What is Software Architecture? how does it relate to Software Design?

- It's the set of structures needed to reason about the system,
- which comprises software elements, relations among them, and properties of both

---

Software Architecture vs. Software Design

- The details that you include in your architecture model to reason about the failures are architectural details.
- The full design of the system includes other details necessary to build the system, but not needed to reason about the failures.
    - Such details are design details.
- Architecture is design, but not all design is architecture.
- Architecture consists of architectural design decisions, and all others are non-architectural.

Types of Requirements:

- Functional requirements:
    - Describe the interactions between the system and its environment independent from the implementation
    - Example: An operator must be able to define a new game.
- Non-Functional requirements:
    - Aspects not directly related to functional behavior.
    - Capture many facets of how the functional requirements are achieved.
    - Example: The response time must be less than 1 second
- Constraints (Pseudo requirements):
    - Imposed by the client or the environment.
    - Example: The implementation language must be Java.

FURPS+

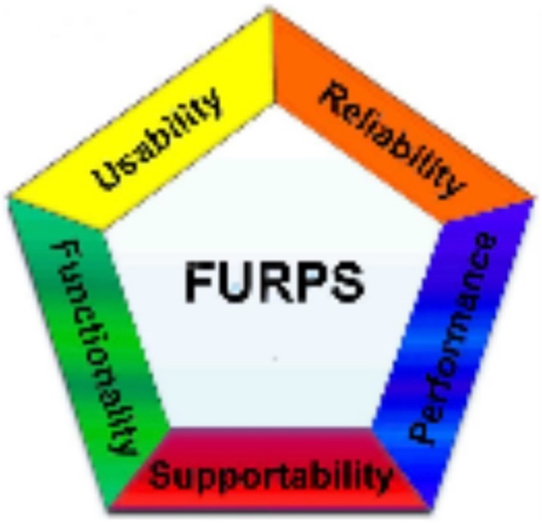
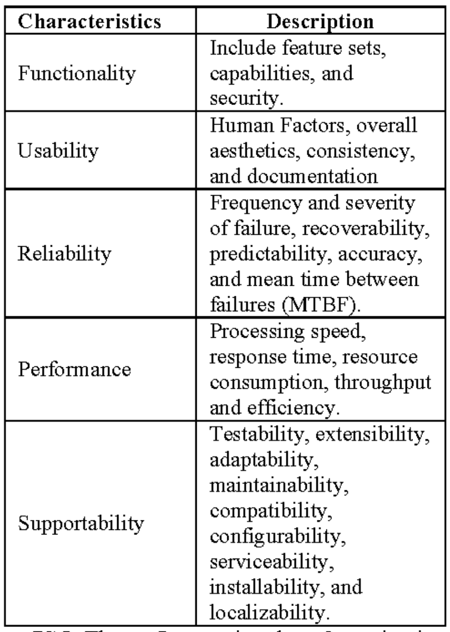


TABLE II. THE CONTENT OF FURPS MODEL



Characteristics	Description
Functionality	Include feature sets, capabilities, and security.
Usability	Human Factors, overall aesthetics, consistency, and documentation
Reliability	Frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF).
Performance	Processing speed, response time, resource consumption, throughput and efficiency.
Supportability	Testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability.

E. ISO: The International Organization for Standardization is an international-standard-setting body

Functional vs. Non-functional Requirements

1. *Functional Requirements*
   - Describe user tasks that the system needs to support
   - Phrased as actions
     - "Advertise a new league"
     - "Schedule tournament"
     - "Notify an interest group"
2. *Non-functional Requirements*
   - Describe properties of the system or the domain
   - Phrased as constraints or negative assertions
     - All user inputs should be acknowledged within 1 second
     - A system crash should not result in data loss

Types of Nonfunctional Requirements

1. *Performance:*

   A performance quality attribute defines a metric that states the amount of work an application must perform in a given time, and/or a deadline that must be met.

- Performance requirements are particularly fatal in real-time applications (like what?)

- **Performance Includes:**
  - Response time
  - Scalability
  - Throughput
  - Availability

  1. Throughput:
     - A measure of the amount of work the system must perform in unit time.
     - Work could be measured in transactions per second, or messages processed per second.
     - Example: An online banking application needs to guarantee that it can execute 1000 tps from Internet banking customers
  2. Availability:
     - The degree to which a system or component is operational and accessible when required for use
  3. Response time:
     - a measure of the latency an application exhibits in processing a business transaction.
     - Response time is most often (but not exclusively) associated with the time an application takes to respond to some input.
     - Guaranteed versus average response time?
       95% of all requests must be processed in less than 4 s, and no requests must take more than 15s.
  4. Scalability:
     - Scalability is a nonfunctional property of a system that describes the ability to appropriately handle increasing (and decreasing) workloads.
     - How well a solution to some problem will work when the size of the problem increases.

2. *Usability*

   is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

3. *Reliability*

- is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
- Failures impact on an application's reliability.

- $Percentage\ of\ availability = \frac{total\ elapsed\ time - sum\ of\ downtime}{total\ elapsed\ time}$

- $Mean\ time\ between\ failures\ MTBF = \frac{total\ elapsed\ time - sum\ of\ downtime}{number\ of\ failures}$

4. *Security*

At the architectural level, security boils down to understanding the precise security requirements for an application, and devising mechanisms to support them.

- **Security Includes:**
  - Authentication
  - Authorization
  - Encryption
  - Integrity
  - Non-repudiation

1. Authentication:
   Applications can verify the identity of their users and other applications with which they communicate

2. Authorization:
   Authenticated users and applications have defined access rights to the resources of the system.

3. Encryption:
   The messages sent to/from the application are encrypted.

4. Integrity:
   This ensures the contents of a message are not altered in transit.

5. Nonrepudiation:
   The sender of a message has proof of delivery and the receiver is assured of the sender's identity.

Khalid Shawki
k.shawki@stud.fci-cu.edu.eg