Non-functional requirement

In systems engineering and requirements engineering, a **non-functional requirement** (**NFR**) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behavior or functions. The plan for implementing *functional* requirements is detailed in the system *design*. The plan for implementing *non-functional* requirements is detailed in the system *architecture*, because they are usually architecturally significant requirements.[1]

Definition

Broadly, functional requirements define what a system is supposed to *do* and non-functional requirements define how a system is supposed to *be*. Functional requirements are usually in the form of "system shall do <requirement>", an individual action or part of the system, perhaps explicitly in the sense of a mathematical function, a black box description input, output, process and control functional model or IPO Model. In contrast, non-functional requirements are in the form of "system shall be <requirement>", an overall property of the system as a whole or of a particular aspect and not a specific function. The system's overall properties commonly mark the difference between whether the development project has succeeded or failed.

Non-functional requirements are often called the "quality attributes" of a system. Other terms for non-functional requirements are "qualities", "quality goals", "quality of service requirements", "constraints", "non-behavioral requirements",[2] or "technical requirements".[3] Informally these are sometimes called the "ilities", from attributes like stability and portability. Qualities—that is non-functional requirements—can be divided into two main categories:

1. Execution qualities, such as safety, security and usability, which are observable during operation (at run time).
2. Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the system.[4][5]

It is important to specify non-functional requirements in a specific and measurable way.[6][7]

Examples

A system may be required to present the user with a display of the number of records in a database. This is a functional requirement. How current this number needs to be, is a non-functional requirement. If the number needs to be updated in real time, the system architects must ensure that the system is capable of displaying the record count within an acceptably short interval of the number of records changing.

Sufficient network bandwidth may be a non-functional requirement of a system. Other examples include:

- Accessibility
- Adaptability
- Auditability and control
- Availability (see service level agreement)
- Backup
- Boot up time
- , current and forecast
- Certification

- Compliance
- Configuration management
- Conformance
- Cost, initial and Life-cycle cost
- Data integrity
- Data retention
- Dependency on other parties
- Deployment
- Development environment
- Disaster recovery
- Documentation
- Durability
- Efficiency (resource consumption for given load)
- Effectiveness (resulting performance in relation to effort)
- Elasticity
- Emotional factors (like fun or absorbing or has "Wow! Factor")
- Environmental protection
- Escrow
- Exploitability
- Extensibility (adding features, and carry-forward of customizations at next major version upgrade)
- Failure management
- Fault tolerance (e.g. Operational System Monitoring, Measuring, and Management)
- Flexibility (e.g. to deal with future changes in requirements)
- Footprint reduction - reduce the exe files size
- Integrability (e.g. ability to integrate components)
- Internationalization and localization
- Interoperability
- Legal and licensing issues or patent-infringement-avoidability
- Maintainability (e.g. mean time to repair – MTTR)
- Management
- Memory Optimization
- Modifiability
- Network topology
- Open source
- Operability
- Performance / response time (performance engineering)
- Platform compatibility
- Privacy (compliance to privacy laws)
- Portability
- Quality (e.g. faults discovered, faults delivered, fault removal efficacy)
- Readability
- Reliability (e.g. mean time between/to failures – MTBF/MTTF)
- Reporting
- Resilience
- Resource constraints (processor speed, memory, disk space, network bandwidth, etc.)
- Response time
- lity
- Robustness

- Safety or factor of safety
- Scalability (horizontal, vertical)
- Security (cyber and physical)
- Software, tools, standards etc. Compatibility
- Stability
- Supportability
- Testability
- Throughput
- Transparency
- Usability (human factors) by target user community
- Volume testing

See also

- ISO/IEC 25010:2011
- Consortium for IT Software Quality
- ISO/IEC 9126
- FURPS
- Requirements analysis
- Usability requirements
- Non-Functional Requirements framework
- Architecturally Significant Requirements
- SNAP Points

References

1. Chen, Lianping; Ali Babar, Muhammad; Nuseibeh, Bashar (2013). "Characterizing Architecturally Significant Requirements". *IEEE Software*. **30** (2): 38–45. doi:10.1109/MS.2012.174 (https://doi.org/10.1109%2FMS.2012.174). hdl:10344/3061 (https://hdl.handle.net/10344%2F3061). S2CID 17399565 (https://api.semanticscholar.org/CorpusID:17399565).
2. Stellman, Andrew; Greene, Jennifer (2005). *Applied Software Project Management* (https://web.archive.org/web/20150209011617/http://www.stellman-greene.com/aspm/). O'Reilly Media. p. 113. ISBN 978-0-596-00948-9. Archived from the original (http://www.stellman-greene.com/aspm/) on 2015-02-09.
3. Ambler, Scott. "Technical (Non-Functional) Requirements: An Agile Introduction" (http://agilemodeling.com/artifacts/technicalRequirement.htm). *Agile Modelling*. Ambysoft Inc. Retrieved 5 October 2018.
4. Wiegers, Karl; Beatty, Joy (2013). *Software Requirements, Third Edition*. Microsoft Press. ISBN 978-0-7356-7966-5.
5. Young, Ralph R. (2001). *Effective Requirements Practices* (https://archive.org/details/unset0000unse_g5k2). Addison-Wesley. ISBN 978-0-201-70912-4.
6. Zimmermann, Olaf; Stocker, Mirko (2021). *Design Practice Repository* (https://leanpub.com/dpr). LeanPub.
7. Glinz, Martin (2008). "A Risk-Based, Value-Oriented Approach to Quality Requirements" (https://www.zora.uzh.ch/id/eprint/7375/10/ieeesoftwaregetPDFV.pdf) (PDF). *IEEE Software*. **25** (2): 34–41. doi:10.1109/MS.2008.31 (https://doi.org/10.1109%2FMS.2008.31). S2CID 19015424 (https://api.semanticscholar.org/CorpusID:19015424).

External links

- Bettel, J. H. Eide (2005). "Quantification and Traceability of Requirements". CiteSeerX 10.1.1.95.6464 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.6464).

- Dalbey, John. "Nonfunctional Requirements" (http://www.csc.calpoly.edu/~jdalbey/SWE/QA/nonfunctional.html). *Csc.calpoly.edu*. Retrieved 3 October 2017.
- "Modeling Non-Functional Aspects in Service Oriented Architecture" (https://web.archive.org/web/20110724131731/http://www.cs.umb.edu/~jxs/pub/scc.pdf) (PDF). *Cs.umb.edu*. Archived from the original (http://www.cs.umb.edu/~jxs/pub/scc.pdf) (PDF) on 24 July 2011. Retrieved 3 October 2017.
- "Non-Functional Requirements: Do User Stories Really Help?" (http://www.methodsandtools.com/archive/archive.php?id=113). *Methodsandtools.com*. Retrieved 3 October 2017.
- "Non-Functional Requirements Be Here - CISQ - Consortium for IT Software Quality" (http://it-cisq.org/non-functional-requirements-be-here). *it-cisq.org*. Retrieved 3 October 2017.
- " "Do Software Architectures Meet Extra-Functional or Non-Functional Requirements?" " (https://ozimmer.ch/practices/2020/11/19/ExtraExtraReadAllboutIt.html). 19 November 2020.
