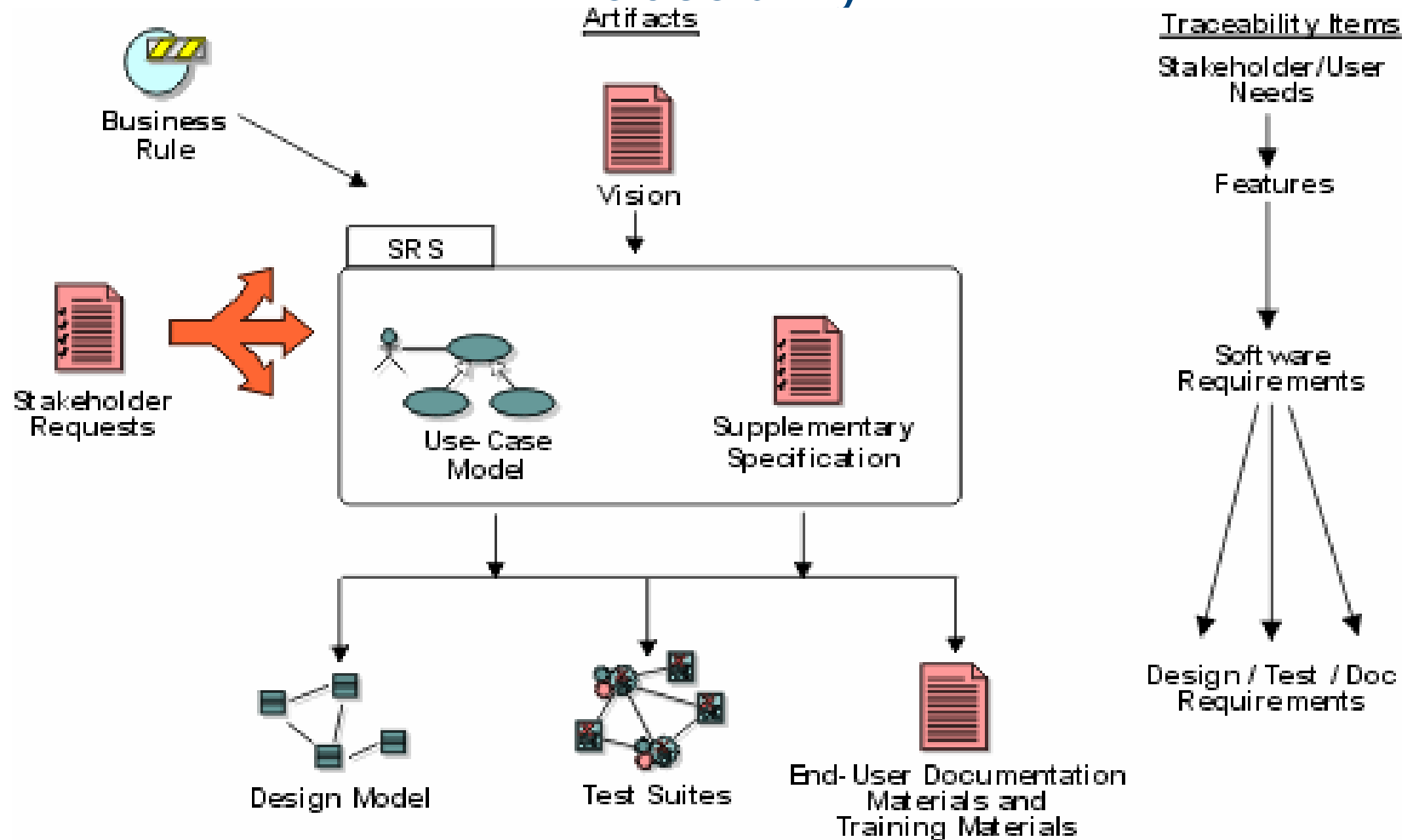


Requirements Specifications



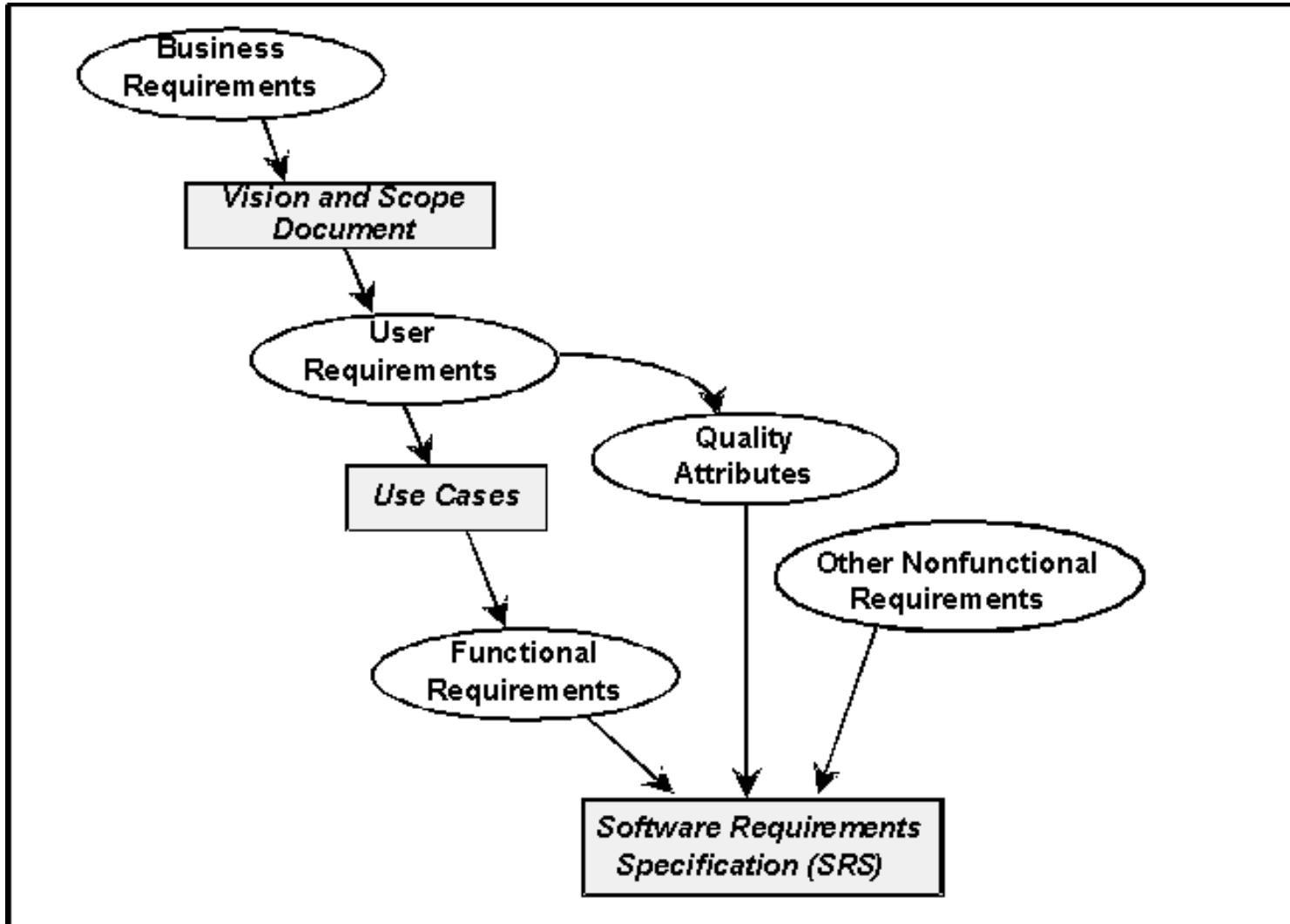
Types of Requirements and Requirements Traceability



[RUP]



The “Flow” of Requirements



Business Requirements

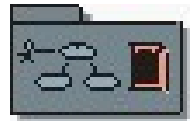
- Projects are launched with the belief that the new product will make the world a better place in some way for someone
- Business Requirements Document
 - Background: rationale and context for new product
 - Business opportunity
 - Market opportunity, internal savings, new capability for internal customers, etc.
 - Business objectives and success criteria
 - Measurable, quantitative benefits
 - Financial and non-financial
 - Customer or market needs
 - Business/Project risks

[Wiegiers Ch. 5]



Business Context

- Stakeholder profiles
 - Identify who will be impacted by success or failure of product/project
 - Identify and characterize their interest
 - How will they be impacted?
 - What role will they play in project?
- Project priorities
 - Constraints: limiting factors
 - Drivers: significant success objective with little flexibility for adjustment
 - Degree of freedom: a factor that can be balanced against constraints and drivers
- Operating environment
 - Use context and required availability, reliability, performance, usability, etc.



Software Requirements Specification

- The Software Requirements Specification (SRS) captures and organizes the complete software requirements for the system
 - A package containing all the use cases of the use-case model, plus Supplementary Specifications
- Basis of communication between all parties
 - Between the developers themselves
 - Between the developers and the external groups (users and other stakeholders)
- Formally or informally, it represents a contractual agreement between the various parties
 - If it is not in the SRS Package, then the developers shouldn't be working on it
 - If it is in the SRS Package, then the developers are accountable to deliver that functionality

[RUP]

Characteristics of a Well-Constructed SRS

- Correct
 - Every requirement contributes to the satisfaction of needs
- Complete
 - Contains all significant requirements, responses to all inputs and full labels and references
- Consistent
 - No subset of individual requirements is in conflict
- Unambiguous
 - Every requirement within it has only one interpretation
- Ranked for importance and stability
 - Identifier indicates its importance and stability
- Verifiable (Testable)
 - There exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement
- Modifiable
 - Changes can be made easily, completely, and consistently.
- Traceable
 - Facilitates backward and forward referencing

[RUP]



Software Requirements Specification Contents

- Use-case model (as Use Cases)
 - A survey of the use-case model
 - A collection of use case diagrams, use cases, actors, relationships, and Use-case package
 - Organize and structure use-case models into packages and sub-packages
- Supplementary specifications
 - Capture system requirements that are not readily captured in behavioral artifacts such as use-case specifications

[RUP]



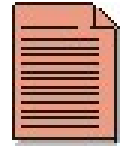
Survey of the Use-Case Model

- Write a Survey Description that includes
 - Which are the primary use cases of the system (the reason the system is built)
 - Typical sequences in which the use cases are employed by the users
 - Relationships between use cases (generalizes, extends, includes relations)
 - Package/Sub-package structure of the use cases
 - System delimitations – things that the system is not supposed to do
 - Important technical facts about the system
 - The system's environment, for example, target platforms and existing software
 - Specify functionality not handled by the use-case model

[RUP]



Supplementary Specifications



Supplementary Specifications

- Supplementary Specifications capture the system requirements that are not readily captured in the use cases of the use-case model
 - Legal and regulatory requirements
 - Applicable industry or organization standards
 - Quality attributes
 - Design and implementation requirements (constraints)
 - etc.
- These are requirements that apply to the system as a whole or to multiple use cases
 - For requirements specific to a use case, place them in a “supplementary requirements” section of that use-case specification

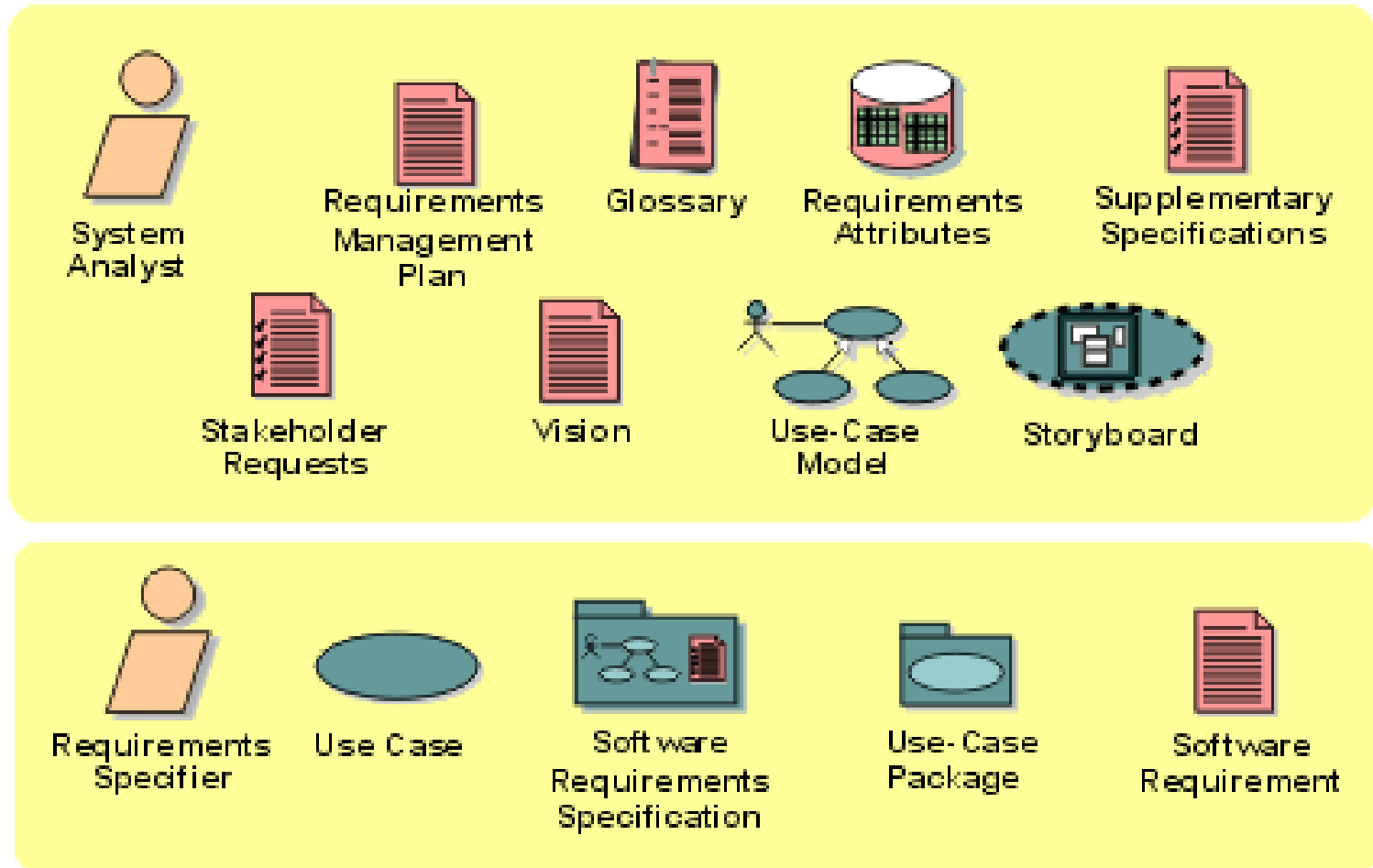
[RUP]



Other Requirements Artifacts



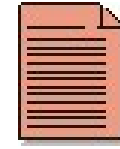
RUP Artifacts Overview



[RUP]



Glossary



Glossary

- The glossary defines important terms used by the project
- Provides a consistent set of definitions to help avoid misunderstandings
- Write the requirements and user documentation using glossary terms

[RUP]



Requirements Attributes

- Requirements attributes are information associated with a particular requirement providing a link between the requirement and other project elements, such as:
 - Priorities
 - Schedules
 - Status
 - Design elements
 - Resources
 - Costs
 - Risks

[RUP]



Example Requirements Attributes

- Tracking status
- Benefit
- Rationale
- Level of effort to implement
- Type and amount of each type of risk involved in implementing
 - Schedule risk
 - Resource risk
 - Technical risk
- Stability of the requirement
- Target release
- Assignment
- Marketing input
- Development input
- Revision history
- Location
- Reasons for change
- Inconclusive requirements

[RUP]



Requirement Writing Guidelines - 1

- Write complete sentences, using proper grammar.
- Use the active voice.
 - Use a format like:
 - <system/user> “shall” < action verb> <observable result>
 - e.g. “The system shall create a grade record.”
- State requirements in a consistent fashion.
 - Use terms consistently (as defined in a glossary).
 - Try to write all requirements at approximately the same level of detail and granularity.
- Avoid being wordy.
 - Requirements statements should be just long enough to convey the clear, correct and complete meaning of a requirement.



Requirement Writing Guidelines - 2

- Avoid use of conjunctions such:
 - “and”, “or”, “but”, ...
- Avoid vague terms like:
 - “user-friendly”, “simple”, “efficient”, “state-of-the art”, “robust”, “improve”, “optimize”, ...
- Avoid redundancy.
 - A requirement should appear only once in the SRS.
 - Uniquely label each requirement.
- Grouping requirements hierarchically can help make the functional requirements easier to understand.
 - See section 3.1.3 in Wiegers example SRS.



A Requirement Example

- 4.2.1 (High) The ICS shall ...
- 4.2.2 (High) The ICS shall provide for a class instructor (or a designated representative) to enter or change grades for students in the class, for a designated grading element.
 - 4.2.2.1 The ICS shall display a list of grading elements for the class.
 - 4.2.2.2 When the instructor selects a grading element, the ICS shall display a list of students, each with an field for entering or changing a grade.
 - 4.2.2.3 After the instructor enters or changes grades and submits them, the ICS shall store the grades in the student records.
 - 4.2.2.4 After submission of grades, the ICS shall display the grade records, for all students in the class.



Requirement Priorities

- Prioritizing requirements allows customer input into balancing schedule, cost and quality constraints and risks,
 - It helps manage customer expectations.
 - It allows developers to make “customer-oriented” decisions when development problems arise.
- Prioritizing should begin early and be complete at the time of agreement over the SRS.
- On a small project stakeholders can agree on priorities on an informal basis.
- For complex systems more formal systems may be needed. (see Table 14-2 in the text for a Prioritization Matrix).



Informal Priority Metric

High (essential)	Software not acceptable unless provided in the manner agreed, in the coming release.
Medium (conditional)	Desirable, but the product would not be unacceptable if absent in the coming release.
Low (optional)	Nice to have someday, if resources permit.



Requirements Status

Proposed	Based on elicitation and analysis, a software requirement is incorporated in the initial draft of the SRS.
Approved	Stakeholders have reviewed the requirement and given approval for its inclusion in the SRS.
Validated	The requirement has been reviewed as part of a formal inspection process, test scenarios have been developed that test the requirement, appropriate changes have been to the requirement, and key stakeholders have signed off on the SRS containing the requirement.



Let Standards Guide You

- Standards collect the best practices and experience of the industry into consensus techniques and guidelines
- Some organizations or customers require standards compliance
- See
 - IEEE Standard 1223-1998: IEEE Guide for Developing System Requirements Specifications
 - IEEE Standard 830-1998: IEEE Recommended Practice for Software Requirements Specifications



Supplementary Requirements



The following slides are for your reference—
to remind you of possible non-functional
requirements to capture in the Supplementary
Requirements Specifications



Characterizing Requirements: FURPS+

- One way to categorize requirements: FURPS+
 - Functionality
 - Usability
 - Reliability
 - Performance
 - Supportability
 - +
 - Design constraints
 - Implementation requirements
 - Interface requirements
 - Physical requirements
- Quality requirements
“-ilities”
- Non-functional
Requirements
-

Another Way to Characterize Requirements: CRUPIC STMPL

- Operational categories
 - Capability
 - Reliability
 - Usability
 - Performance
 - Installability
 - Compatibility
 - Customer and user requirements
 - Mostly visible at run-time
- 
- Developmental categories
 - Supportability
 - Testability
 - Maintainability
 - Portability
 - Localizability
 - Developer and support requirements
 - Mostly visible at build-time
- 

[RUP]



Usability Requirements

- Learn-ability
- Remember-ability
- Efficiency in use
- Reliability in use
- User satisfaction
- etc.

[RUP]



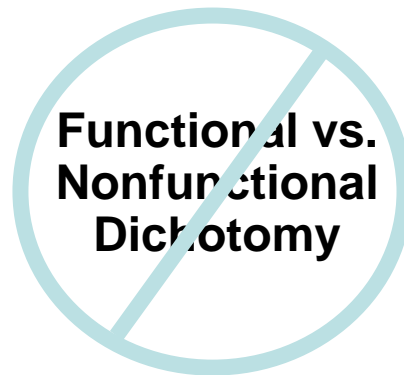
Quality Attributes

From Len Bass, Paul Clements, Rick
Kazman, *Software Architecture in Practice*,
Addison-WesleyLongman, 1998



“Every good quality is noxious if unmixed.”
(Ralph Waldo Emerson)

- Functionality is the primary, but not the exclusive, quality of a system
 - the mapping of a system’s functionality onto software structures determines the architecture’s support for qualities
 - functionality and other qualities must be designed in from the start
 - cannot go back and add in quality



Two Broad Categories

- Two broad categories
 - Observable via execution (run-time behavior)
 - Not observable via execution (build-time behavior)
- Qualities are attributes of what?
 - The system
 - The business
 - The architecture
- Quality must be considered during all life-cycle phases
 - Different qualities manifest themselves differently during the phases
 - Architecture is critical to realization of many qualities
 - Not all qualities are architectural
 - Some qualities are architectural *and* non-architectural
- Qualities are rarely independent \Rightarrow trade-offs

[Bass, Clements, Katzman, *Software Architecture in Practice*]



System Runtime Quality Attributes

- Performance
 - responsiveness: throughput and latency
 - primarily driven by inter-component interaction
 - analyzed via stochastic queueing and workload models
 - historically, the primary driving factor in architecture (but no longer alone)
- Security
 - denying service to unauthorized usage
 - assuring service to authorized usage in spite of unauthorized attempts at denying services
 - architectural solutions
 - special security components that coordinate the interaction of the other components

[Bass, Clements, Katzman, *Software Architecture in Practice*]



System Runtime Quality Attributes (continued)

- Availability
 - the proportion of time the system is up and running
 - realibility (time between failures) and time to repair
 - reliability: fault-tolerance, error detection and handling
 - repair: fault isolation, ease of component replacement or modification
- Functionality
 - ability to perform the task required
 - non-architectural, but allocation of functionality to structure interacts with architectural qualities
- Usability
 - learnability
 - efficiency
 - memorability
 - architecture: information availability and organization; efficiency
 - error avoidance
 - error handling
 - satisfaction

[Bass, Clements, Katzman, *Software Architecture in Practice*]



System Build-Time Quality Attributes

- Modifiability (maintainability)
 - extend or change capabilities
 - delete unwanted capabilities
 - adapt to new operating environments
 - restructure
 - architectural issue: scope of change
 - local component, a few components, or change in underlying architectural style
- Portability
 - ability to run under different computing hardware/software environments
 - approach: isolate platform-specific concerns in a portability layer

[Bass, Clements, Katzman, *Software Architecture in Practice*]



System Build-Time Quality Attributes (continued)

- Reusability (integrate-ability, modifiability)
 - system structures and/or components can be used in other systems
 - architecture: relatively small, loosely coupled components
- Integrability (\equiv integrate-ability)
 - ability to make components work together
 - architecture
 - external component complexity
 - interaction mechanisms and protocols
 - clean separation of concerns
 - completeness of specification
 - interoperability: integrability of components in one system with those in another system
- Testability (controllability, observability)
 - separation of concerns, information hiding, ‘uses’ structure for incremental development



Business Qualities

- Time to market
 - incorporation of existing components
- Cost
 - reuse, technical maturity of organization
- System lifetime
 - invest in modifiability and portability
- Targeted market
 - functionality, portability, performance, reliability, usability, product line architecture
- Rollout schedule
 - modifiability (flexibility, customizability, extensibility)
- Use of legacy systems
 - integrability

[Bass, Clements, Katzman, *Software Architecture in Practice*]

