

Software Design and Architecture

[Quality Attributes] – Chapter 03, L01

Lecture Outlines

- Architecture and Requirements
- Quality Attribute Considerations
- Quality Design Decisions
- Major Quality Attributes

Architecture and Requirements

- System requirements can be categorized as:
  - ✓ **Functional requirements:**
    - State what the system must do, usually in the form of "system shall do <requirements>".
    - Functional requirements define **specific behavior** or function
    - The plan for implementing functional requirements is produced in the system detailed design.
  - ✓ **Non-functional requirements or Quality attributes (QA)):**
    - Specify criteria that can be used to judge the operation of the **system as a whole** rather than specific behavior
    - The plan for implementing non-functional requirements is detailed in the system architecture

Architecture and Requirements

- Achieving quality attributes must be considered throughout design, implementation, and deployment.

- Architecture, by itself, is unable to achieve qualities. It provides the foundation for achieving quality, but this foundation will be to no use if attention is not paid to the details in later phases of software development.

Architecture and Requirements

- Architectural documents are used by a variety of system stakeholders:
  - ✓ E.g. Developers, managers, sales, testers, support, maintenance, and customers

- Each stakeholder will have their own opinion about what non-functional requirement matters the most:
  - ✓ e.g. The development team will care about maintainability more than the customer
  - ✓ e.g. Quality assurance team will be more interested in the testability of the application
  - ✓ What will the end user be interested in???

Quality Attribute Considerations

- Quality attributes are constraints on the manner in which the system implements and delivers its functionality (how the system achieves its functionality).
- Extra: Quality attributes are used as criteria for evaluating the performance and effectiveness of the system during and after development.
  - ✓ i.e. Efficiency, availability, scalability, adaptability, security, dependability, etc. …
- **Example:** assume the following functional requirement *"when the user presses the green button the Options dialog appears"*:
  - ✓ A performance QA might describe how quickly the dialog will appear
  - ✓ An availability QA might describe how often this function will fail
  - ✓ A usability QA might describe how easy it is to learn this function.

Quality Attribute Considerations – Scenarios

- Quality attributes can have unstable definition or an overlapping concerns, **Example**:
  Is a system failure due to a denial of service attack an aspect of availability, performance, security, or usability?

- Extra: A **Denial-of-Service (DoS) attack**:
  - ✓ Availability → system is down.
  - ✓ Performance → system becomes very slow.
  - ✓ Security → the attack is a security breach.
  - ✓ Usability → users can't access the service.

- Here, the same problem "overlaps" multiple attributes.

- A solution for this is (unstable definitions and overlapping concerns) is to use ***quality attribute scenarios*** as a means of characterizing quality attributes.

Extra:

Solution: Quality Attribute Scenarios

- Instead of **vague** terms, define concrete scenarios that describe:
  - Stimulus (what happens),
  - Response (what the system should do),
  - Measure (how success is judged).

◆ **Real Example**

- Attribute: **Availability**
  - Scenario: If the payment server crashes during a transaction (**stimulus**),
  - The system should automatically switch to a backup server within 2 seconds (**response**),
  - With 99.99% uptime guaranteed (**measure**).
- This way, the attribute is **clear, testable, and not overlapping**.

Quality Attribute Considerations – Scenarios

- A quality attribute scenario has the following parts:
  1. **Stimulus** – a condition that needs to be considered
  2. **Source of stimulus** - (e.g., human, computer system, etc.)
  3. **Environment** - what are the conditions when the stimulus occurs?
  4. **Artifact** – what elements of the system are stimulated.
  5. **Response** – the activity undertaken after arrival of the stimulus
  6. **Response measure** – when the response occurs it should be measurable so that the requirement can be tested.

Extra: 6 Steps for Using Quality Attribute Scenarios

1. **Source** → Who triggers the action (e.g., user, attacker, system).
2. **Stimulus** → The action or event that happens (e.g., request, failure, attack).
3. **Environment** → The situation in which it happens (e.g., under load, during maintenance).
4. **Artifact** → The part of the system affected (e.g., database, server, UI).
5. **Response** → How the system reacts (e.g., log error, retry, deny access).
6. **Response Measure** → How success is judged (e.g., response time < 2 sec, 99% uptime).

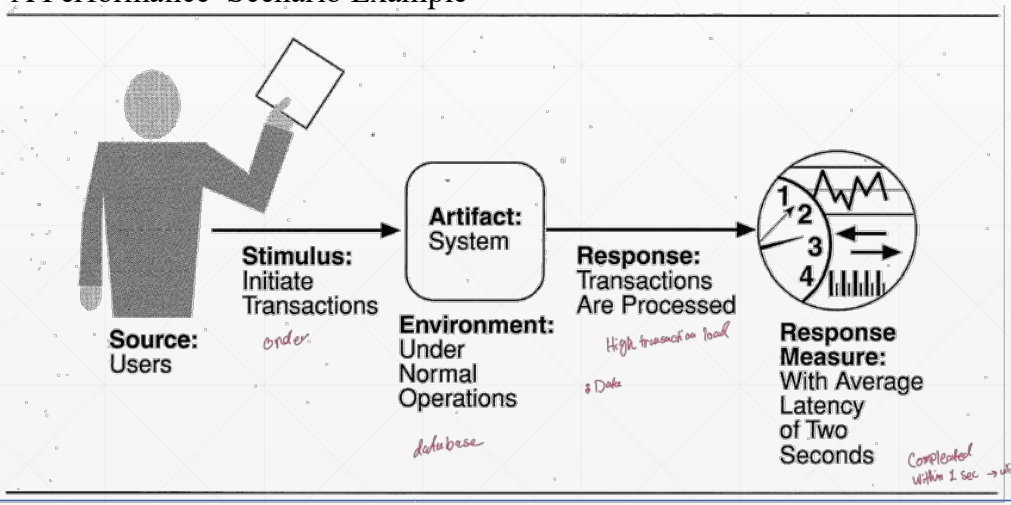
A Quality Attribute Scenario Template

- A template for writing a quality attribute scenario which is commonly used is:

*Some (**source**) generates some events (**stimulis**) that arrives at some **artefact** under some conditions (**environment**) and must be dealt with (**response**) in a satisfactory way (**response measure**)*

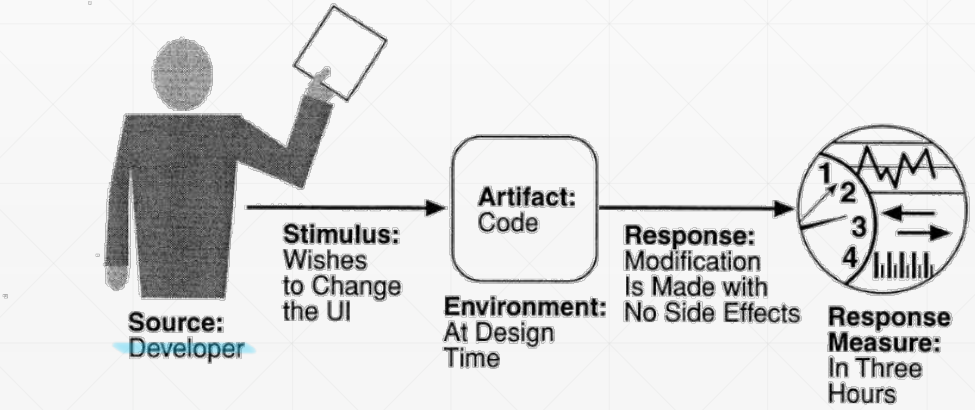
Quality Attribute Considerations – Scenarios

A Performance Scenario Example



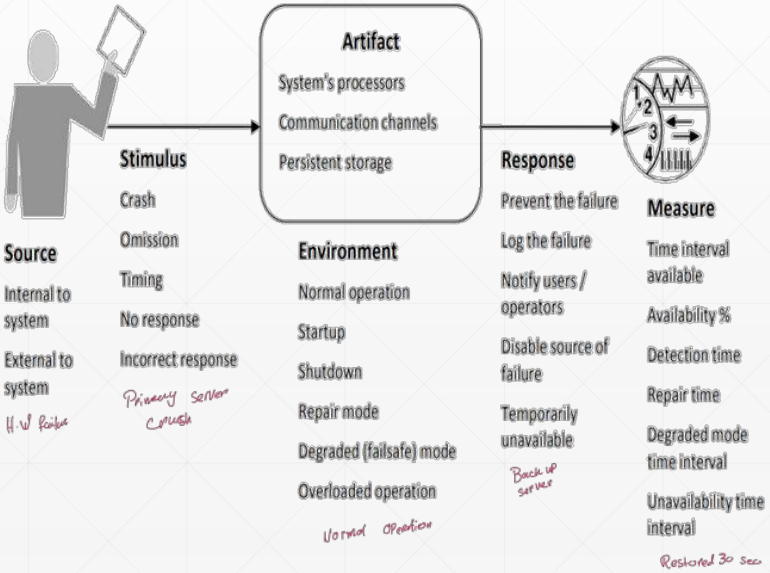
Quality Attribute Considerations – Scenarios

A maintainability Scenario Example



Quality Attribute Considerations – Scenario

Example: General Availability Scenario



Extra:

- **Source**
  - Internal to System: Problems or events that occur within the system itself, like a software bug or a memory leak.
  - External to System: Issues that come from outside the system, such as a power outage or a hacking attempt.
- **Stimulus**: The event that triggers the system's response. This can include:
  - Crash: Sudden failure of the system.
  - Omission: Failure to respond when expected.
  - Timing: Response at the wrong time.
  - No Response: Complete lack of a response.
  - Incorrect Response: The system responds, but the response is wrong.

Quality Attribute Considerations – Tactics

- There are a collection of primitive design techniques that an architect can use to achieve a quality attribute **response**.
- We call these architectural design primitives ***tactics***.
- Tactics, like design patterns, are techniques that architects have been used for years by experienced architects that we can simply capture and apply in our designs.
- We will show examples of tactics later in the following lecture.



Tactics are intended to control responses to stimuli.

Extra Examples of Tactics

➢ 1. Redundancy

- **Purpose**: To prevent system failure in case of a component failure.
- **Tactic**: Implement multiple instances of critical components so that if one fails, others can take over.
- **Stimulus**: Hardware failure, network outage.
- **Response**: Automatic failover to backup systems.

➢ 2. Caching

- **Purpose**: To improve response time and reduce load on the system.
- **Tactic**: Store frequently accessed data in a temporary storage area to make it quickly accessible.
- **Stimulus**: High number of read requests.
- **Response**: Serve data from cache rather than re-fetching it from the primary storage.

Quality Design Decisions

- Architecture design is a systematic approach to making design decisions.

These decisions can be categorized into:

1. Allocation of system responsibilities (responsibilities of system functions)
2. Coordination model (coordination between system elements)
3. Data model (identify data abstractions, their operations and properties)
4. Management of resources
5. Binding time decisions (Determine how and when architectural elements are bounded)
6. Choice of technology
7. Mapping among architectural elements (mapping of modules and runtime elements)

Extra: Quality Design Decisions Explanation

1. Allocation of System Responsibilities:

- ✓ This is about deciding which parts of the system will handle specific tasks.
- ✓ For example, in a shopping app, one part handles user logins, another manages the shopping cart, and other processes payments.

2. Coordination Model:

- ✓ This involves figuring out how different parts of the system will communicate and work together.
- ✓ For example, how the payment system informs the inventory system when an item has been purchased.

3. Data Model:

- ✓ This decision revolves around the types of data the system will use, how it's organized, and how it's accessed.
- ✓ For example, deciding to use a customer object that includes name, address, and order history.

4. Management of Resources:

- ✓ This is about deciding how the system's resources like memory, processor time, and network bandwidth are used and allocated.
- ✓ For example, ensuring there are enough server resources during a high traffic period on a website.

5. Mapping among Architectural Elements

- ✓ This involves planning out how different parts of the system (like code modules) correspond to runtime elements (like processes or tasks).
- ✓ For example, mapping a user interface module to specific services that run when the application is used.

6. Binding Time Decisions:

- ✓ These decisions determine when parts of the system are set or chosen, such as when certain modules are loaded or when specific configurations are applied.
- ✓ For instance, choosing whether a user's language preference is set at installation or at runtime.

7. Choice of Technology:

- ✓ This is about selecting the technologies the system will use, like programming languages, databases, or hardware platforms.
- ✓ For example, deciding to use Python for server-side logic, PostgreSQL for the database, and AWS for cloud hosting.

Summary

- Architecture and Requirements
- Quality Attribute Considerations
- Quality Design Decisions
- Next … Major Quality Attributes