Architectural patterns play a significant role in fulfilling qualitative requirements for a software system. These qualitative attributes include items like availability, extensibility, maintainability, reliability, scalability and testability. There are many more examples of software quality attributes, but these will be the six that I focus on for this particular discussion. Each quality requirement should be able to be achieved by some architectural style, pattern, or tactic. The available architectural options for the qualitative requirement might be different between distributed and embedded systems as well.

Availability

Refers to the percentage of time that the software system is running and able to perform as expected. Availability can be impacted by many processes, like deployment methods, scheduled maintenance, faults, etc. Availability often goes hand in hand with Reliability, which is a measure of a system's fault tolerance.

Architectural Design Pattern (Distributed): Load Balancing Pattern. Consists of distributing network requests across multiple servers/nodes to ensure availability and performance. (GeeksForGeeks, 2024a)

Architectural Design Style (Embedded): Redundancy. Having multiple nodes available at a time to respond to requests, and if a failure were to happen the other nodes can take over. This principle is also prevalent in distributed systems. (GeeksForGeeks, 2024b)

Extensibility

Refers to the ability of the system or application to be modified or extended without needing to make major changes to existing code.

Architectural Design Pattern (Distributed or Embedded): Factory Pattern. Allows for objects to be created without specifying the exact class, delegating the responsibility of object creation to subclasses. This is a pattern that is typically found at the software application code level.

Maintainability

How easy it is to understand, repair, and improve a software system. Many times, this comes down to having architecture patterns in place that are well documented.

Architectural Design Pattern (Distributed or Embedded): Model View Controller (MVC). This pattern promotes modularity and a separation of concerns, making the system easier to maintain. This is a Layered architectural design style, many of these layered styles lend themselves to better maintainability.

Reliability

Ability of a system to respond effectively to faults and failures without inhibiting the user/process.

Architectural Design Pattern (Distributed): Exponential Backoff Pattern. When a request fails for unavailability or other issues, the client waits for a set amount of time until the request is retried. The wait time is extended for each retry until the maximum number of retries is reached. Similar to the circuit breaker pattern. This design increases reliability in the case that there is a transient error that should not have inhibited the originating request.

Scalability

The ability of a software solution to increase or decrease resources or processing power in response to metrics or load.

Architectural Design Pattern (Distributed or Embedded): Event Driven Architecture. Events are published triggering actions that are stateless and can be performed asynchronously and in parallel. In embedded systems this will be limited to the hardware resources of the machine hosting the application.

Testability

The ease with which a system can utilize automated testing and unit testing.

Architectural Design Pattern (Distributed or Embedded): Clean Architecture. Consists of drawing lines between the domain, infrastructure, and presentation logic of an application. This is another layered architecture pattern, and having code segmented in each of those layers promotes clear boundaries for testing. This architecture pattern also supports test driven development very well.

GeeksForGeeks. (2024a, March 21). *Design Patterns for High Availability*. GeeksforGeeks. https://www.geeksforgeeks.org/design-patterns-for-high-availability/

GeeksForGeeks. (2024b, April 2). *Factory method for designing pattern*. GeeksforGeeks. https://www.geeksforgeeks.org/factory-method-for-designing-pattern/