

STUDY MODULE

Requirements Analysis & PRD

A simple guide to turning raw findings into clear requirements, then into a well-structured product document.

ADSI
2026

Contents

1. Introduction
2. Where Analysis & PRD Sit
3. The Analysis Stage
4. What Happens During Analysis
5. Functional vs Non-Functional Requirements
6. Prioritizing with MoSCoW
7. Actors & System Scope
8. What Is a PRD
9. The Seven Core Components of a PRD
10. Case Study: From Finding to PRD
11. Summary

How to use this module

Read the sections in order. Each one explains a single idea in plain language and, where useful, shows a short example based on the same case study: a simple point-of-sale (POS) system for a small business. By the end you should understand what happens during requirements analysis, what a PRD is, and how the two connect.

1. Introduction

Before we build any software, we must first understand what it should do and why. This module covers the early part of that work: analysing the requirements we collected, and writing them down in a clear document called a PRD (Product Requirements Document).

This stage sits after information gathering (interviews, observation, questionnaires) and before technical design. Everything here answers one question: what does the system need to do? The how of building it comes later.

Key terms. In this module, **SRS / SKPL** is the requirements document (the “what”), and **SDD / DPPL** is the design document (the “how”). A PRD belongs to the requirements side.

2. Where Analysis & PRD Sit

It helps to see the whole flow of development at a glance. Each step feeds the next:

- **Information Gathering** — collect raw data from users and stakeholders.
- **Analysis** — turn that raw data into clear, valid requirements.
- **PRD** — write those requirements into a structured product document.
- **Process Modeling (DFD)** — model the processes and data in more detail.
- **Design (SDD)** — describe how the system will be built technically.

This module focuses on the two middle-front steps: Analysis and PRD. Both still belong to the requirements side (SRS / SKPL). Technical design (SDD / DPPL) only begins afterwards.

3. The Analysis Stage

Analysis is the bridge between raw data and a finished requirements document. It takes the messy notes we gathered and turns them into requirements that are clear, valid, and ready to write down.

Raw data cannot become a PRD directly. It must be processed first. Without this step, a requirements document is just a pile of wishes that may not truly be needed.

A requirement is “valid” when it reflects a real need, does not conflict with other needs, and is specific enough to act on. Part of the job of analysis is to keep asking simple questions — who needs this, why, and how will we know it is done — until each requirement becomes clear.

The idea in three steps

- **Raw data** — complaints, interview notes, and observations.
- **Analysis** — the data is filtered, validated, and prioritized.
- **Clear requirements** — functional and non-functional needs, ready to write.

Why it matters: *if a shop owner says “the system is slow,” analysis turns that vague complaint into a measurable requirement, such as “search must respond in under 3 seconds.”*

4. What Happens During Analysis

During analysis we usually do the following activities. They do not have to happen in a strict order, and we often repeat them.

- **Filter and group.** Organize overlapping data into themes of related needs.
- **Validate.** Confirm each need is real, not a passing wish, and resolve conflicts.
- **Separate functional and non-functional.** Distinguish what the system does from its quality and limits.
- **Prioritize.** Rank the needs by importance, for example with MoSCoW (see Section 6).
- **Identify actors.** Determine who will interact with the system.
- **Set the scope.** Draw the line between what is inside and outside the system.

The result of these activities becomes the raw material for the PRD. In practice these steps are not a straight line: as we write things down, we often discover a missing need and go back to gather or analyse a little more. That looping is normal and healthy.

5. Functional vs Non-Functional Requirements

Requirements come in two kinds. Keeping them apart makes the PRD clearer and easier to check later.

Functional requirements — what the system does

These describe actions the system can perform.

- The system can record sales transactions.
- The system can print monthly sales reports.
- The system can search products by name.
- The system can manage stock data.

Non-functional requirements — quality and constraints

These describe how well the system works, or the limits it must respect.

- The system responds to searches in under 3 seconds.
- User data is stored encrypted.
- The interface is easy for non-technical users.
- The system is available 99% of the time during operating hours.

Quick test: if the sentence says the system “can do” something, it is functional. If it describes speed, security, ease of use, or availability, it is non-functional.

Three tips for writing good requirements

- **Be clear.** Use plain words anyone can understand, and avoid vague terms like “fast” or “easy” without a number.
- **Be testable.** Write it so you can later check whether it is met — “under 3 seconds” can be tested; “should be quick” cannot.
- **One need per line.** Keep each requirement to a single idea so it is easy to trace and prioritize.

6. Prioritizing with MoSCoW

Not all requirements are equally important. MoSCoW is a simple way to rank them and decide what fits in the product scope. The name comes from four categories:

- **Must have** — mandatory. The product fails without it. Example: record transactions.
- **Should have** — important, but can wait a little. Example: automated reports.
- **Could have** — nice to have if time allows. Example: display themes.
- **Won't have** — agreed to skip for now. Example: a mobile app.

Sorting requirements this way keeps the team focused on what truly matters and prevents the scope from growing out of control.

A simple way to apply MoSCoW is to list every requirement, then decide its label together with the users and the team. When time or budget is tight, the “Must have” items are built first, and the lower categories can wait for a later version.

7. Actors & System Scope

Actors

An actor is any party that interacts with the system. It can be a person, a role, or another system. For the POS example, the actors are the cashier and the shop owner.

Identifying actors early is useful because later modeling steps — such as data-flow diagrams — are built around who sends data into the system and who receives it.

Scope

Scope is the firm boundary between what the system is responsible for and what lies outside it. Stating the scope clearly protects the project from “scope creep” — the tendency for new requests to keep piling on.

Example: *the POS system covers transactions, product and stock management, and sales reports (in scope). Tax-accounting integration and a mobile app are out of scope for now.*

8. What Is a PRD

A PRD (Product Requirements Document) distills the results of analysis into a structured form. It answers a single question: what product will be built, and why?

A few things make a PRD useful:

- It is product- and business-oriented, not deeply technical.
- It is written as readable, narrative text so anyone can follow it.
- It becomes the input for later modeling and design.

Because a PRD focuses on what and why (not how), it stays useful whether the team later uses a structured approach (DFD) or an object-oriented one (UML).

A PRD and an SRS are closely related. In many courses the PRD is the more product- and business-focused document, while the SRS / SKPL is the more formal, technical statement of requirements. In practice the PRD often feeds into the SRS — but both live on the requirements side, before design.

9. The Seven Core Components of a PRD

A typical PRD contains the seven components below. You do not need heavy formatting — clear headings and short, numbered requirements are enough.

1. Vision / Goal

States why the product exists and which problem it solves.

Example: *“Provide a digital POS system so small businesses can record transactions and stock automatically, quickly, and with fewer errors.”*

2. User Persona

A short, semi-fictional profile of a user type: who they are, their role, and their problem.

Example: *Sari, 35 — grocery-shop owner. Not tech-savvy. Needs a simple system that is quick to learn.*

3. User Story

Requirements written from the user’s point of view, using a fixed format: **As a [role], I want to [action], so that [benefit].**

- As a cashier, I want to search products by name, so that transactions go faster.
- As a shop owner, I want to view daily sales reports, so that I know how the business is doing.

4. Features

The product's capabilities, usually grouped by module — for example a transaction module, a product module, a stock module, and a report module.

5. Functional & Non-Functional Requirements

The needs from Section 5, written neatly and given codes so they can be traced later. For example **FR-01** “record transactions” or **NFR-01** “search response under 3 seconds.”

6. Scope

A clear statement of what is in scope and what is out of scope, as described in Section 7.

7. Success Metrics

Measurable signs that the product works. Examples: transaction time reduced by 40%, 90% of users finish a sale unaided, and zero stock discrepancy at month-end.

10. Case Study: From Finding to PRD

This short example shows how one small complaint becomes a proper PRD item. It ties together everything in this module.

Step 1 — Gathering finding

The user said: *“I often miscount change and forget when items run out of stock.”*

Step 2 — Analysis result

We translate the complaint into needs and prioritize them: automatic transaction recording (Must) and low-stock alerts (Should).

Step 3 — Into the PRD

We record it as a user story plus transaction and stock features, with a clear success metric: stock discrepancy equals zero.

In short: one user complaint is traced into a requirement, then framed cleanly in the PRD — complete with a way to measure success.

11. Summary

The main points to remember from this module:

- Analysis turns raw data into requirements that are clear, valid, and prioritized.

- A PRD distills those needs: vision, persona, user story, features, functional and non-functional requirements, scope, and success metrics.
- Analysis and the PRD work back and forth — it is normal to loop back and refine.
- Both belong to the requirements side (**SRS / SKPL**, the “what”), not yet the design side (**SDD / DPPL**, the “how”).

Quick checklist: is your PRD complete?

Before moving on, check that your PRD contains all seven parts:

- A clear vision or goal that states the problem being solved.
- At least one user persona describing a real type of user.
- User stories in the “As a... I want to... so that...” format.
- A list of features, grouped by module.
- Functional and non-functional requirements, each with a code.
- A scope statement listing what is in and what is out.
- Success metrics you can actually measure.

Up next: *process and data modeling with DFD (Power Designer).*