**Quality Attributes:**

It is a measurable or a testable property of a system that indicates how well the system satisfies the needs of the stakeholders.

**Types of Requirements:**

Functional Requirements: Defines what the system does and how it reacts to runtime stimuli.

Quality Attribute Requirements: Qualifications of the functional requirements. A qualification is for example how fast a function is performed, the time taken to deploy the product or operation costs.

Constraints: A design choice that is based upon zero degrees of freedom such that it is influenced more by external factors. Example: using a particular programming language.

**Response of Architecture to these Requirements:**

Functional Requirements: Assigning responsibilities to architectural elements

Quality Attribute Requirements: Satisfied by various structures in the architecture and the behavior of elements in those structures

Constraints: Satisfied by accepting them and reconciling other design choices with them.

**Functionality:**

It is the ability of the system to do the work for which it was built. It does not define architecture, you could achieve the same functionality by creating different types of architectures. If functionality was the only thing that mattered, you wouldn’t need to divide the system into sub pieces at all. Even though it is independent of any structure, it assigns responsibilities to elements thereby resulting in a very simple architecture.

**Quality Attribute Considerations:**

There are three problems with the previous descriptions of attributes:

1. The definitions are not testable. For example: a system may be modifiable in terms of one set of changes and might not be modifiable with respect to another.
2. There are disagreements as to what quality a particular concern belongs to. For example, a denial-of-service attack can be considered a part of availability, performance, security etc. While all of these are correct, it doesn’t help architects come up with solutions for the problems
3. The attribute communities have their own vocabulary. For example: the security community has attacks on the system, the performance community has events arriving at the system, all of these might refer to the same thing but are spoken about differently by their respective communities.

The solution to problem 1 and 2 is to use quality attribute scenarios. The solution to the third problem is to provide a discussion of these attributes with respect to the vocabulary used by their communities.

**Specifying Quality Attribute Requirements:**

A quality attribute consists of these parts:

* Stimulus: An event arriving at the system. For example: the security community has attacks on the system, the performance community has events arriving at the system.
* Stimulus Source: The source of the stimulus decides how it will be treated. For example: the stimulus from a trusted source will be treated differently than a stimulus from an untrusted source.
* Response: It refers to the responsibilities that the system or user must carry out in response of the stimuli.
* Response Measure: Determining whether the response to the stimuli was satisfactory, determining if the requirement was achieved. For example: latency could be a measure for performance.

These 4 were the most important parts but there are two more

* Environment: Can be understood as the set of circumstances in which a scenario takes place. For example: a request for modification that arrives after the code has been frozen will be treated differently than if it arrived before the code had been frozen.
* Artifact: It is the part of the system to which that specific requirement applies. Usually, it is the entire system but occasionally it could be a smaller subsystem.

In order: Source of stimulus, Stimulus, Environment, Artifact, Response, Response Measure.

General Quality attribute scenarios are the one which are not specific to a system and concrete quality attribute scenarios are the ones which are specific to a system.

**Achieving Quality Attributes through Tactics:**

The required quality attribute can be achieved using architectural tactics. Tactics are design decisions that allows you to achieve a quality attribute response in answer of an stimuli. A tactic focuses on one quality attribute response and within tactic tradeoffs are not considered which makes tactics different from architectural patterns.

Why are tactics used:

1. Design patterns are difficult to implement, by using different tactics an architect can augment existing patterns to achieve a quality attribute goal.
2. If no pattern exists, tactics allow architects to construct a design fragment from ‘first principles’
3. Categorizing tactics makes the design more systematic. The tactics will overlap and you might have to choose from multiple tactics. The decision to choose which tactic depends on trade offs and other quality attributes and the cost to implement

These tactics must also be refined. For example performance: schedule resources is a common tactic but this tactic needs to be a bit more specific such as if it will use round robin etc.

**Guiding quality design decisions:**

The seven categories are:

1. Allocation of responsibilities: It involves identifying the important responsibilities and determining how these are implemented on runtime and non runtime elements.
2. Coordination Model: Elements interact with each other through mechanisms which are referred to as coordination model. Important decisions include identifying the elements that are going to interact, the properties of their interaction such as timeliness correctness etc, and the communication mechanism that will be used between external entities and the system and between elements.
3. Data Model: The collection of representation of data and how to interpret it. Involves choosing data abstractions, operations and properties; compiling metadata and organizing the data.
4. Management of Resources: Involves identifying resources that must be shared, determining how these resources must be shared and what to do in case of contention, determining the impact of saturation on them (for example when a CPU is heavily loaded, performance degrades) and determining which element manages each resource.
5. Mapping among architectural elements: Architecture must provide mappings between different elements in different types of architectural structures, it must also provide mapping between software and environment elements.
6. Binding Time Decisions: Allowable ranges of variation at different times by different entities. For example from design time by a developer to runtime by an end user. For example: For resource management you can allow your system to recognize a new device at runtime, for choice of technology, an app store can recognize the appropriate version of the application with respect to your smartphone. While making these decisions it is important to recognize their cost and the cost of any modifications after their implementation.
7. Choice of Technology: Every architecture is realized by some technology. In some cases the stakeholders decide the technology in which case it can become a strain, in other cases it is up to the architect. Includes deciding which technologies are available, determining whether this technology has adequate tools to support its development, determining the extent of internal familiarity as well as external support for the technology, determining the side effects of using it.