Non-Functional Requirements Capture

(Quality Attribute)

Goals

In software engineering projects, non-functional requirements, also known as quality attributes, are specifications that define the operational attributes of a system rather than its specific behaviors. Unlike functional requirements, which outline what a system should do, non-functional requirements describe how the system performs certain functions under specific conditions. Non-functional requirements generally increase the cost as they require special efforts during the implementation, but by defining these requirements in detail early in the engagement, they can be properly evaluated when the cost of their impact on subsequent design decisions is comparatively low.

Documenting Non-Functional Requirements - Best Practices

- Be specific: Avoid ambiguity and make sure the requirement is quantitative, measurable and testable.
- Relate requirements with business objectives and understand the real impact of the system's behavior.
- Break it down: Try to define requirements at the component or process scope instead of the whole solution.
- Understand trade-off: Non-functional requirements may be in conflict

with each other and it can be difficult to balance them and prioritize which one to implement.

Template

This template can serve as a structured framework for capturing and documenting non-functional requirements effectively. Adjustments can be made to tailor it to the specific needs and preferences of the project team.

- Requirement name: name or title
- Description: brief description. Describe the importance and impact of this requirement to the business.
- Priority: High/Medium/Low or Must-have/Nice-to-have, etc
- Measurement/Metric: metric or measurement criteria
- Verification Method: Automated test, benchmark, simulation, prototyping, etc.
- Constraints: Budget, Time, Resources, Infrastructure, etc.
- Owner/Responsible Party
- Dependencies: technical dependencies, data dependencies, regulatory dependencies, etc.

Examples

To support the process of capturing a project's *comprehensive* non-functional requirements, this document offers a taxonomy for non-functional requirements and provides a framework for their identification, exploration, assignment of customer stakeholders, and eventual codification into formal engineering requirements as input to subsequent solution design.

Operational Requirements

Quality Attribute	Description	Common Metrics
Availability	System's uptime and accessibility to users.	- Uptime: Uptime measures the percentage of time that a system is operational and available for use. It is typically expressed as a percentage of total time (e.g., 99.9% uptime means the system is available 99.9% of the time). Common thresholds for uptime include: 99% uptime: The system is available 99% of the time, allowing for approximately 3.65 days of downtime per year. 99.9% uptime (three nines): The system is available 99.9% of the time, allowing for approximately 8.76 hours of downtime per year. 99.99% uptime (four nines): The system is available 99.99% of the time, allowing for approximately 52.56 minutes of downtime per year. 99.999% uptime (five nines): The system is available 99.999% of the time, allowing for approximately 5.26 minutes of downtime per year.

Data Integrity	Accuracy and consistency of data throughout its lifecycle.	- Error Rate: The proportion of data entries that contain errors or inaccuracies. (\text{Error Rate} = \left(\frac{\text{Number of Errors}}{\text{Total Number of Entries}} \right) \times 100) - Accuracy Rate: The percentage of data entries that are correct and match the source of truth. (\text{Accuracy Rate} = \left(\frac{\text{Number of Accurate Entries}}{\text{Total Number of Entries}} \right) \times 100) - Duplicate Record Rate: The percentage of data entries that are duplicates. (\text{Duplicate Record Rate} = \left(\frac{\text{Number of Duplicate Entries}}{\text{Total Number of Entries}} \right) \times 100)
Disaster recovery and business continuity	Determine the system's requirements for disaster recovery and business continuity, including backup and recovery procedures and disaster recovery	- Backup and Recovery: The application must have a Backup and Recovery plan in place that includes regular backups of all data and configurations, and a process for restoring data and functionality in the event of a disaster or disruption. - Redundancy: The application must have Redundancy built into

	testing.	its infrastructure, such as redundant servers, network devices, and power supplies, to ensure high availability and minimize downtime in the event of a failure. - Failover and high availability: The application must be designed to support Failover and high availability, such as by using load balancers or Failover clusters, to ensure that it can continue to operate in the event of a system failure or disruption. - Disaster Recovery plan: The application must have a comprehensive disaster Recovery plan that includes procedures for restoring data and functionality in the event of a major disaster, such as a natural disaster, cyber attack, or other catastrophic event. - Testing and Maintenance: The application must be regularly tested and maintained to ensure that it can withstand a disaster or disruption, and that all systems, processes, and data can be quickly restored and recovered.
Reliability	System's ability to maintain	- Mean Time Between Failures (MTBF): The system should

	functionality under varying conditions and failure scenarios.	achieve an MTBF of at least 1000 hours, indicating a high level of reliability with infrequent failures. - Mean Time to Recover (MTTR): The system should aim for an MTTR of less than 1 hour, ensuring quick recovery and minimal disruption in the event of a failure. - Redundancy Levels: The system should include redundancy mechanisms to achieve a redundancy level of N+1, ensuring high availability and fault tolerance.

Performance Requirements

Quality Attribute	**Description**	**Common Metrics**
Capacity	Maximum load or volume that the system can handle within specified performance criteria.	- Maximum Load Capacity: The system should be capable of handling peak loads without exceeding predefined performance degradation thresholds. Maximum load capacity may be expressed in terms of concurrent users, transactions per second, or data volume. - Resource Utilization: Measures the

		percentage of system resources (CPU, memory, disk I/O, network bandwidth) consumed under normal operation. - Concurrency: Measures the number of simultaneous users or transactions the system can handle without degradation in performance. - Throughput: Measures the rate at which the system processes transactions, requests, or data. Thresholds may be defined in terms of transactions per second, requests per minute, or data throughput in bytes per second.
Performance	Define the expected response times, throughput, and resource usage of the solution.	- Response time: The application must load and respond to user interactions within 500 ms for button clicks. - Throughput: The application must be able to handle 100 concurrent users or 500 transactions per second. - Resource utilization: The application must use less than 80% of CPU and 1 GB of memory. - Error rates: The application must have an error rate less than 1% of all requests, and be able to handle and recover from errors gracefully, without impacting user experience or data integrity.
Scalability	Determine	- Load Balancing: The application

	how the system will handle increased user loads or larger datasets over time.	must be able to handle a minimum of 250 concurrent users and support load balancing across at least 3 servers to handle peak traffic. - Database Scalability: The application's database must be able to handle at least 1 million records and support partitioning or sharding to ensure efficient storage and retrieval of data. - Cloud-Based Infrastructure: The application must be deployed on cloud-based infrastructure that can handle at least 100,000 requests per hour, and be able to scale up or down to meet changing demand. Microservices Architecture: The application must be designed using a microservices architecture that allows for easy scaling of individual services, and be able to handle at least 500 requests per second. - Caching: The application must be able to cache at least 10,000 records, with a cache hit rate of 95%, and support caching across multiple servers to ensure high availability.

Security and Compliance Requirements

Quality	Description	Common Metrics

Attribute		
Compliance	Adherence to legal, regulatory, and industry standards and requirements.	See Microsoft Purview Compliance Manager
Privacy	Protection of sensitive information and compliance with privacy regulations.	- Compliance with Privacy Regulations: Achieve full compliance with GDPR, CCPA and HIPAA. - Data Anonymization: Implement anonymization techniques in protecting individual privacy while still allowing for data analysis. - Data Encryption: Ensure that sensitive data is encrypted according to encryption standards and best practices. - User Privacy Preferences: The ability to respect and accommodate user privacy preferences regarding data collection, processing,

		and sharing.
Security	Establish the security requirements of the system, such as authentication, authorization, encryption, and compliance with industry or legal regulations.	See Threat Modeling Tool
Sustainability	Ability to operate over an extended period while minimizing environmental impact and resource consumption.	- Energy Efficiency: Kilowatt-hours/Transaction. - Carbon Footprint: Tons of CO2 emissions per year.

System Maintainability Requirements

Quality Attribute	Description	Common Metrics
Interoperability	Ability to interact and exchange data with other systems or	- Data Format Compatibility: The system must be interoperable with various Electronic Health Records (HER) systems to exchange patient data securely. - Protocol Compatibility: The

	components.	system should import and export banking information from the ERP using REST protocol. - API Compatibility: The solution must adhere to API standards, ensuring backward compatibility with previous API versions, and providing comprehensive documentation for developers.
Maintainability	Ease of modifying, updating, and extending the software over time.	- Code Complexity: The level of complexity in the system's codebase, measured using metrics such as cyclomatic complexity or lines of code per function. Lower code complexity makes maintenance tasks easier and reduces the likelihood of introducing defects. A cyclomatic complexity score of less than 10 or a lines of code per function metric below 50 is often desirable. - Code Coverage: The percentage of code covered by automated tests. Higher code coverage indicates better testability and facilitates easier maintenance by enabling faster detection of defects. A code coverage threshold of 80% or higher is commonly targeted. - Documentation Quality: The

		comprehensiveness and clarity of documentation accompanying the system, including design documents, technical specifications, and user manuals. Well-written documentation reduces the time and effort required for maintenance tasks. Documentation should cover at least 80% of system functionality with clear explanations and examples. - Dependency Management: The management of external dependencies and libraries used in the system. Proper dependency management reduces the risk of compatibility issues and simplifies maintenance tasks such as updates and patches. - Code Churn: The frequency of code changes within a software system. High code churn may indicate instability or frequent updates, making maintenance more challenging. A code churn rate of less than 20% is generally considered acceptable.
Observability	The ability to measure a system's internal state	-System Metrics: CPU usage, memory usage, disk I/O, network I/O, and other resource utilization metrics.

	and performance based on the outputs it generates, such as logs, metrics, and traces.	- Application Metrics: Response times, request rates, error rates, and throughput. - Custom Metrics: Application-specific metrics, such as user sign-ups, or specific business logic indicators.
Portability	Ability to run the software on different platforms, environments, and devices.	- Platform Compatibility: The ability of the software to run on different operating systems (e.g., Windows, macOS, Linux) or platforms (e.g., desktop, mobile, web). Portability requires the software to be compatible with multiple platforms, with a goal of supporting at least three major platforms. - Hardware Compatibility: The ability of the software to run on different hardware configurations, such as varying processor architectures (e.g., x86, ARM) or memory sizes. Portability involves ensuring compatibility with a wide range of hardware configurations, with a goal of supporting common hardware architectures. - File System Independence: The software's ability to operate independently of the underlying file system, ensuring compatibility with different file systems (e.g., NTFS,

		ext4, APFS). Portability involves using file system abstraction layers or APIs to abstract file system operations and ensure consistency across platforms. - Data Format Compatibility: The software's ability to read and write data in different formats, ensuring compatibility with common data interchange formats (e.g., JSON, XML, CSV). Portability involves supporting standard data formats and providing mechanisms for data conversion and interoperability.	

User Experience Requirements

Quality	Attribute	Description	Common Metrics
Accessibility	The solution must be usable by people with disabilities. Compliance with accessibility standards. Support for assistive	- Alternative Text for Images: All images and non-text content must have alternative text descriptions that can be read by screen readers. - Color contrast: The application must use color	

	technologies	schemes that meet the recommended contrast ratio between foreground and background colors to ensure visibility for users with low vision. - Focus indicators: The application must provide visible focus indicators to highlight the currently focused element, which is especially important for users who rely on keyboard navigation. - Captions and Transcripts: All audio and video content must have captions and transcripts, to ensure that users with hearing impairments can access the

		content. - Language identification: The application must correctly identify the language of the content, to ensure that screen readers and other assistive technologies can read the content properly.
Internationalization and Localization	Adaptation of the software for use in different languages and cultures. Tailoring the software to meet the specific needs of different regions or locales.	- Language and Locale Support: The software's support for different languages, character sets, and locales. Portability requires internationalization and localization efforts to ensure that the software can be used effectively in different regions and cultures, with support for at least

		five major languages. - Multi currency: The system's support for multiple currencies, allowing different symbols and conversion rates.	
Usability	Intuitiveness, ease of learning, and user satisfaction with the software interface.	- Task Completion Time: The average time it takes for users to complete specific tasks. A user must be able to complete an account settings in less than 2 minutes. - Ease of Navigation: The ease with which users can navigate through the system and find the information they need. This can be measured by observing user interactions or conducting	

	usability tests. - User Satisfaction: User satisfaction can be measured using surveys, feedback forms, or satisfaction ratings. A satisfaction score of 70% or higher is typically considered satisfactory. - Learnability: The ease with which new users can learn to use the system. This can be measured by the time it takes for users to perform basic tasks or by conducting usability tests with novice users.

Last update: August 26, 2024