# Understanding and Meeting Requirements

The start of any software project is defining what the software will do. It may sound like a ’duh’ but defining a project can be difficult, but a complete and well-defined set of requirements can make the difference between a successful project and an embarrassing failure.

The first step is defining the Requirements for a project. They will represent the interests of several Stakeholders in the project: for example, clients, developers, managers and users each wanting different things.

It can be helpful to split the requirements into two broad categories: Functional and Non-Functional. The line separating these can be thin, but the differences are important to remember. Broadly speaking Functional Requirements explain how the system must work and Non-Functional Requirements explain how the system should perform.

A successful product needs both these sets of requirements to perform and be accepted and praised by users.

## Functional Requirements

Functional Requirements define the features and functions of a solution to ensure it behaves as expected. Functional Requirements should be:

* User-centric – focusing on what the users needs rather than how it should be built.
* Specific and measurable – they should be clearly defined and allow for evaluation during testing.
* Complete – they should cover all the essential functions of the system.

Requirements are often categorised depending on what they define:

* **Inputs/outputs –** these specify what data the system can accept and how it can be formatted, and what data the system should produce and how it should be formatted. For example, it will accept a csv of bank transactions and output a html page including a table of the transactions**.**
* **Calculations Required –** these specify how the system should transform the input data into output data. For example, applying a specific algorithm to calculate derived data.
* **Technical specifications** – technical constraints for the project including hardware requirements, software platforms and programming interfaces that must be supported. For example, the app must run under Windows 10, support STMP for email and need less that 8Gb of RAM.
* **Data manipulation needed** – the operations that change, organise or retrieve data. For example, using SQL to access data stored in a database.
* **Data processing** – the steps to collect, transform and output data. Will it be in real time or batched processing?

## Non-Functional Requirements

The Non-Functional Requirements of a project are not related to the systems functionality, but how the system should perform. They are critical for ensuring the systems usability, reliability and efficiency often influencing the overall user experience.

* **How not What** – These requirements should focus on how the system should allow users to perform tasks rather than how that tasks should be achieved.
* **Measurable** – we should be able to testing that the requirements have been met. For example, Response time of less than 2 seconds.
* **Can be conflicting** – for example a requirement of a highly secure system my conflict with a requirement of easy to use.

Again, it is useful to categorize these requirements:

* **Accessibility** - These specify how easy it should be for users to learn and use the system. Does the system allow for colour blindness in the UI or the deaf not hearing audible alerts.
* **Efficiency -** resource consumption for a given load. Avoid wasted effort, choose the best search algorithm to minimise resource use.
* **Reliability** - Define how likely the software is to work without failing for a given time. Bugs in the code, hardware failures, or problems with other system components reduce reliability.  What is the critical failure time under normal usage? Does a user need access to this all hours of every day?
* **Scalability** - Describe how the system's growth, such as more users, data, or transactions, won't affect its performance. Horizontal scaling is where more computational nodes are added (additional servers) and Vertical Scaling where each node is made more powerful (additional RAM or CPUs).
* **Maintainability** - How much time does it take to fix components, and how easily can an administrator manage the system?
* **Security** - Describe levels of authorization and authentication for different roles and protect against viruses and other threats. What encryption standards should be used and when? What sensitive information is stored? What legal requirements are needed to store credit card data?
* **Robustness** - In computer science, **robustness** is the ability of a computer system to cope with errors during execution and cope with erroneous input.
* **Performance** – how quickly does the system response to users’ actions? Do processes complete within an acceptable timeframe? Will the overnight payroll complete before the banks cutoff time, so people are paid on time?

## Testing

Once requirements have been defined, we can use testing to check that we have met them. Testing can be a complex processing but its use at all levels of software development can make development more efficient and effective.

* **Functional Testing** – ensure that a application feature meets the software requirements. Can user’s login? Is it possible to add new data?
* **Unit testing** – Test the smallest functional unit of code to ensure it works correctly.
* **Smoke testing**– Test if basic functionality works or burn up and let the smoke out?  Often used before continuing with other tests. Does it compile? Does it start?
* **Integration testing** – test how the different components work when put together. The printing system works independently, the reporting engine work independently but can we print a report?
* **Systems testing** – end to end testing of an entire system for performance-based metrics.
* **Non-functional testing** – testing for non-functional requirements such as performance, speed and resource consumption. The report is generated ok, but takes forever and consume all the RAM on the server, when its supposed to be quick and small.
* **Availability testing** – Evaluates that a system will stay usable to users over a period of time, or does it fail and need restarting every day.
* **Compatibility testing** – ensures that the software works across the different platforms it is required on. Does the webpage work on Internet Explorer and Firefox? Does the app work on Windows 10 and 11?
* **Configuration testing** – does the system work on different configurations of hardware and software? What is the minimum RAM needed?
* **Load testing** – How does the application work under different workload? Can we expect the game servers to run fine on launch day with a massive influx of users? Will the additional server come online as load increases?