Quality Attributes
Traditional Classification of Requirements

- Functional
- Non-Functional (Quality Attributes)

A popular software myth: first we build a software that satisfies functional requirements, then we will add or inject non-functional requirements to it.

- This idea leads to loss of resources and finally poor quality.
- So we should design for qualities from the very beginning (architecture level).
Functionality and Architecture

- Functionality and quality attributes are orthogonal [in theory]. But not all qualities are achievable to any level desired with any functionality.
- Functionality may be achieved in many ways (it is not so architectural.)
- Architecture is a means of achieving quality attributes by structuring functionality into elements.
Architecture and Qualities

• Achieving qualities must be considered throughout design (including SA), implementation, and deployment.

• Qualities have both architectural and non-architectural aspects. For example
  – In usability: selecting form elements vs. supporting undo operation
  – Performance: amount of communication among components vs. algorithms
Architecture and Qualities

• Quality attributes are not independent and may be achieved in isolation.

• Positive Correlation; e.g.
  – Modifiability and Buildability (in many cases)

• Negative Correlation (conflict); e.g.
  – Reliability vs. Security
  – Performance vs. All Other Qualities
Classification of Quality Attributes

1. Qualities of the system: availability, modifiability, performance, security, testability, and usability.
2. Business qualities (such as time to market) that are affected by the architecture.
3. Architecture qualities, such as conceptual integrity.
System Quality Attributes

• System quality attributes have been of interest to the software community at least since the 1970s

• Shortcoming of the previous work:
  – The definitions for an attribute are not operational.
    • Modifiability with regards to which aspect?
  – Which quality a particular aspect belongs to.
    • Is a system failure an aspect of availability, an aspect of security, or an aspect of usability?
  – Each attribute community has developed its own vocabulary.
    • Performance community events, security community attacks, availability community failures, and usability community user input may actually refer to the same occurrence.
Classifications of System Quality Attributes

- Observable via Execution
  - e.g. performance and security

- Not observable via execution
  - e.g. modifiability and testability

- The categories are totally independent (orthogonal), although members of the second category indirectly affect members of the first.

- Non-observable qualities are important too. (sometimes even more important!!)
Quality Attribute Scenarios

- Is the solution to the stated problems.
- A QAS is a quality-attribute-specific requirement, that consists of:
  1. Source of stimulus: actuator; e.g. a human or computer system
  2. Stimulus: event.
  3. Environment: the condition under which the stimulus occurs; e.g. system is overloaded.
  4. Artifact: pieces of system that is stimulated.
  5. Response: desired reaction
  6. Response measure: response should be measurable in some fashion so that the requirement can be tested.

- Scenarios may be general or concrete (for specific system)
Quality attribute parts
Example: Availability General Scenario
Example: Availability Concrete Scenario

Source: External to System
Stimulus: Unanticipated Message
Environment: Normal Operation
Artifact: Process
Response: Inform Operator Continue to Operate
Response Measure: No Downtime
Modifiability Concrete Scenario

Source: Developer

Stimulus: Wishes to Change the UI

Environment: At Design Time

Artifact: Code

Response: Modification Is Made with No Side Effects

Response Measure: In Three Hours
Performance

• **Performance** is about *timing*:
  – interrupts, messages, requests from users, or the passage of time
  – basically: how long it takes the system to respond when an event occurs

• **Complexity**:
  – number of event sources & arrival patterns
  – this characterization is the language to construct general performance scenarios
Performance Scenarios

• **Performance Scenarios:**
  – Start with a request for service arriving at the system. Satisfying the request consumes resources. Usually events are handled in parallel.

• **Arrival Patterns:**
  – *periodic* - most often seen in real-time systems
  – *stochastic* - events arrive according to some probabilistic distribution
  – *sporadic* - a pattern that can’t be represented by either
Sample performance scenario

Source: Users

Stimulus: Initiate Transactions

Environment: Under Normal Operations

Artifact: System

Response: Transactions Are Processed

Response Measure: With Average Latency of Two Seconds
Usability

• How easy is it for the user to accomplish a desired task & what kind of user support does the system provide.
  – Learning system features
  – Using a system efficiently
  – Minimizing the impact of errors
  – Adapting the system to user needs
  – Increasing confidence & satisfaction
Sample usability scenario

Source: Users

Stimulus: Minimize Impact of Errors

Environment: At Runtime

Artifact: System

Response: Wishes to Cancel Current Operations

Response Measure: Cancellation Takes Less Than One Second
Sample testability scenario

Source: Unit Tester

Stimulus: Performs Unit Test

Environment: At the Completion of the Component

Artifact: Component of the System

Response: Component Has Interface for Controlling Behavior and Output of the Component Is Observable

Response Measure: Path Coverage of 85% Is Achieved within Three Hours
Sample security scenario

Source: Correctly Identified Individual
Stimulus: Tries to Modify Information
Environment: Under Normal Operations
Artifact: Data within the System
Response: System Maintains Audit Trail
Response Measure: Correct Data Is Restored within a Day
Notes on Scenarios

• Concrete scenarios role for quality attribute requirements is similar to use cases role for functional requirements.

• A collection of concrete scenarios can be used as the quality attribute requirements for a system.

• One of the uses of general scenarios is to enable stakeholders to communicate.
System Quality Attributes

- Availability (related to Reliability)
- Modifiability (includes Protability and Reusability, Scalability)
- Performance
- Security
- Testability
- Usability (includes Self-Adaptability and User-Adaptability)
Business Qualities

- Time to market
- Cost and benefit
- Predicted lifetime of the system
- Targeted market
- Rollout schedule
- Integration with legacy systems
Qualities of the Architecture

• Conceptual Integrity
  – Conceptual integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas. [Brooks 75]

• Correctness and Completeness

• Buildability

There should be a way to evaluate SA
Specifying Requirements

Functional Requirements

Quality Attributes Requirements

Use Cases

Concrete Scenarios