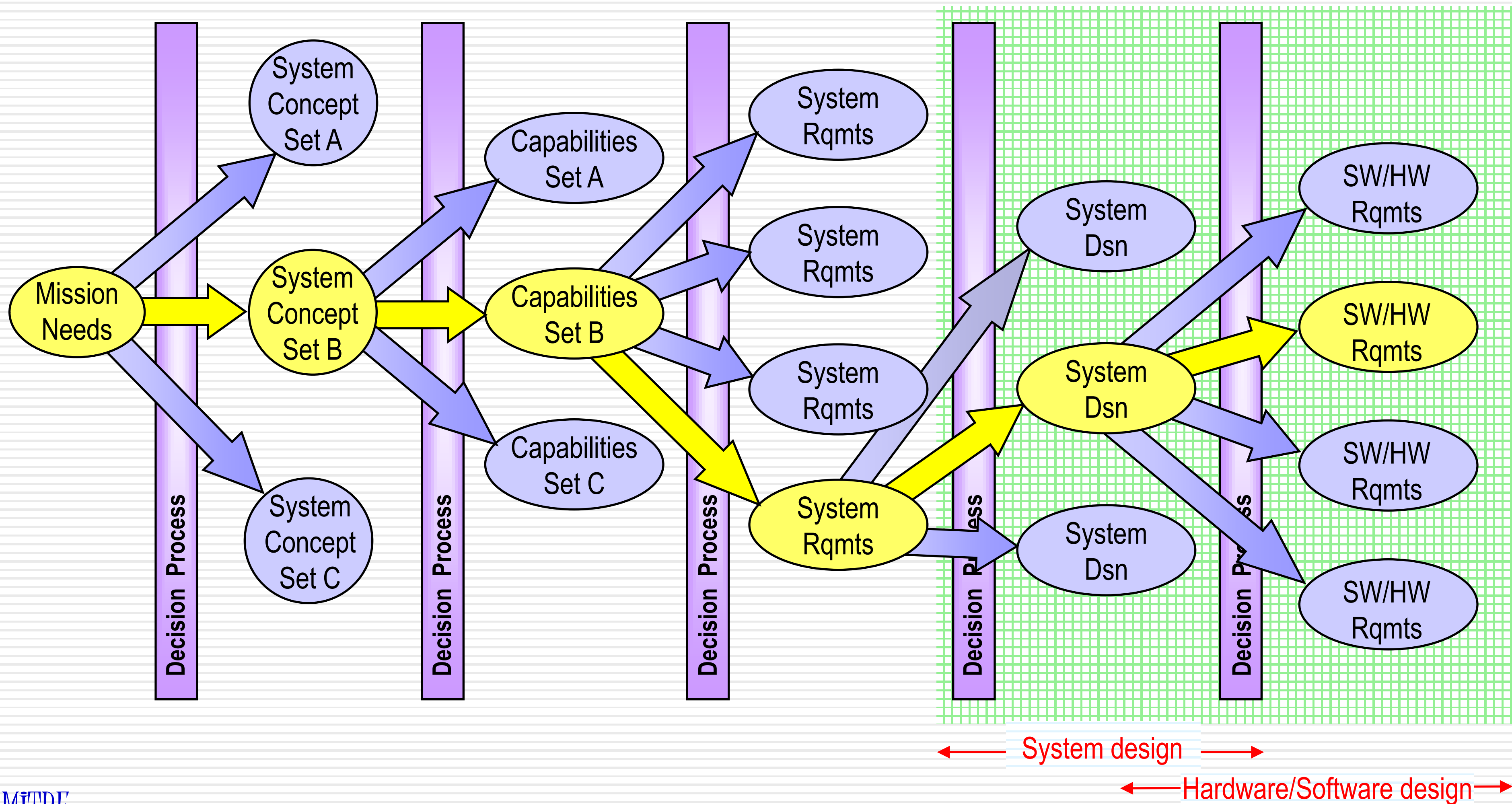
Historic requirements progression



Al Florence

Decision points for requirements definition

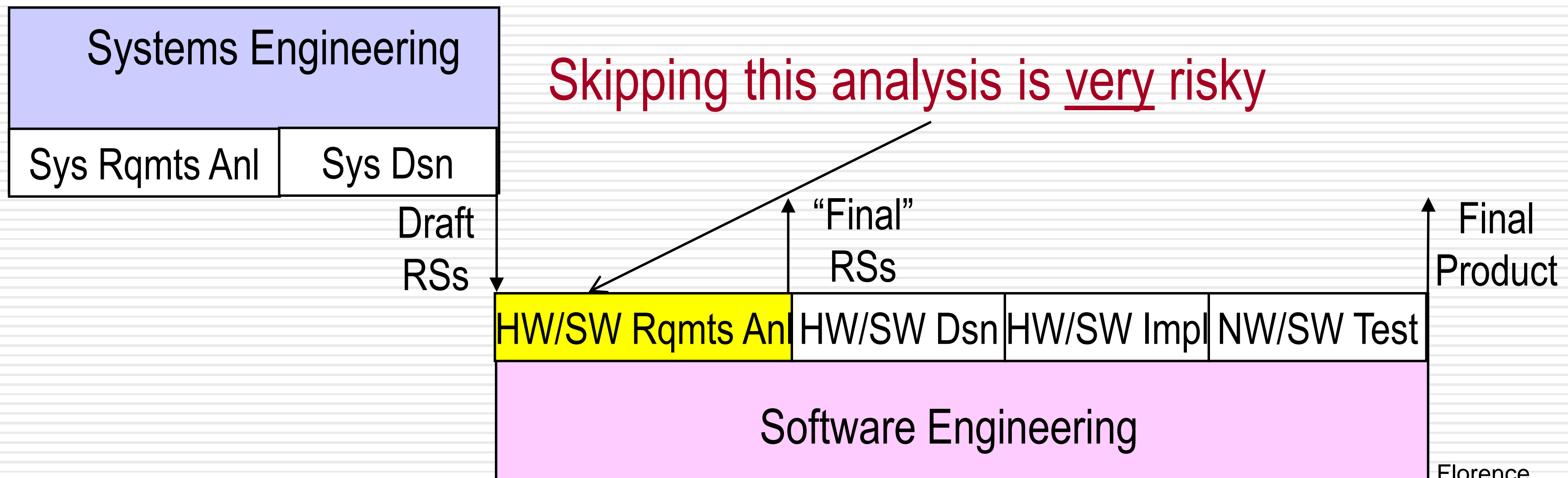
Each step in progression involves deciding between alternative approaches using trade off analyses



MITRE

Requirements analysis

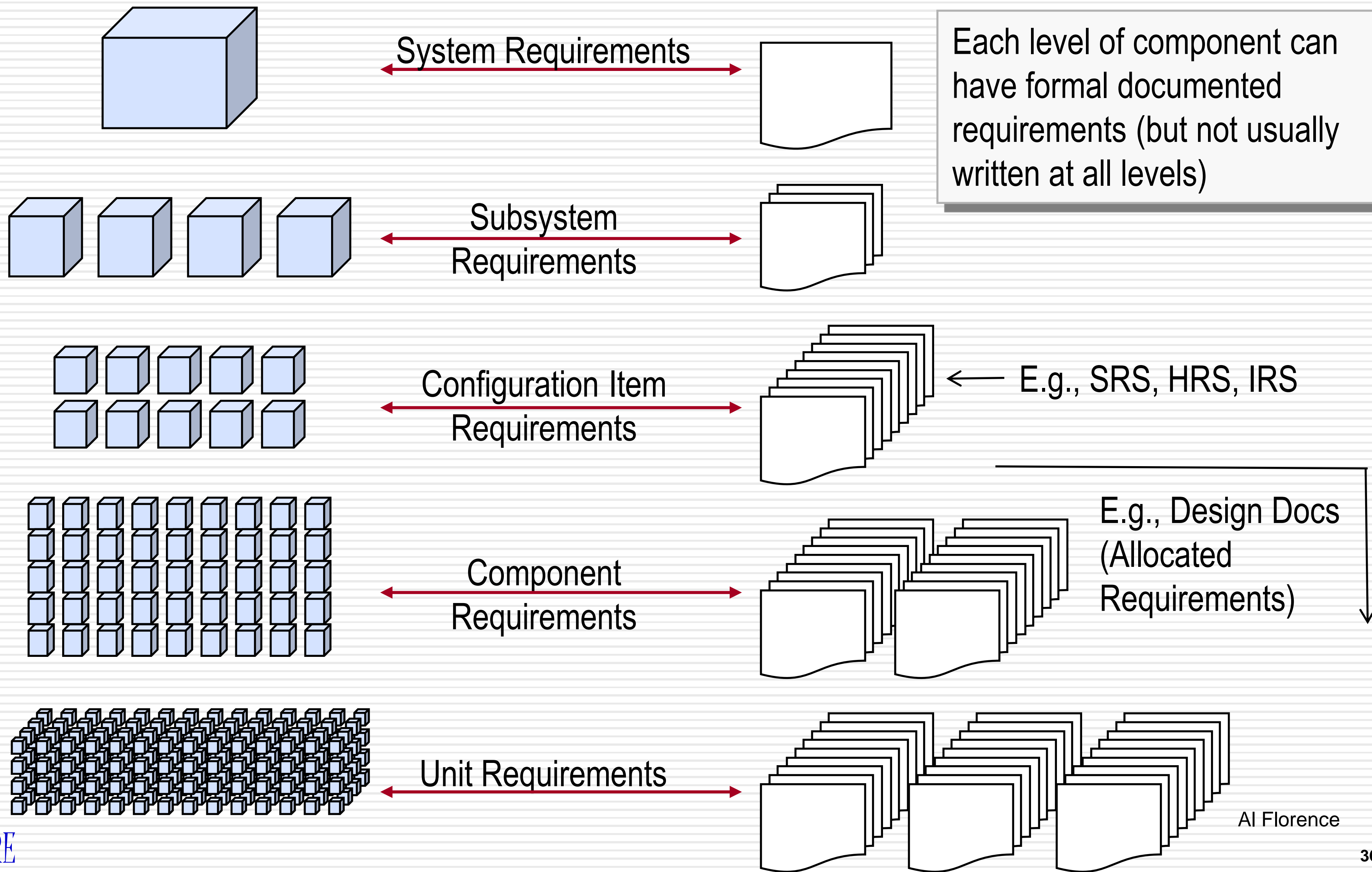
- Requirements allocation to HW/SW components not same as defining HW/SW requirements
- Once requirements allocated to components, HW/SW requirements analysis needed to:
  - » Derive down to specifice HW/SW requirements
  - » Place into form suitable for implementation
- Involves trade-off analyses



Requirements analysis techniques

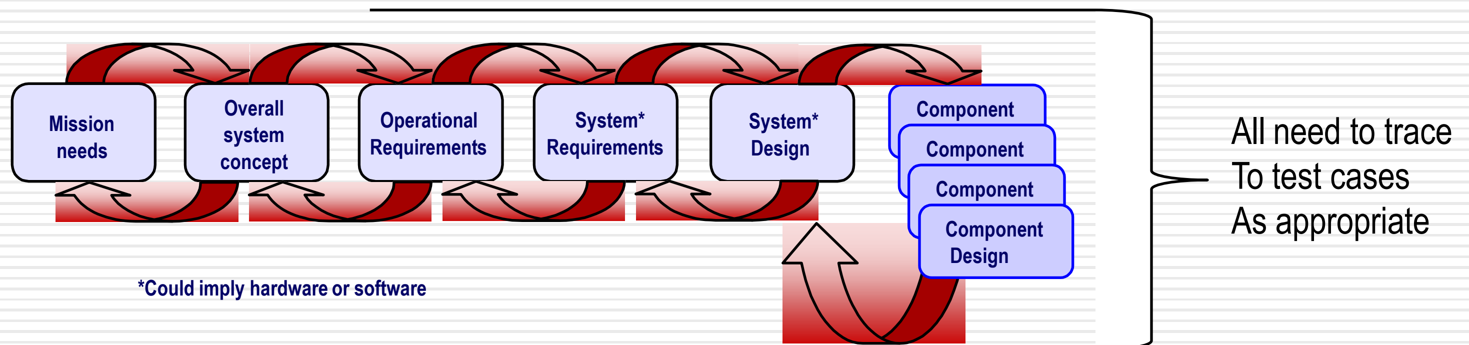
- Different techniques/processes are used
  - ad hoc techniques
  - Functional techniques
  - Object-oriented techniques
- New processes arrive every day
  - Agile Unified Process
  - Extreme Programming
  - Cleanroom Software Engineering
- Many tools exist
  - DOORS
  - Analyst Pro
  - Rational Rose
- All should produce the same result – a description of behavior of the system
- Important to select technique to be appropriate to system

Levels of requirements documentation



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 (from a higher level)
- Sometimes mapping is simple (1–1, 1–many or 1–few)
- Sometimes mapping is complex and/or indirect (1–to–many *or* many–to–1)
- Use of automated tools (such as DOORS) to track mapping recommended



Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Conclusion
- References
- Contact Information

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

Challenges - *some typical problems*

- Requirements volatility
- Delayed definition of requirements
- Human Machine Interfaces (HMIs)
- Prioritizing requirements
- Mixing requirements and design
- Overs-specified / over-constrained / unbounded
- Use cases
- Levels of detail

Requirements volatility (1 of 2)

- Requirements always change – get used to it
  - Some don't change, but are defined late
- Not necessarily bad (sometimes good) but careful management needed 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 require
- Developers better able to design defensively if they know
  - Which requirements are likely to change
  - Degree of change that could be expected
- Remember IEEE quality – *ranked for stability*

Requirements volatility – recommendations (2 of 2)

- The following are 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
  - » Correlate with assessment of late definitions
  - » Assess dependencies between requirements and design
    - > Some requirements deeply affect design globally
    - > Others have limited design impact (GUI formats)
  - » Ensure requirements dependencies are well understood
  - » Define and monitor requirements stability with metrics
    - > Track immature requirements, undefined requirements, and changing requirements

Delayed definition of requirements (1 of 2)

- 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

Delayed definition of requirements (2 of 2)

- 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

	Early definition	Late definition
Level of understanding of desired behavior	high	low
Likelihood that requirement will change	low	high
External component dependencies	complex	simple
Internal design dependencies	strong	weak

Al Florence

Human Machine Interfaces (1 of 2)

- Human Machine Interface
  - » General term referring to interface between the system and humans
    - > Operators, information recipients, spectators, …
  - » How information is conveyed to humans from system
  - » How information is provided to system by humans
  - » How control is achieved by operators over system
- Sometimes referred to as
  - » HCI (human computer interface), MMI (man machine interface), HSI (human system interface), ...
  - » GUI - graphical user interface
    - > Strictly speaking, GUI refers only to HMIs that employ graphics
    - > Text-based menus generally are not included in this category
- HMIs are externally visible – hence are part of system requirements
  - » Failure to treat as requirements can lead to problems

Human Machine Interfaces (2 of 2)

- If HMI is awkward and ineffective, system will be a failure
  - If system cannot be effectively and efficiently used, role will be diminished
- Recommendations:
  - For systems where humans are a part of mission, human performance must be a part of system performance
    - Define performance requirements 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 produce common view

Prioritizing requirements (1 of 2)

- Not all requirements are equal
  - » Some are more firm than others
  - » Others may be guesses and have flexibility about final behavior
- Traditional approach was to use “shall” to denote firm requirements
  - » But too binary
- One approach – “Threshold” and “Objective”
  - > Threshold – minimum acceptable value necessary to satisfy need
    - Failure to meet threshold values will seriously degrade program performance, make program too costly, or cause program to be no longer timely
  - > Objective – value desired by user
    - Represents an operationally meaningful, time critical, and cost-effective increment above threshold for each program parameter
    - Program objectives (parameters, and values) may be refined based on the results of preceding program phase(s)

Prioritizing requirements (2 of 2)

- Recommendations
  - Define rigidity of requirements and ranges of acceptance
    - Thresholds and objectives
    - “Must haves” versus “wanna haves” versus “wouldn’t it be nice”
  - Assess potential for being changed
  - Remember IEEE quality *ranked for importance*

Mixing requirements and design

- What might happen...
  - Inefficient and ineffective testing
    - Software testing is based on System Requirements Specifications (SRS)
    - If SRS contains design as well as behavior, either
      - Testers must separate design from behavior before testing, or
      - Testers must test for design as well as behavior, requiring breaking into internals of HW/SW
  - Inefficient processing
    - If algorithm is specified as part of SRS, designers might not have flexibility to optimize
  - Excessive CM effort – baselined design changes require Configuration Managment authority
- Recommendations:
  - Place all design information (including. algorithms) into separate volumes
    - e.g., A Design Document
    - Ensure all requirements are externally visible and can be tested without examining design/construction

Over-specified / over-constrained / unbounded

- 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
- Too general – results in inefficient system that does everything but not well
- Result is wasted resources
- Recommendations (these are very general but important)
  - » Focus on prioritization of requirements
  - » Ensure what is needed is emphasized
  - » Build system in a series of increments
    - > With most critical functions completed early

A note on use cases and requirements

- *According to Brooch, 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."*
- *This is misleading*
- Use cases provide some information about behavioral requirements but do not specify them to sufficient level of detail for development
- They are however when used with scenarios fundamentally important to requirements definition
  - » Help to elicit requirements by looking at the users' view
  - » Help to describe how system will be used

A use case is an important *AID* to defining requirements - but use cases do not themselves define requirements

Levels of detail for requirements (1 of 2)

- 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
    - e.g., preexisting external interfaces
    - e.g., required screen formats and display icons
- As development proceeds, general requirements are refined until they become specific behaviors
  - “Pressing Ctrl and I at the same time results in the selected text being converted to an italics font within 0.5 sec”
- At some point, all requirements defined
  - Perhaps not formally

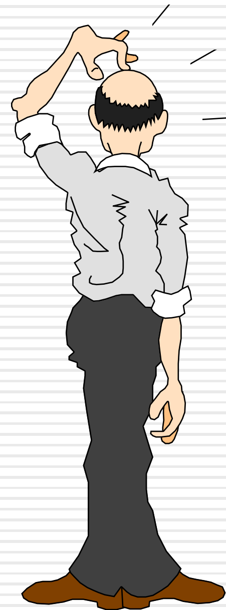
No matter how much detail is provided, requirements are never design (for a specific component)

Levels of detail for requirements (2 of 2)

- 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)
- ***But wait!!!!!*** *(the audience protests...)* ***I don't care about a lot of the details – I know what my real requirements are, the rest is in the noise!! Let the designers decide later.***
- Good idea, but remember, what is or is not a requirement (in our specific use of term) is not dictated by whether it is important or not
  - just whether it is externally visible outside of component

End of Part I

- This half of tutorial presented an overview of some essentials of requirements engineering
  - » Many additional details are involved
- Proper care and feeding of requirements is essential to project success
  - » Different types of requirements must be managed separately
- Any questions?....



Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Conclusion
- References
- Contact Information

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

Requirements Management (1 of 3)

- Requirements management
  - » Starts at the system level
  - » Continues through lower levels of requirements allocations
  - » Continues throughout all development phases
  - » Through the entire lifecycle
- An understanding of the requirements at all levels is established with all stakeholders (users, developers, customers)
- Commitments to Requirements are obtained at all levels with all stakeholders
- Inconsistencies between project work and requirements are resolved

Requirements Management (2 of 3)

- Traceability of requirements is essential so requirements can be traced to/from design, implementation and test.
- Bi-directional traceability of requirements is established and maintained:
  - » User needs to/from system level
  - » System level to/from subsystem level to/from system tests procedures
  - » Subsystem to/from software/hardware/facilities/procedures/subsystem test procedures, etc.
  - » Software requirements are traced:
    - > to/from design
    - > to/from software test procedures
    - > to/from software test reports

Requirements Management (3 of 3)

- Changes to requirements are managed at all levels
  - » Requirements are baselined
  - » Changes to baselined requirements are:
    - > Formally requested
    - > Assessed not limited to:
      - Cost to implement
      - Schedule time to implement
      - Functional effect
      - Interface effect
    - > Approved or disapproved by:
      - Acquirer
      - Supplier

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Conclusion
- References
- Contact Information

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

Verification & Validation

- Verification *(Are we building the product right?)*

The process of determining whether or not the products of a given phase of the development cycle fulfill the requirements established during the previous phase.

- Validation *(Are we building the right product?)*

The process of evaluating the products at the end of the development process to ensure compliance with requirements.

IEEE Standard for Software Verification and Validation Plans

Independence

- Independence of verification and validation means that the associated activities are performed in parallel with, and in addition to, the activities associated with the development of the product.
- Independence also means that the organization performing the IV&V activities is separate from the organization responsible for developing the product.

Scope of IV&V

- The scope of the IV&V effort is dependent on several project factors such as:
  - » Cost
  - » Size of products
  - » Schedule
  - » Complexity
  - » Criticality
  - » Security
  - » Safety
  - » Risk
- IV&V can be very costly.
- Analysis of project factors will support the development of a cost effective IV&V effort that is appropriately tailored to the scope of the application.

IV&V during the Development Life Cycle

- Independent Verification is conducted throughout the development life cycle phases: requirements, design, implementation, integration and test.
- Activities of verification may include:
  - » Reviews
  - » Analysis
  - » Prototypes
  - » Simulations
  - » Testing

Independent Verification (1 of 4)

- Requirements Phase – Independent Verification ensures that the products of the requirements phase satisfy the criteria established during previous activities, such as planning.
- Products to verify may include:
  - Requirements Specifications
  - Interface Specifications
  - Development Schedules
  - Development Plans
  - Quality Assurance Plans
  - Configuration Management Plans
  - Risk Management Plans
  - Test Plans

Independent Verification (2 of 4)

- Design Phase – Independent Verification ensures that the products of the design phase satisfy the criteria established during the requirements phase.
- Products to verify may include:
  - » Design documents
  - » Interface design
  - » Updated products of prior phases including requirements

Independent Verification (3 of 4)

- Implementation Phase – Independent Verification ensures that the products of the implementation phase satisfy the criteria established during the design phase.
- Products to verify may include:
  - » Code
  - » Unit Test Plans
  - » Unit Test Procedures
  - » Unit Test Reports
  - » Updated products of prior phases including requirements

Independent Verification (4 of 4)

- Test Phases – Independent Verification ensures that the products of the test phases satisfy the criteria established during the implementation phase.
- The test phases may cover:
  - » Functional Tests (usually against requirements)
  - » System Integration Tests
  - » Certification Tests
- Test Phases Products may include:
  - » Functional Tests, System Integration Testing, Acceptance Testing, and Certification Testing:
    - > Test Plans
    - > Test Descriptions
    - > Test Procedures
    - > Test Reports
  - » Updated products of prior phases including requirements

Independent Validation

- Independent Validation ensures that the correct products are developed with the main focus on compliance with specified requirements.
- Planning for Independent Validation activities start at or before the requirements phase.
  - » This includes ensuring that requirements are testable
- Independent Validation testing is conducted at the end of the development life cycle.
- Products and activities that are used to validate that the software/system satisfies its specified requirements are:
  - » Validation Test Plans
  - » Validation Test Descriptions
  - » Validation Test Procedures
  - » Validation Test Conduct
  - » Validation Test Reports

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

- Conclusion
- References
- Contact Information

Proper Specification of Requirements

- Verification *(Are we building the product right?)*
  - The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase.
- The following examples show a method of verifying requirements during their specification.
  - Specification in this context means documenting/writing the requirements.
  - Many methods of verifying requirements exist:
    - The verification method here is to verify that the requirements are specified in a fashion that satisfies the needs of the stakeholders (users, developers, customers).
    - These needs have been established in prior activities such as:
      - conceptual design / system level requirements analysis
      - request for proposal
      - proposals
      - planning, etc.

Class Participation

Determine the problems with these 3 requirements:

3.2.5.9 All computer-resident information that is sensitive shall have system access controls. Access controls shall be consistent with the information being protected and the computer system hosting the data.

- All computer-resident information that is sensitive shall have system access controls. Access controls shall be consistent with the information being protected and the computer system hosting the data.

The interval for propagating changes to suppliers shall be configurable.

Independent Verification of Requirements

- A Government agency, while modernizing their information systems, reverse-engineered requirements.
- With domain knowledge of the application, several teams were involved.
  - » They represented:
    - > the users
    - > the contractors
    - > the acquisition organization
- This author was assigned as a consultant to guide the teams in the proper specification of requirements.
- The examples presented show some of the requirements:
  - » as initially specified by the teams
  - » next a critique of the requirements by this author
  - » finally the re-specified requirements based on the critique

Background (1 of 2)

- It needs to be noted that requirements do not “live alone”
  - » They depend on other requirements and/or
  - » on clarifying comments

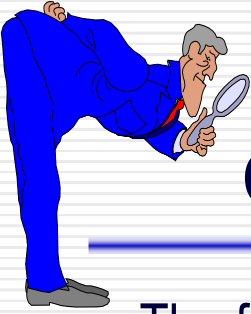
  to present a complete view of the functionality associated with a related set of requirements.

- A related set of functional requirements may be introduced with a preamble describing the capability of the functional set.
  - » The preamble does not itself establish requirements; this is done later in the requirements’ specifications.

- Some requirements may be amplified with clarifying comments which are, again, not part of the requirements, but add understandability.

Background (2 of 2)

- Some requirements are documented sequentially with the requirements stated first setting the “stage” for the following requirements which add more and more capability.
  - » The later stated requirements depend on the earlier requirements to complete their functionally.
  - » An example may be the use of the word “processing”. If the processing of a functional set of related requirements has been described in earlier requirements the later requirements may amplify and/or reference the processing without having to restate the processing.



Criteria for Specifying a Good Requirement (1 of 3)

- The following are some critical attributes that requirements must adhere to:

  used to critique requirements

- Completeness: Requirements should be complete.

  They should reflect system objectives and specify the relationship between the software and the rest of the subsystems.

- Traceability: Each requirement must be traceable to some higher-level source, such as a system-level requirement.

  Each requirement should also be traced to lower level design and test abstractions such as high-level and detailed-level design and test cases.

- Testability: All requirements must be testable in order to demonstrate that the software end product satisfies its requirements.

  In order for requirements to be testable they must be specific, unambiguous, and quantitative whenever possible. Avoid negative, vague and general statements.

Criteria for Specifying a Good Requirement (2 of 3)

- **Consistency:** Requirements must be consistent with each other; no requirement should conflict with any other requirement.

  Requirements should be checked by examining all requirements in relation to each other for consistency and compatibility.

- **Feasibility:** Each requirement must be feasible to implement.

  Requirements that have questionable feasibility should be analyzed during requirements analysis to prove their feasibility,

- **Unique identification:** Uniquely identifying each requirement is essential if requirements are to be traceable and testable.

  Uniqueness also helps in stating requirements in a clear and consistent fashion.

Criteria for Specifying a Good Requirement (3 of 3)

- Design Free: Software requirements should be specified at a requirements level not at a design level.

  The approach should be to describe the software requirement functionally from a system (external) point of view, not from a software design point-of-view, i.e. describe the system functions that the software must satisfy. Some requirements may have design embedded due to constraints placed on them by the system, interfaces or legacy.

- Use of "shall" and related words: In specifications, the use of the word "shall" indicates a binding provision.

  Binding provisions must be implemented by users of specifications. To state non-binding provisions, use "should" or "may". Use "will" to express a declaration of purpose (e.g., "The Government will furnish..."), or to express future tense. MIL-STD-490A

Note: Methods other that the use of "shall" can be used to specify requirements such as using a matrix with a column for requirements and another column for comments or italics or underlines for comments or requirements.

Ensuring the Verification of Requirements

- The process was for the teams to first define and reverse-engineer the requirements based on:
  - » Knowledge of the system
  - » Knowledge of what is needed
  - » Any existing legacy documentation
    - > requirements
    - > design
    - > users manuals
    - > pamphlets
    - > procedures, etc.
- Critiquing this definition of the requirements against the critical attributes and then subsequently correcting them as per the critique ensured that the requirements were verified for stakeholders needs.
- Many iterations of this process were done until it was felt that the requirements were well specified, defined and verified.

Example 1 (1 of 2)

- Initial specification:

3.4.6.3 The system shall prevent processing of duplicate electronic files by checking a new SDATE record. An e-mail message shall be sent

- Critique:
  1. Two "shalls" under one requirement number
  2. When is the SDATE record checked?
  3. Against what other records is the SDATE record checked?
  4. What is checked in the SDATE record?
  5. What action is taken after the SDATE record is checked?
  6. What does the email message say?
  7. When is the email message sent?
  8. The requirement has design implications, SDATE record

A requirement should specify what the data in the record are and not the name of the record as it exists in the design and implementation.

Example 1 (2 of 2)

- Re-specification:

3.4.6.3 The system shall:

a. prevent processing of duplicate electronic files by immediately checking the date and time of the submission against prior submissions, and

b. immediately send the following e-mail message:

   1. request updated submission date and time, if necessary, and
   2. state that the submission was successful, when successful.

Example 2 (1 of 2)

- Initial specification:

  After the system receives the Validation file, the system shall:
  - notify the individual about acceptance or rejection.
  - the acceptance file must contain the name control and ZIP code of the individual.
  - rejected validation request must include the Reason Code.

- Critique:
  1. The second and third bullets don't make sense, try to read them as such:
     - the system shall the acceptance file must...
     - the system shall rejected validation…
  2. Use of both "shall" and "must".
  3. Where are the reason codes?
  4. Who is notified?
  5. How is the individual notified?
  6. No unique identifier
  7. Use of bullets, bullets are difficult to trace.

Example 2 (2 of 2)

- Re-specification:

  3.2.7.3 When the system receives a validation file, the system shall:

  a. reject the file if it does not contain the individual's:

     1. name, and/or
     2. ZIP code, and

  b. notify the individual via electronic transmission about acceptance or rejection with a reason code for rejection. (Reference Reason Code, Table 5.4.8), and

  c. request corrected resubmission, if rejected.

Example 3 (1 of 2)

- Initial specification

  The Financial Agent sends to the Government by 6:00 PM ET on the same day after receipt the file CRDF that includes only critical data collected from the enrolled individual.

- Critique

  1. No unique identifier provided.
  2. The word “shall” is missing.
  3. How is the file sent?
  4. Has design implications: “CRDF”.

     Should define data, not name of data file - this should be done in the design.
  5. The critical data has to be identified.

Example 3 (2 of 2)

- Re-specification

3.3.1.3 The Financial Agent shall send the Government, via electronic transmission, the following critical data collected from each enrolled individuals by 6:00 PM ET on the day of receipt or the next day if received after 5:30 PM:

a. Name,

b. Address,

c. Zip code,

d. Social security number.

Example 4

Initial specification:

Software will not be loaded from unknown sources onto the system without first having the software tested and approved.

Critique:

- If it's tested and approved, can it be loaded from unknown sources?
- If the source is known, can it be loaded without being tested and approved?
- Requirement is ambiguous and stated as a negative requirement, which makes it difficult to implement and test.
- A unique identifier is not provided, which makes it difficult to trace.
- The word "shall" is missing.

Re-specification:

3.2.5.2 Software shall be loaded onto the operational system only after it has been tested and approved.

Example 5 (1 of 2)

Initial specification:

3.2.5.7 The system shall process two new fields (provides production count balancing info to states) at the end-of-state record.

Critique:

- This requirement cannot be implemented or tested.
- It is incomplete. What are the two new fields?
- “Info” should be spelled out.

Re-specification:

3.2.5.7 The system shall provide the following data items (provides production count balancing information to states) at the end-of-state record:

a. SDATE, and

b. YR-TO-DATE-COUNT

Example 5 (2 of 2)

Re-Critique:

- This rewrite has design implications SDATE record and YR-TO-DATE-COUNT.
- From a requirements viewpoint it should specify what the data in the records are, not the name of the record as it exists in the design and implementation.

Re-Re-Specification*:*

3.2.5.7 The system shall provide the following data items (provides production count balancing information to states) at the end-of-state record:

a. submission date and time, and

b. year-to-date totals.

Example 6

Initial specification:

3.2.9.1 When doing calculations the software shall produce correct results.

Critique:

- Really? This is not a requirement.
- This type of requirements should not be specified!
- It should be deleted.

Re-specification:

Requirement deleted.

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

- Conclusion
- References
- Contact Information

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

Independent Validation of Adherence to Requirements

- Validation *(Are we building the right product?)*
  - » The process of evaluating software at the end of the software development process to ensure compliance with software requirements.
- These examples show a method of constructing validation scenarios and procedures to support the evaluation of a software product to ensure that the it satisfied its requirements as specified.

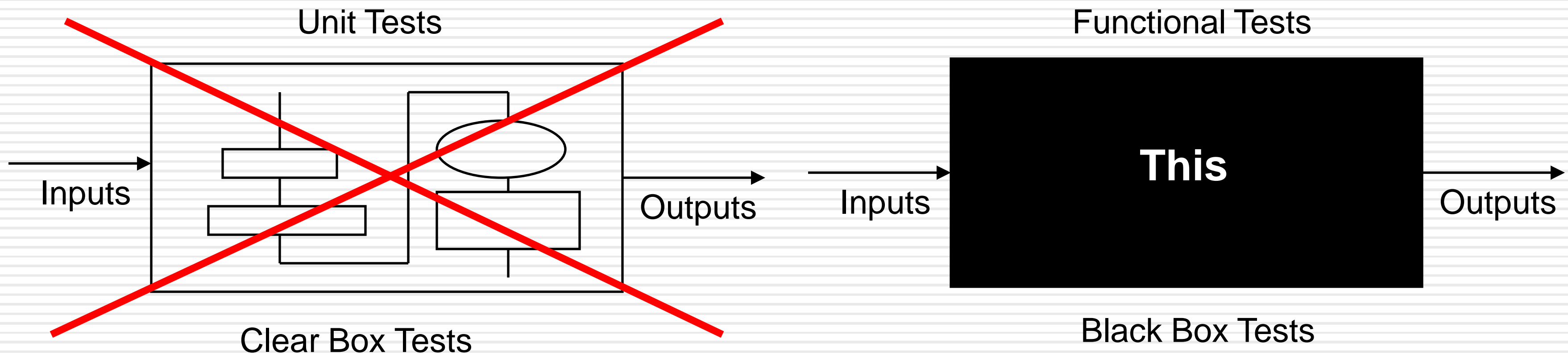
Independent Validation of Adherence to Requirements

- A Government agency's system
  - » had been operation for 5 years
  - » had never been through either functional or acceptance tests.
  - » had almost no documented requirements
- MITRE was asked to support an IV&V effort on the system.
  - » reverse-engineered requirements from:
    - > whatever legacy documentation existed
    - > interviews with domain experts and system users
  - » developed:
    - > Validation Plans
    - > Validation Schedules
    - > Validation Scenarios
  - » supported the development of validation procedures
- The Government agency conducted the validation testing

Example of Validation

- The following examples Illustrate:
  - » High-level validation scenarios
  - » Detailed validation procedures

Test execution examples are not provided since test activities are beyond the scope of this presentation.



Validation Scenarios

Validation scenarios describe at a high-level what needs to be accomplished during testing to ensure that the implemented system satisfies its requirements as specified. Validation scenarios describe the functionality that is to be tested.

- Validation Scenarios include:
  - A high-level statement of the purpose of the scenario (requirement/functionality tested)
  - Description of scenario
    - test conditions
    - test conduct
    - test validation
  - Description of what/where to validate
  - Identifies the validation method : test, demonstration, inspection, analysis

Validation Procedures (1 of 2)

- Detailed validation procedures implement the validation scenarios and describe how the testing is accomplished to validate that the system, as developed, satisfies its requirements as specified.
- The validation procedures establish the specific data and steps needed to be performed in order to validate the system/software against its requirements.

Validation Procedures (2 of 2)

- The following provides a sample of the information contained in each validation procedure:
  - » Identification of requirements tested by the procedure.
  - » Identification of test data or other information required to determine test results.
  - » Test operators' actions for each step, as required:
    - > Initiate the test case and apply test inputs
    - > Perform interim evaluations of test results
    - > Request data dumps
    - > Record data and test results
    - > Modify data, if needed
    - > Repeat the test case, if needed
    - > Use evaluation criteria to validate that requirements are satisfied
    - > Determine Pass/fail
    - > Provide test comments

Scenarios/Procedures

- Scenarios and procedures can be developed at different levels for either:
  - an individual unique single requirement
    - One scenario and one procedure may be necessary for a requirement
  - a logically related set of requirements that provides a functional capability
    - In this case, the set of requirements may be grouped and addressed by one or a few scenarios and procedures

Example 1 (1 of 3)

Requirement

3.3.1.3 The Financial Agent (FA) shall send the Government the following critical data collected from the enrolled individuals by 6:00 PM ET on the same day as receipt or the next day if received after 5:30 PM:

a. Name,

b. Address,

c. Zip code,

d. Social security number.

Example 1 (2 of 3)

Validation Scenario

S0032 for 3.3.1.3 - Validate that the FA sends the Government critical data collected from the enrolled individuals by 6:00 PM ET on the same day as receipt or the next day if received after 5:30 PM .

1. Construct a file with the required critical data for an individual.
2. Initiate input to the system of the constructed file.
3. Validate that the requirement was met.

Validation Method - Demonstration

To validate that the requirement was met, check to see if the Government received the critical data by 6:00 PM ET on the same day as receipt by viewing the appropriate file in the Government's system.

Example 1 (3 of 3)

Validation Procedure

Date	Req#	P/F	Scen#	Procedure P0024	QA
	3.3.1.3		S0032	1. Data: Al Florence, 26 Dutch Creek Drive, Columbine, Colorado, 80123-1623, 374-XX-4237 2. Input data into Enrolled Individual Critical Data file on FA System. 3. Initiate the execution of the Enrollment Function on the FA System. 4. Validate that the Government received the data in (1) by 6 PM ET on the same day as receipt by checking the Enrolled Individual Critical Data file in the Government System.	
Comments					

Date - Date test conducted
P/F - Pass/Fail indication
Procedure - Procedure's number and text
Req # - Requirement(s) being verified
Scen # - Scenario being implemented
QA - Quality Assurance witness' initials
Comments - Comments on test results

Example 2 (1 of 3)

- Requirement 1

  3.3.2.1 Prior to noon each day, the FA shall accept a payment file from the enrolled individual.

- Requirement 2

  3.3.2.2 Within one hour after receipt of the payment file from the individual submitting the payment file, the FA shall provide the individual an acknowledgement of its receipt.

- Requirement 3

  3.3.2.3 Upon receipt of the payment file, the FA shall:

  a. Reject the payment file if the individual is not enrolled.

  b. Reject the payment file if the payment type is invalid.

  c. Send the payment file to the Government if the payment file is not rejected.

Example 2 (2 of 3)

Validation Scenario

S0033 for 3.3.2.1, 3.3.2.2, 3.3.2.3. Validate that the FA receives payment file, sends acknowledgement, correctly processes, and sends the payment file received from the individual submitting the payment to the Government.

1. Construct:
   a. enrollment records,
   b. payment files - multiple sets representing enrolled and non-enrolled individuals, and valid and invalid payment type,
2. Initiate input to the FA of the constructed files,
3. Validate that the requirements were met.

Validation Method - Demonstration

- To validate that the Financial Agent received and accepted the payment files from the individuals submitting these files, check that the FA sends the acknowledgement.
- To validate that the FA correctly processed the payment file and sent the payment file to the Government, check the appropriate Government files.

Example 2 (3 of 3)

Validation Procedure

Date	Req#	P/F	Scen#	Procedure P0025	QA
	3.3.2.1 3.3.2.2 3.3.2.3		S0033	1. Enrollment Data: a. Steve Jenkins, 244 Maple St, Fairfax, VA 20171, 334-XX-4445; b. Jeff Hunt, 517 Main Ave, Fairfax, VA 20171, 422-XX-5555; 2. Payment Data: a. Steve Jenkins, 334-XX-4445, Valid Payment Type; b. Jeff Hunt, 422-XX-5555, Invalid Payment Type; c. Barbara Jones, 335-XX-1234, Valid Payment Type; d. Fred Smith, 275-XX-4321, Invalid Payment Type; 3. Initiate input of enrollment data to the FA System. 4. Check for enrollment file acknowledgements. 5. Initiate input of payment file to the FA System. 6. Check for payment file acknowledgements. 7. Analyze Government files for receipt and correct processing by the FA. Only the payment file for Steve Jenkins should be in the Government files since only Steve Jenkins was enrolled and presented a valid payment type.	
Comments					

Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

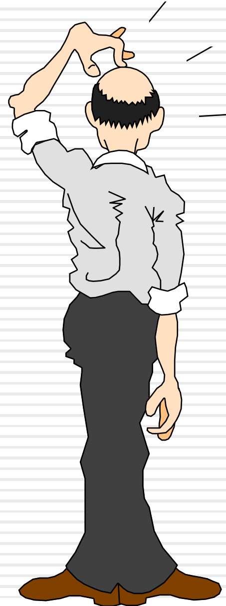
- Conclusion
- References
- Contact Information

PART II

- Requirements Management
- Independent Verification & Validation of Requirements
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation (Adherence to Requirements)

End of Part II

- This half of the tutorial presented:
  - » Information on requirements management and IV&V
  - » Examples of applying some requirements practices to real projects
- Proper care and feeding of requirements is essential to project success
- Any questions?....



Agenda

PART I

- Motivation - *why do we care?*
- Nature of Requirements - *what are they?*
- Creating Requirements - *How do we define them?*
- Challenges - *some typical problems*

PART II

- Requirements Management
- Independent Verification & Validation
- Examples of Effective Requirements Practices
  - » Independent Verification of Requirements (Proper Specification of Requirements)
  - » Independent Validation of Requirements

- Conclusion
- References
- Contact Information

Conclusion

- Applying Effective Requirements Practices to the:
  - » Management
  - » Specification
  - » Implementation
  - » Verification
  - » Validation

of requirements will increase the probability of developing high quality systems that meet specified requirements within cost and schedule.

References and Suggested Readings

- Brooch, Grady, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley. 1999
- *Capability Maturity Model Integration (CMMI®), Version 1.3.* Software Engineering Institute. 2011
- Florence, Al, *Reducing Risk with the Proper Specification of Software Requirements*. CrossTalk, The Journal of Defense Software Engineering. April 2002
- Dr. Young, Ralph, *Effective Requirements Practices,* Addison-Wesley, 2001.
  (includes over 230 references on requirements!)
- Dr. Young, Ralph, *Recommended Requirements Gathering Practices. CrossTalk, The Journal of Defense Software Engineering.* April 2002
- *IEEE Recommended Practices for Software Requirements Specifications.* IEEE Std 830-1998, IEEE Computer Society. October 20, 1998

Contact Information

Al Florence
florence@mitre.org
703 395 8700 – Cell
303 955 2286 – Home