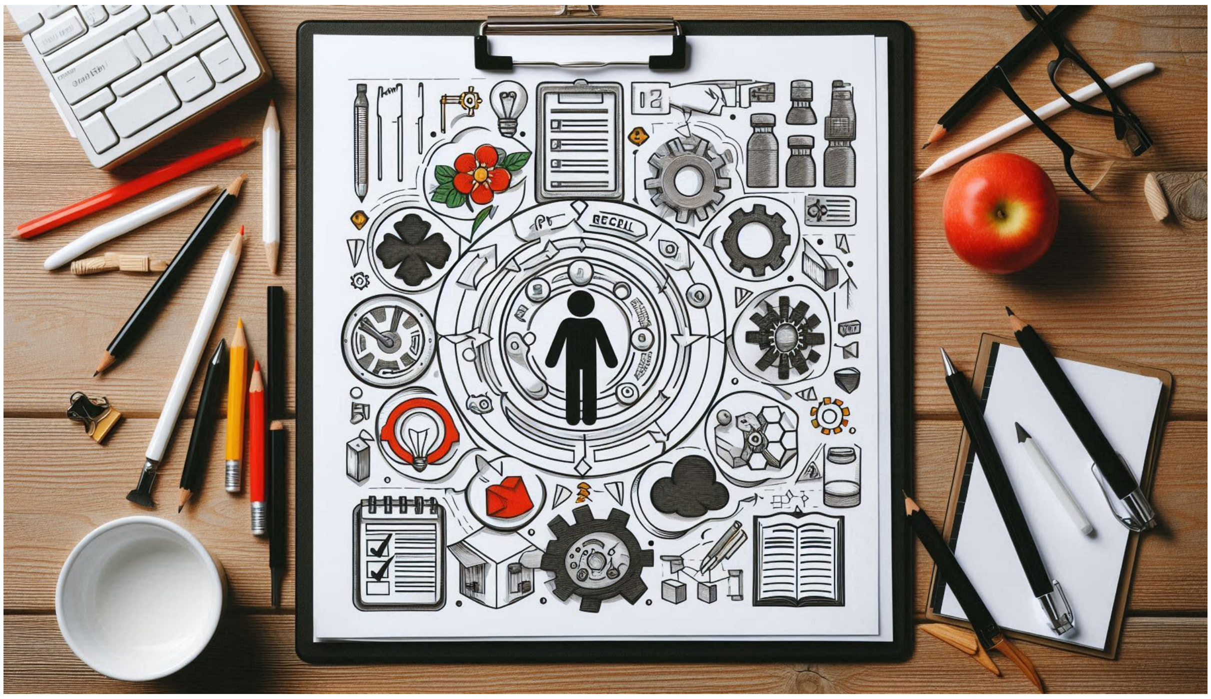
Understanding Functional and Non-Functional Requirements



When designing a software solution, one of the most critical steps is defining both functional and non-functional requirements. These requirements guide the development process, ensuring that the final product meets user needs, operates as expected, and performs reliably in real-world conditions.

Functional vs. Non-Functional Requirements

Functional requirements describe the specific behaviours and functions that the system must provide. They are directly related to what the system should do and include details such as the inputs, processes, and outputs.

Non-functional requirements are more about how the system operates. They define the quality attributes and constraints, such as security, performance, and usability, which determine the overall system performance but not the specific behaviour of the application.

Examples:

Functional Requirements	Non-Functional Requirements
Users can browse room availability, select their desired room, and complete the booking process through a mobile application.	The system must be able to handle increased traffic during peak booking seasons without performance degradation.
Include multimedia content (videos, images, maps) for an interactive experience.	Data encryption must be employed to protect sensitive guest information.
Enable guests to create profiles that store preferences, past bookings, and special requests.	Regular backups and disaster recovery procedures must be in place.
Implement a loyalty rewards program that incentivises repeat stays and interactions with the app.	The app should load quickly (within 2 seconds) and provide a responsive experience.

Secure by Design

The concept of **'secure by design'** means that security is embedded into the system from the very beginning, rather than being added as an afterthought. When software is built to be secure by design, developers prioritise security at every stage of the software development lifecycle, including during the definition of functional and non-functional requirements.

This approach influences:

- **Functional requirements** by ensuring security features, such as authentication, authorisation, encryption, and data validation, are included from the start.
- **Non-functional requirements** by shaping performance, reliability, and other factors around the need to maintain a secure environment, such as ensuring the system can handle threats and protect user data.

By adopting a secure by design mindset, decisions about how the system operates, including platform selection, data management, and system architecture, are influenced by security considerations.

Functional Requirements

Functional requirements are focused on what the system will do and are usually determined by considering the:

- Inputs required
- Data needed
- Data processing
- Logic of the system
- Deployment and usage platforms

Inputs Required

Defining functional requirements starts by identifying the **inputs** that the system must process. This includes understanding what data the users or external systems will provide, in what formats, and under what conditions. Inputs might come from user interfaces, external sensors, databases, or third-party services.

Example: Users can browse room availability, select their desired room, and complete the booking process through a web application.

Data Needed

After identifying the inputs, it’s necessary to define the data needed to fulfil the system's purpose. This includes details about data storage, management, and retrieval. What data does the system need to function, and how should it manage that data?

Example: Storing data in a MySQL database which can be processed by PHP code and viewed on an HTML/CSS/JavaScript front-end.

Data Processing

Functional requirements must also detail how the system will process the input data. What algorithms, transformations, and computations must be performed on the data? This includes mapping user interactions to backend operations and defining how business rules are applied.

Example: Each guest's profile will be central to the app, offering tailored experiences, such as personalised room settings (e.g., temperature, lighting) and curated local attractions

Logic of the System

The system's logic describes how different components and data flows work together. For example, how user actions trigger processes, how the system responds to invalid data, or what workflows must be implemented to meet user needs.

Example: Guests can use the app to book rooms, check in, and access their rooms without visiting the front desk.

Deployment and Usage Platforms

Functional requirements need to reflect the platforms where the software will be deployed, whether it's a web application, a mobile app, or a desktop solution. The software's functionality may vary depending on platform-specific requirements, such as device capabilities, operating systems, or network environments.

Example: Develop a mobile-friendly web application where customers can search for a book rooms, as well as check-in facilities and personalised recommendations.

Non-Functional Requirements

Non-functional requirements define how the system should perform in terms of security, accessibility, scalability, and performance.

Security Considerations

Non-functional requirements include specific security measures like encryption standards, password policies, and protection against common threats such as SQL injections or DDoS attacks. The system must ensure that it can handle threats without sacrificing performance or usability.

Example: Data encryption must be employed to protect sensitive guest information

Required Accessibility Features

Accessibility ensures the system can be used by people with disabilities. This includes features like screen readers, voice control, and keyboard navigation, as well as compliance with standards such as the Web Content Accessibility Guidelines (WCAG).

Example: Support for multiple languages to accommodate international guests. Ensure compliance with W3C and WCAG.

Scalability Requirements

Scalability refers to the system's ability to handle increased load or user demand without performance degradation. Non-functional requirements define how the system must scale, horizontally (adding more servers) or vertically (upgrading existing servers), to ensure smooth performance under varying loads.

Example: The system must be able to handle increased traffic during peak booking seasons such as summer without performance degradation.

Key Performance Indicators and Metrics

Non-functional requirements also cover performance metrics such as:

- Responsiveness
- Load handling
- Reliability

*See the document on **Key Performance Indicators** for more details.*

Responsiveness

This is about how quickly the system responds to user actions or external inputs. This includes page load times, system feedback delays, and transaction speeds. For example, the app should load quickly (within 2 seconds) and provide a responsive experience.

Load Handling

This is about how well the system manages concurrent users or processes under stress. This includes the maximum number of users or requests the system can handle at once before performance drops.

Reliability

Ensuring the system is available and functional over time. Reliability is often expressed in terms of uptime percentages or acceptable failure rates. For example, the app should be available 95% of the time – KPIs need to be realistic, you will never achieve an uptime of 100%.

User Acceptance Criteria

User acceptance criteria are a set of non-functional requirements that define the conditions under which the end users will accept the system. This might include usability thresholds, performance expectations, or system compatibility with specific devices or environments. For example, a web application needs to be viewable on both desktop and mobile devices without the loss of features on a smaller screen.

Using 'Spike Testing' to Establish Requirements

Spike testing is an early product-testing method that involves rapidly building a simple version of the software (a "spike") to explore unknowns, validate assumptions, and understand the potential problems and scope of the solution. Spike testing can be instrumental in establishing both functional and non-functional requirements.

For example, spike testing can:

- Help determine the scope of inputs, processes, and outputs needed for a system.
- Expose challenges related to performance, security, or scalability early in the development process, guiding more accurate non-functional requirements.
- Allow the development team to gauge how the system handles different conditions and edge cases, leading to better refinement of both functional and non-functional requirements.

Conclusion

Understanding and defining functional and non-functional requirements are essential steps in building a software solution that not only meets user needs but also performs reliably and securely. By incorporating security by design principles and using tools like spike testing, development teams can ensure that their software is robust, scalable, and secure. Both sets of requirements work together to create a complete, functional, and high-quality solution.