Understanding Quality Attributes

主讲教师： 王灿
Email: wcan@zju.edu.cn
TA: 李奇平 liqiping1991@gmail.com
Course FTP: ftp://sa:sa@10.214.51.13

Understanding Quality Attributes

Architecture and quality attributes

Quality Attribute Scenarios

Guiding Quality Design Decisions

Why Software Projects Fail?

- More than 80% of software projects are delivered late, more than half do not include needed features, and cost overruns of 15% or higher are commonplace.

  ---The Standish Group's Chaos Report 2004

- Systems are frequently redesigned, not because they are functionally deficient, but because they are difficult to maintain, port, or scale, or are too slow, or have been compromised by network hackers.
  - Performance, security, availability…
  - On budget, on schedule…

Architecture and Requirements

- An architecture is shaped by:
  - Functional requirements
    - What the system must do, its behavior and its reaction to runtime stimuli
  - Quality attribute requirements
    - How the functionality shall be delivered
  - Constraints
    - Design decision already been made, e.g. using a certain programming language, or adoption of a specific pattern

Functionality

- Functionality means the ability of the system to do the work intended
- Functionality alone does not determine architecture
  - Given a set of functionality, you can have them divided up in various ways and assigned them to different architectural elements; hence forming different architecture.
- If functionality were the only concern we could have implemented a system as one large module in Cobol or even assembly language.

Quality Attributes and Functionality

- Quality attributes do not stand on their own. They pertain to the function of the system
- E.g.
  - A functional requirement: "When the user presses the green button, the Options dialog appears."
  - Corresponding quality attribute:
    - Performance QA: how quickly the dialog will appear
    - Availability QA: how often this function will fail, and how quickly it will be repaired
    - Usability QA: how easy it is to learn this function

Architecture and Quality Attributes

- **It is the mapping of a system's functionality onto software structures that determines the architecture's support for qualities.**
  - Quality attributes dictate a software system's architecture.
- No quality attribute is entirely dependent on design or other phases.
  - Satisfactory inclusion of quality attributes means you must get the big picture (architecture) and the details (implementation) right!

Performance: Architectural. VS Non-Architecture Aspects

- Non-architectural aspects of Performance:
  - Choice of algorithms
  - How these algorithms are coded

- Architectural aspects of Performance:
  - Communication between components
  - Partially on partitioning of functionality
  - Allocation of resources

Modifiability: Architectural. VS Non-Architecture Aspects

- Non-architectural aspects of Modifiability :
  - Coding techniques used within a module
- Architectural aspects of Modifiability:
  - How functionality is partitioned

Usability: Architectural. VS Non-Architecture Aspects

- Non-architectural aspects of usability:
  - Clear interface
  - What font? ...
- Examples of architectural aspects of usability
  - Whether the system provides an “undo’ capability to the user?
  - Can we – reuse data previously entered?
  - These are architectural because they are going to require the cooperation of several elements.

Architecture and Quality Attributes (Revisited)

- Architecture is critical to achieving quality attributes.
  - Architecture alone cannot achieve these qualities.

- In a complex system, quality attributes can never be achieved in isolation
  - The achievement of one attribute may negatively (or perhaps positively) influence the achievement of another

Understanding Quality Attributes

Architecture and quality attributes

Quality Attribute Scenarios

Guiding Quality Design Decisions

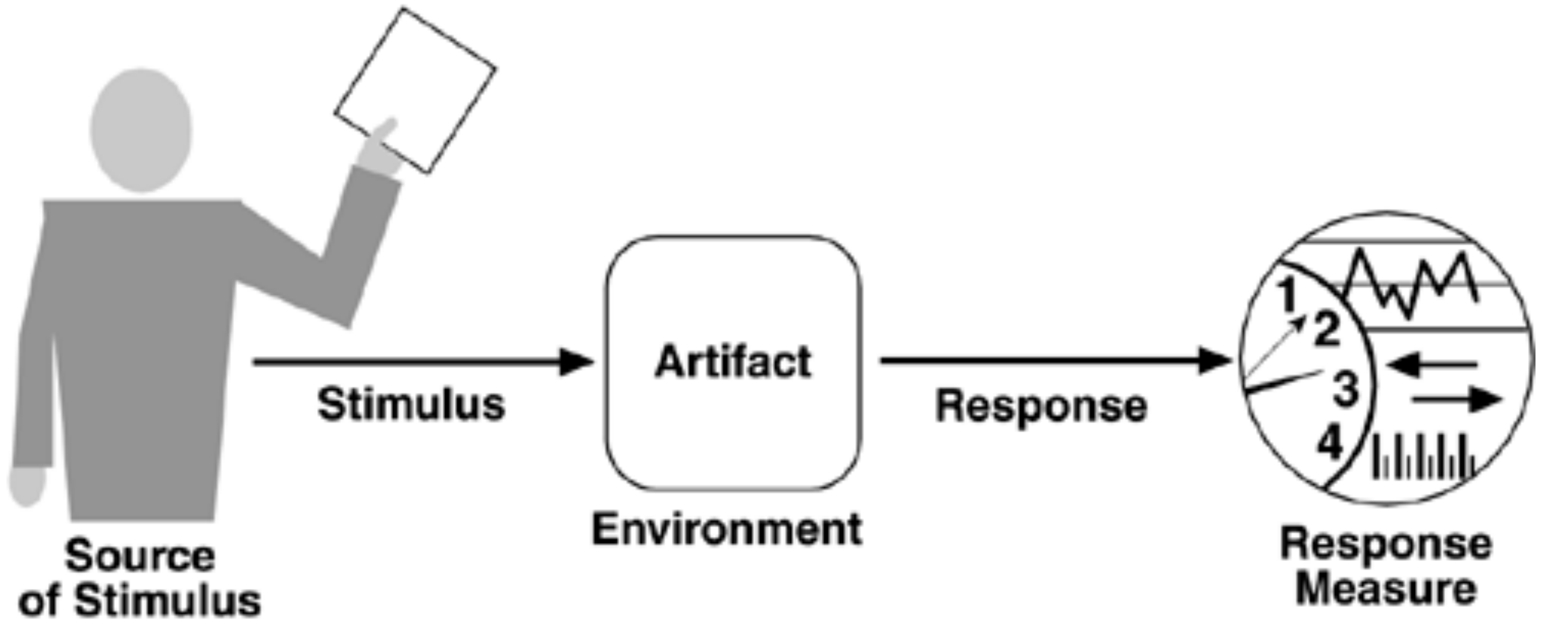
Types of Quality Attributes

- Qualities attributes describing system properties at runtime
  - Availability
  - Interoperability
  - Performance
  - Security
  - Usability
- Qualities attributes describing system properties during development
  - Modifiability
  - Testability

Presenting System Quality Attributes

- Problems in presenting system quality attributes
  - Non-testable definitions
    - E.g. high performance, modifiable system
  - Overlapping attribute concerns
    - E.g. A system failure from DOS attack (Availability? Security? Usability?)
  - Non-uniform vocabulary
    - E.g. attacks, failures, events

Quality Attribute Scenarios



QAS Parts (1)

- Stimulus: an event arriving at the system.
  - Performance: an event to be processed
  - Usability: a user operation
  - Security: an attack to the system
  - Modifiability: a request for a modification
  - Testability: completion of a development phase
- Stimulus source
  - Stimulus from different source may be treated differently by the system
  - e.g. a request from a trusted user in a security scenario

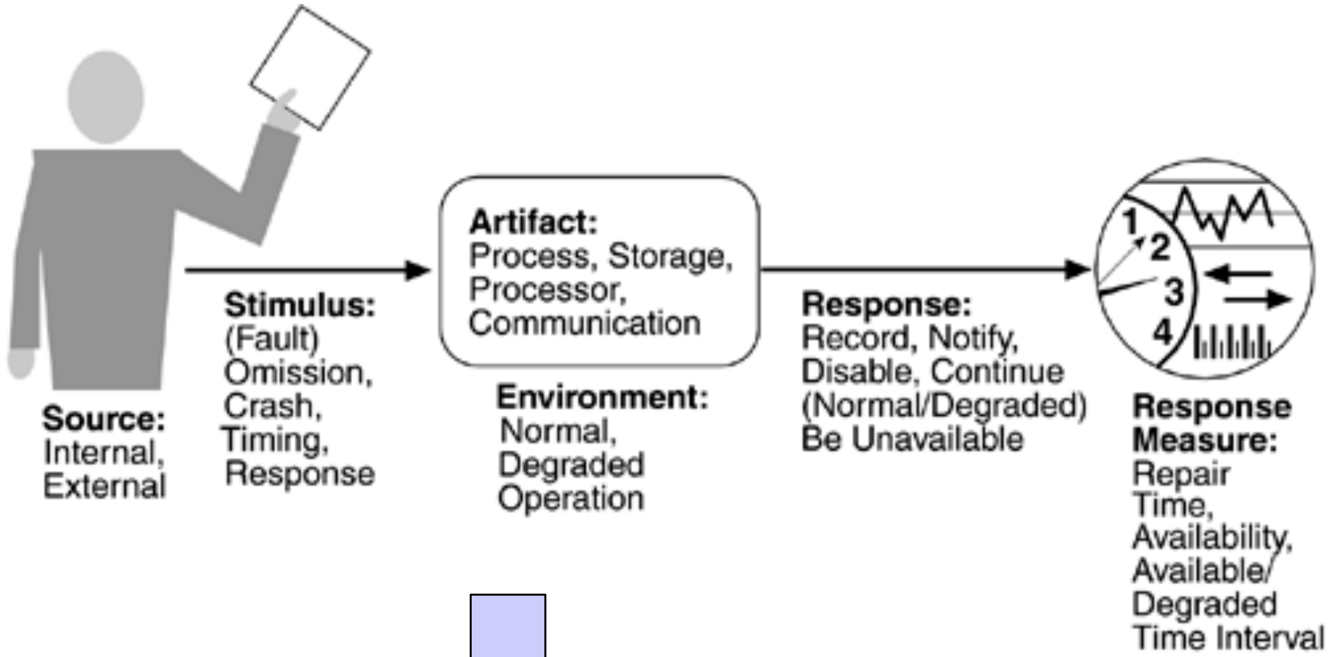
QAS Parts (2)

- Response: how the system responds to the stimulus
  - For runtime qualities, a set of system responsibility
    - e.g. in a performance scenario, an event arrives and the system processes the event and generates a response
  - For development-time qualities, a set of developer responsibility
    - e.g. in a modifiability scenario, a modification request arrives and the developers implement, test and deploy the modification
- Response measure
  - Performance: latency or throughput
  - Modifiability: labor or time spent

QAS Parts (3)

- Environment: the set of circumstances in which the scenario takes place
  - E.g. a request for a modification that arrives at the design stage VS. at the implementation stage
  - System performance during normal operation VS. overloaded
- Artifact: the portion of the system to which the requirement applies
  - Usually the entire system, with some occasional exception

General Scenarios VS. Concrete Scenarios



General scenarios

- An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and continues to operate with no downtime.

System specific scenarios

Tactics

- A tactic is a design decision that influences the control of a quality attribute response
- Tactics VS. Patterns
  - The focus of a tactic is on a single quality attribute response, i.e. no consideration of tradeoffs
  - In contrast, tradeoffs are built into patterns
  - While patterns are difficult to apply, tactics are more accessible options for architects.
  - Architects can use tactic to construct patterns

Example Tactics

- An example tactic is:
  - “introduce redundancy” to increase availability
    - this would require “synchronization” also.
- Tactics can be refined
  - E.g. refine “schedule resources” with a “short-job-first” implementation
- Tactics are context-dependent
  - E.g. “manage-event-rate” is sometimes not applicable in a real-time system

Understanding Quality Attributes

Architecture and quality attributes

Quality Attribute Scenarios

Guiding Quality Design Decisions

Architectural Design Decision

- Seven important categories of design decisions in creating a software architecture
  1. Allocation of responsibilities
  2. Coordination model
  3. Data model
  4. Management of resources
  5. Mapping among architectural elements
  6. Binding time decisions
  7. Choice of technology

Allocation of Responsibilities

- Identifying the important responsibilities
  - System functions
  - Architectural infrastructure
  - Satisfaction of quality attributes
- Determining how these responsibilities are allocated to non-runtime and runtime elements
  - Modules
  - Components & connectors
- Strategies
  - functional decomposition
  - modeling real-world objects
  - grouping

Coordination Model

- Software works by having elements interact with each other through coordination model. Decisions about the coordination model include:
  - Identifying the elements of the system that must coordinate, or are prohibited from coordinating
  - Determining the properties of the coordination
  - Choosing the communication mechanisms
    - Stateful VS. stateless, synchronous VS. asynchronous, guaranteed versus nonguaranteed delivery etc.

Data Model

- Data model concerns representation and interpretation of data
  - Choosing the major data abstractions, their operations, and their properties
  - Compiling metadata needed for consistent interpretation of the data
  - Organizing the data

Management of Resources

- Hard resources VS. soft resources
  - Identifying the resources that must be managed and determining the limits for each
  - Determining which system element(s) manage each resource
  - Resource sharing and arbitration.
  - Determining the impact of saturation on different resources

Mapping among Architectural Elements

- Two types of mapping in an architecture design: (1) between different types of architecture structures; (2) between software elements and environment elements
  - The mapping of modules and runtime elements to each other
  - The assignment of runtime elements to processors
  - The assignment of items in the data model to data stores
  - The mapping of modules and runtime elements to units of delivery

Binding Time Decisions

- Allow variation to be bound at different times in the software life cycle, e.g.
  - Build-time selection of modules via a parameterized makefile
  - Runtime negotiation of protocols
  - Plug & play for resource management
  - An app store providing appropriate version of app for a smartphone

Choice of Technology

- Considerations for an architect to choose a suitable technology to implement architectural decisions:
  - Available technologies
  - Available tools to support this technology choice (IDEs, simulators, testing tools, etc.)
  - Extent of internal familiarity as well as the degree of external support
  - Side effects of choosing a technology
  - Whether a new technology is compatible with the existing technology stack

Reading Assignment

- Read Chapter 5 of the textbook.