System Quality Attributes

- Two broad categories of quality attributes against which a system can be measured at the architectural level:
  - Observable via execution
  - Not observable via execution

- Architecture is crucial to the realization of many of the qualities of interest in a system, and these qualities should be designed in and evaluated at the architectural level.

- Some qualities are not architecturally sensitive, and attempting to achieve these qualities via architectural means is not fruitful.

Performance (1)

- Performance refers to the responsiveness of the system:
  - The time required to respond to events, or
  - The number of events processed in some interval of time

- Performance qualities are often expressed by:
  - The number of transactions per unit time, or
  - The time a system takes to complete a transaction

- Performance is often a function of how much communication and interaction there is between the components of the system—clearly an architectural issue.

Performance (2)

- Performance can be analyzed by looking at:
  - The arrival rate and distributions of service requests
  - Processing time
  - Queue sizes, and
  - Latency: the rate at which requests are serviced

- Performance can be simulated by building a stochastic queuing model of the system based upon anticipated workload scenarios.

- Performance has been the driving factor in system architecture.
Security

- Security is a measure of the system’s ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users.
- Types of threads that might be made to system
  - Denial of service: This attack prevents a targeted system from providing, receiving, or responding to network services by flooding the target with connection requests or queries
  - IP source address spoofing: This attack attempts to gain access to the target by assuming the identity of a host trusted by the target.

Availability

- Availability measures the proportion of time the system is up and running.
- It is measured by the length of time between failures as well as by how quickly the system is able to resume operation in the event of failure.
  \[ \alpha = \text{mean time to failure} \div (\text{mean time to failure} + \text{mean time to repair}) \]
- Closely related is reliability that is usually measured with mean time to failure.

Functionality

- Functionality is the ability of the system to do the work for which it was intended.
- Performing a task requires most of the system’s components work in a coordinated manner to complete the job.
- Functionality is largely non-architectural in nature.
- Software architecture constrains the allocation of functionality to structure when other quality attributes are important.

Usability

- Usability can be broken down into the following areas:
  - Learnability: how quick and easy is it for a user to learn to use the system’s interface?
  - Efficiency: does the system respond with appropriate speed to a user’s request?
  - Memorability: can the user remember how to do system operations between uses of the system?
  - Error avoidance: does the system anticipate and prevent common user errors?
  - Error handling: does the system help the user recover from errors?
  - Satisfaction: does the system make the user's job easy?
Usability (2)

- Non-architectural strategies
  - Making a user interface clear and easy to use
  - Using familiar metaphors, standards, and interface conventions
  - Screen layout, color and shading etc
- Architectural strategies
  - Making sure that the right information is available to the user at the right time
  - Making sure that the user's instructions get to the right destination in the system
  - The above information must flow to and across the appropriate components via the connectors

Non-Runtime System Quality Attributes

- Modifiability
- Portability
- Reusability
- Integrability
- Testability

Modifiability (1)

- Modifiability can be broadly categorized as follows:
  - Extending or changing capabilities: adding new functionality, enhancing existing functionality, or repairing bugs
  - Deleting unwanted capabilities: to streamline or simplify the functionality of an existing system, perhaps to deliver a less-capable version of a product to a wider customer base
  - Adapting to new operating environment: processor hardware, input/output devices, and logical devices
  - Restructuring: rationalizing system services, modularizing, optimizing, or creating reusable components

Modifiability (2)

- Modifiability may be the quality attribute most closely aligned to the architecture of a system
  - The ability to make change quickly and cost effectively follows directly from the architecture:
  - Modifiability is largely a function of the locality of any change
  - Making a widespread change is more costly than making a change to a single component
  - Since the architecture defines the components and the responsibilities of each, it also defines the circumstances under which each component will have to change
  - An architecture classifies all possible changes into three categories
    - Changes within one component
    - Changes within more than one components
    - Change to the underlying architectural style

Portability

- Portability is the ability of the system to run under different computing environment
- A system is portable means that all of the assumptions about any particular computing environment are confined to one component (or at worst, a small number of easily changed components)
- A portability layer is used to encapsulate any platform-specific considerations
- A portability layer is a set of software services that gives application an abstract interface to its environment
  - This interface remains constant even though the implementation of that layer changes as the system is ported from environment to environment
  - A portability layer results from a straightforward application of the design principle of information hiding

Reusability

- Reusability means that designing a system so that the system's structure or some of its components can be reused again in future application.
- Reusability is related to software architecture in that
  - Architectural components are the units of reuse, and
  - How reusable a component is depends on how tightly coupled it is with other components
- Reusability is actually a special case of modifiability
Integrability

Integrability is the ability to make the separately developed components of the system work correctly together.

Integrability depends on:
- The external complexity of the components
- Their interaction mechanisms and protocols
- The degree to which responsibilities have been clearly partitioned
- The interfaces to the components

Testability

Software testability refers to the ease with which software can be made to demonstrate its faults via testing.

A system's testability relates to several structural issues:
- Its level of architectural documentation
- Its separation of concerns, and
- The degree to which the system uses information hiding

Business Quality Attributes

- Time to market
  - Using pre-built components or reusable components
  - Cost
  - Different architectures will yield different costs
  - Projected lifetime of the system
    - A modifiable, extensible product is more likely to survive longer in the marketplace
  - Targeted market
    - For general-purpose software, portability and functionality are key to market share
    - For a large but specific market, a product-line approach should be considered
  - Rollout schedule
  - Extensive use of legacy systems

Quality of the Architecture

- Conceptual integrity is the underlying theme or vision that unifies the design of the system at all levels
- Correctness and completeness are essential for the architecture to allow the satisfaction of all the system's requirements and run time resource constraints
- Buildability is the quality of the architecture that allows the system to be completed by the available team in a timely manner
  - It refers to the ease of constructing a desired system, and
  - It is usually measured in terms of cost and time

Architectural Means for Achieving Quality

- Choose a software architecture that accommodates changes
- The architecture must handle:
  - Modifications introducing new computing environment
  - New platform deployments
  - New operator interfaces
  - Addition of the new sensing devices
- The goal is to produce a system structure such that each anticipated change will only affect a small number of system components

Capture Possible Changes

- A wise architect will capture the changes that should be accounted for in the design:
  - Changes anticipated for this system or experienced by similar system
  - Changes anticipated by mining the requirements specifications for areas of uncertainty or ambiguity
  - Decrease in functionality brought about because of a management decision to field a subset of the system
Architecture Design

[Reference: Hofmeister et al]

Architectural Views

- We use the following views
  - Conceptual View
  - Module View
  - Code View
  - Execution View

Conceptual View

- Shows how the functionality required of the system is mapped on to architecture elements called conceptual components and connectors.

\[ C_v = R \rightarrow \{ C_m, C_n \} \]

- \( R \) = Functional Requirements
- \( C_m \) = Components
- \( C_n \) = Connectors

- The conceptual view focuses on the domain

Conceptual View Questions

- How does the system fulfill the requirements?
- How are COTS (Commercial off-the-shelf) components to be integrated?
- How do they interact with the rest of the system?
- How is domain-specific hw/sw incorporated?
- How is functionality partitioned into product releases?
- How does the system incorporate portions of prior generations of the product?
- How will it support future generations?

Module View

- Shows how components and connectors from the conceptual view are mapped to subsystems and modules

\[ M_v = C_v \rightarrow \{ S_s, M \} \]

- \( S_s \) = Subsystems, \( M \) = Modules

- Shows how the conceptual solution can be realized with today’s software platforms and technologies.
Module View Questions
- How is the product mapped to the software platform?
- What system support/services does it use?
  - Exactly where?
- How can dependencies between modules be minimized?
- How can reuse be maximized?
- How can the product be insulated from changes to COTS software, platform or standards?

Execution View
- Shows how modules are mapped to elements provided by the runtime platform
  \[ E_v = M_v \rightarrow H \]
  - \( H \) = Hardware Architecture
- Defines runtime entities such as memory usage, execution flow of control
- Runtime control flow could be different from logical (conceptual) control flow

Execution View Questions
- How does the system meet its performance requirements?
- How can one balance resource usage (load balancing, for instance)?
- How can one achieve necessary concurrency, replication, distribution etc.?
- How can the impact of changes in the runtime platform be minimized?

Code View
- Shows the mapping from the execution view to deployment components (for example, executables, runtime libraries)
- Shows how modules from Module View are mapped to source components, and
- how deployment components are produced from source components
  \[ K_v = M_v \rightarrow \{ C_v \} \rightarrow \{ C_y \} \]

Code View Questions
- How can time and effort for product upgrades be reduced?
- How should product versions and releases be managed?
- How can build time be reduced?
- What tools are needed to support the development environment?
- How are integration and testing supported?

Why different views?
- Separation of concerns
- Addresses architectural concerns at different levels of abstractions.
- The architect can address high priority decisions first, and can more easily analyze the design trade offs.
The four views

**Conceptual View**
- Components, connectors, configuration
- Module constraints, modules
- New module partitioning, runtime entities
- Changes to runtime entities

**Module View**
- Components, connectors, configuration
- Runtime entities, modules, subsystems
- New module partitioning, runtime entities

**Code View**
- Components, connectors, configuration
- Runtime entities, modules, subsystems
- Changes to runtime entities

**Execution View**
- Components, connectors, configuration
- Module constraints, modules
- New module partitioning, runtime entities
- Changes to runtime entities

**Source Code**

Design Tasks of the Conceptual View

Global Analysis: Organisational factors, Technological factors, Product factors
- Develop strategies

Central Design Tasks:
- Conceptual components, Conceptual connectors, Conceptual configuration, Global evaluation
- Resource budgeting

To Execution View

To Module View

Design Tasks of the Module View

Global Analysis: Organisational factors, Technological factors, Product factors
- Develop strategies

Central Design Tasks:
- Modules, Layers, Global evaluation
- Interface Design

To Code View

To Execution View

Design Tasks of the Code View

Global Analysis: Organisational factors, Technological factors, Product factors
- Develop strategies

Central Design Tasks:
- Source components, Intermediate components, Deployment components, Global evaluation
- Build script design, Configuration, Management Design

To Execution View

From Conceptual View

To Module View

From Module View

To Code View

Source Code

Design Tasks of the Execution View

Global Analysis: Organisational factors, Technological factors, Product factors
- Develop strategies

Central Design Tasks:
- Runtime entities, Communication paths, Execution configuration, Global evaluation
- Resource allocation

Final Design Task: Resource allocation

To Code View