**Quality criteria and architecture**

The quality criteria is a criteria that can be used to judge operations of the system instead of specific behaviors. In system development quality is classified as non-functional requirement of service requirements. Non-functional requirements are often called quality attributes. As it is stated on [MSDN Microsoft] webpage there are 13 common quality attributes which are categorized in 4 specific areas:

**Design qualities:** conceptual integrity, maintainability, reusability.

**Run-time qualities:** availability, interoperability, manageability, performance, reliability, scalability, security.

**System qualities:** supportability, testability.

**User qualities:** usability.

Each of these attributes provides the key issues and the decisions you must make when building your software:

* **Conceptual integrity** is responsible for defining the consistency and coherence of the overall design.
* **Maintainability** is responsible for the system’s ability to undergo changes with a degree of ease.
* **Reusability** is responsible for the system’s ability to continue operating in the expected way over time.
* **Availability** is responsible for defining the proportion of time that the system is functional and working.
* **Interoperability** is responsible for the system’s or different systems ability to operate successfully by communicating and exchanging information with other external systems written and run by external parties.
* **Manageability** is responsible for defining how easy it is for system administrators to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning.
* **Performance** is responsible for indication of the responsiveness of a system to execute specific actions in a given time interval.
* **Reliability** is responsible for the system’s ability to continue operating in the expected way over time.
* **Scalability** is responsible for the system’s ability to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
* **Security** is responsible for the system’s capability to reduce the chance of malicious or accidental actions outside of the designed usage affecting the system, and prevent disclosure or loss of information.
* **Supportability** is responsible for the system’s ability to provide information helpful for identifying and resolving issues when it fails to work correctly.
* **Testability** is responsible formeasuring of how well system or components allow you to create test criteria and execute tests to determine if the criteria are met.
* **Usability** is responsible for designing the application interfaces with the user and consumer in mind so that they are intuitive to use, can be localized and globalized, provide access for disabled users, and provide a good overall user experience.

Some of these attributes are measured in metrics but some of them are hard to measure. For example performance may be measured with time and space but usability can’t be measured in metrics so developers have to be more creative.

The plan for implementing non-functional requirements is detailed in the [system architecture](https://en.wikipedia.org/wiki/Systems_architecture). To design the architecture Ian Sommerville suggests that it is required to make a decisions based on these attributes [Architectural design decisions]. It is important to decide on how the architecture will look, how many platforms should it work on, if it’s a web based application, should there be more clients (like web client and desktop client), is the software will be split and placed around the separate hardware and locations, what database should it use and so on. For this reason there are many architecture patterns that allow you to decide how you going to build your software and point out all the advantages and disadvantages of using them, and explains in which situations those may be used. Ian Sommerville described the 5 most common architectures: MVC(Model-View-Controller), Layered Architecture, Repository, Client-server and Pipe and filter architectures [Architectural patterns]. Every each of them is unique and good for specific software and might be similar to each other. System architecture should be well documented and viewed from many angles like - logical view, physical view, process view and development view, but developers still argue about it [Architecture views]. Also users of agile methods claim that detailed design documentation is mostly unused and is a waste of money and time. To ensure quality while planning the architecture it’s suggested to use these quality attributes and to analyze the trade-offs between them. The importance and priority of the attributes differs from system to system.

[MSDN Microsoft] <https://msdn.microsoft.com/en-us/library/ee658094.aspx>, 01.12.2015

[Architectural design decisions] Software Engineering, 10th edition, Ian Sommerville, Page 171.

[Architectural patterns] Software Engineering, 10th edition, Ian Sommerville, Page 175.

[Architecture views] Software Engineering, 10th edition, Ian Sommerville, Page 173.