**Functional requirement:**

* Functional requirements define what a product must do, what its features and functions are.
* Functional requirements may involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describe all the cases where the system uses the functional requirements, these are captured in use cases.
* Functional requirements are supported by non-functional requirements (also known as "quality requirements"), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). Generally, functional requirements are expressed in the form "system must do <requirement>," while non-functional requirements take the form "system shall be <requirement>."
* The plan for implementing functional requirements is detailed in the system design, whereas non-functional requirements are detailed in the system architecture.
* As defined in requirements engineering, functional requirements specify particular results of a system. This should be contrasted with non-functional requirements, which specify overall characteristics such as cost and reliability. Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.

**Process:**

A typical functional requirement will contain a unique name and number, a brief summary, and a rationale. This information is used to help the reader understand why the requirement is needed, and to track the requirement through the development of the system.[7] The crux of the requirement is the description of the required behavior, which must be clear and readable. The described behavior may come from organizational or business rules, or it may be discovered through elicitation sessions with users, stakeholders, and other experts within the organization.

Many requirements may be uncovered during the use case development. When this happens, the requirements analyst may create a placeholder requirement with a name and summary, and research the details later, to be filled in when they are better known.

**Types of functional requirements and their specifications:**

Functional requirements can be classified according to different criteria. For example, we can group them on the basis of the functions a given feature must perform in the end product. Of course, they would differ depending on the product being developed, but for the sake of an example, the types of functional requirements might be

* Authentication
* Authorization levels
* Compliance to laws or regulations
* External interfaces
* Transactions processing
* Reporting
* Business rules, etc.

Requirements are usually written in text, especially for Agile-driven projects. However, they may also be visuals. Here are the most common formats and documents:

* Software requirements specification document
* Use cases
* User stories
* Work Breakdown Structure (WBS), or functional decomposition
* Prototypes
* Models and diagrams

Let’s see what each of them is about.

**Software requirements specification document:**

Both functional and nonfunctional requirements can be formalized in the software requirements specification (SRS) document. To learn more about software documentation in general, read our article on that topic. The SRS contains descriptions of functions and capabilities that the product must provide. The document also defines constraints and assumptions. The SRS can be a single document communicating functional requirements or it may accompany other software documentation like user stories and use cases.

We don’t recommend composing SRS for the entire solution before the development kick-off, but you should document the requirements for every single feature before actually building it. Once you receive the initial user feedback, you can update the document.

**SRS must include the following sections:**

**Purpose.** Definitions, system overview, and background.

**Overall description.** Assumptions, constraints, business rules, and product vision.

**Specific requirements.** System attributes, functional requirements, and database requirements.

It’s essential to make the SRS readable for all stakeholders. You also should use templates with visual emphasis to structure the information and aid in understanding it. If you have requirements stored in some other document formats, provide a link to them so that readers can find the needed information.

**Use cases:**

Use cases describe the interaction between the system and external users that leads to achieving particular goals.

**Each use case includes three main elements:**

* **Actors.** These are the external users that interact with the system.
* **System.** The system is described by functional requirements that define an intended behavior of the product.
* **Goals.** The purposes of the interaction between the users and the system are outlined as goals.

**There are two formats to represent use cases:**

* Use case specification structured in textual format
* Use case diagram

A **use case specification** represents the sequence of events along with other information that relates to this use case. A typical use case specification template includes the following information:

* Description
* Pre- and Post- interaction condition
* Basic interaction path
* Alternative path
* Exception path

A **use case diagram** doesn’t contain a lot of details. It shows a high-level overview of the relationships between actors, different use cases, and the system.

**The use case diagram includes the following main elements:**

* **Use cases.** Usually drawn with ovals, use cases represent different interaction scenarios that actors might have with the system (log in, make a purchase, view items, etc.).
* **System boundaries.** Boundaries are outlined by the box that groups various use cases in a system.
* **Actors.** These are the figures that depict external users (people or systems) that interact with the system.
* **Associations.** Associations are drawn with lines showing different types of relationships between actors and use cases.

Finally, all user stories must fit the **INVEST quality model:**

* I – Independent
* N – Negotiable
* V – Valuable
* E – Estimable
* S – Small
* T – Testable

**Independent.** This means that you can schedule and implement each user story separately. This is very helpful if you implement continuous integration processes.

**Negotiable.** This means that all parties agree to prioritize negotiations over specification. This also means that details will be created constantly during development.

**Valuable.** A story must be valuable to the customer. You should ask yourself from the customer’s perspective “why” you need to implement a given feature.

**Estimable.** A quality user story can be estimated. This will help a team schedule and prioritize the implementation. The bigger the story is, the harder it is to estimate it.

**Small.** Good user stories tend to be small enough to plan for short production releases. Small stories allow for more specific estimates.

**Testable.** If a story can be tested, it’s clear enough and good enough. Tested stories mean that requirements are done and ready for use.

**Functional decomposition or Work Breakdown Structures (WBS)**

A functional decomposition or WBS is a visual document that illustrates how complex processes break down into their simpler components. WBS is an effective approach to allow for an independent analysis of each part. WBS also helps capture the full picture of the project.

We suggest the following logic of functional decomposition:

1. Find the most general function.
2. Find the closest sub function.
3. Find the next level of sub function.
4. Check your diagram.

Or the decomposition process may look like this:

**High Level Function ->Sub-function -> Process -> Activity**

The features should be decomposed to the point at which the lowest level parts can’t be broken down any further.

**Example:**

**Software prototypes**

Software prototype is an umbrella term for different forms of early stage deliverables that are built to showcase how requirements must be implemented. Prototypes help bridge the vision gaps and let stakeholders and teams clarify complicated areas of products in development. Traditionally, prototypes represent how the solution will work and give examples of how users will interact with it to accomplish their tasks.

Prototypes can be cheap and fast visual representations of requirements (throwaway prototypes) or more complex ones (evolutionary prototypes). The latter can even become the early versions of the product that already have some pieces of the final code. Effectively, evolutionary prototypes may even turn into minimum viable products or MVPs that we’ve described in a separate article.

**Design documents and prototypes:**

Design requirements are usually collected and documented using three main formats that morph into one another:

**Wireframes.** Wireframes are low-fidelity graphic structures of a website or an app. They help map different product pages with sections and interactive elements.

**Mockups.** Once wireframes are ready, they are turned into mockups, visual designs that convey the look and feel of the final product. Eventually, mockups can become the final design of the product.

**Design prototypes.** These documents contain visuals and allow for some interface interactions, like scrolling, clicking on links, or filling in forms. Design prototypes can be built from scratch using HTML and CSS, but most UX teams use prototyping services like InVision.