🧠📘 STUDY JOURNAL – Requirements Gathering and Analysis

Author: Abner Soberon
Date: April 16, 2025
Context: Personal deep-dive study journal for professional growth as a technical AI Product Manager.
Purpose: To reinforce and expand understanding of the requirement gathering and analysis process, with practical insights and structured breakdown.

🔹 1. INTRODUCTION

The first step in building any software system is understanding **what the client needs**. Before writing a single line of code, a business analyst or project team must gather and analyze the requirements that define the product.

This process happens during the **Analysis Phase** of the Software Development Life Cycle (SDLC). This phase transforms abstract client expectations into actionable, structured, and technically feasible requirements that guide the development process.

🔹 2. REQUIREMENTS GATHERING

Requirements gathering refers to the **systematic collection of needs, goals, and expectations** from the client. These needs must be captured in a clear, specific, and unambiguous way to ensure the development team builds exactly what the client expects.

🔸 2.1 Methods of Gathering Requirements

Method	Description	Strengths	Limitations
Interviews	One-on-one or group sessions where analysts ask clients direct questions.	Deep, flexible, real-time clarification.	Time-consuming; may miss unspoken needs.
Questionnaires	Written forms with predefined questions. Useful when clients are remote or numerous.	Broad reach, efficient for data collection.	Risk of vague, incomplete, or misunderstood responses.

Document Searches	Analysts review existing business documents, reports, or manuals.	Helps understand existing systems and processes.	May be outdated or lack context.
Observations	Analysts shadow users and observe how work is done.	Reveals real workflows, implicit tasks.	Time-intensive; behavior may change under observation.
Meetings	Structured sessions for discussing and validating requirements.	Encourages collaboration and alignment.	Requires careful facilitation to stay productive.

🔸 2.2 Types of Requirements

Type	Description	Example
Business Requirements	High-level goals from a business perspective.	“The company must increase online sales by 30%.”
Functional Requirements	Specific features or functions the software must perform.	“The system must allow users to reset their password.”
Non-Functional Requirements	Qualitative aspects like performance, security, or usability.	“Each transaction must complete within 3 seconds.”

🔹 3. FEASIBILITY STUDY

A **feasibility study** evaluates whether a proposed project should move forward. It examines **technical, operational, economic, and schedule aspects** to identify risks and determine viability.

🔸 3.1 Types of Feasibility

Type	Focus	Question Answered
Technical Feasibility	Technology and resources.	Do we have or can we get the tools and infrastructure?
Operational Feasibility	Functionality and effectiveness.	Will the product fulfill the user’s operational needs?
Economic Feasibility	Costs and financial return.	Will the benefits outweigh the development costs?

Schedule Feasibility	Project timeline.	Can it be realistically completed on time?

> 💡 *A feasibility study provides clarity and confidence before heavy investment in development.*

🔹 4. SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

An **SRS document** formalizes all gathered and refined requirements. It acts as a **contract between the client and development team**, ensuring all stakeholders agree on what will be built.

🔸 4.1 Benefits of an SRS

- Establishes shared expectations.
- Reduces development effort by avoiding misunderstandings.
- Serves as a reference for design, testing, and future enhancements.
- Supports cost and time estimations.

🔸 4.2 Characteristics of a Good SRS

Attribute	Description
Correct	All information reflects real client needs.
Precise	Clearly stated, specific, and unambiguous.
Complete	Covers all scenarios, inputs, and outputs.
Traceable	Every requirement is linked to its origin and tracked through the project.

🔸 4.3 Structure of an SRS

I. Introduction

- **Purpose**: Objectives of the document.
- **Scope**: Overview of the software, name, function, and benefits.

- **Definitions**: Technical terms and abbreviations.
- **References**: External documents or standards used.
- **Overview**: Summary of the document's structure.

II. General Description

- **Product Perspective**: Relation to existing systems.
- **Product Functions**: Summary of what the software will do.
- **User Characteristics**: Who the users are and their capabilities.
- **Constraints**: Hardware, OS, policy limitations.
- **Assumptions**: External conditions presumed to remain true.

III. Specific Requirements

- **External Interfaces**: UI, hardware, software, communication interfaces.
- **Functional Requirements**: Detailed system functionalities.
- **Use Cases**: User-scenario-driven descriptions of functionality.
- **Non-Functional Requirements**: Performance, reliability, availability.
- **Inverse Requirements**: What the system must not do.
- **Design Constraints**: Organizational or regulatory restrictions.
- **Database Requirements**: Data structure, formats, integrity.
- **Other Requirements**: Anything not previously captured.

🔹 5. ANALYZING REQUIREMENTS

🔸 Overview

This step ensures the gathered requirements are **technically valid, complete, and implementable**. It bridges the gap between planning and actual development.

🔸 Two Levels of Analysis

Level	Focus	Activities
System-Level	Broad organizational and system goals.	Detect inconsistencies, gaps, ambiguities. Hold meetings. Document and resolve open issues. Create supporting prototypes if necessary.
Software-Level	Specific product design.	Define architecture, performance, data models, and behavior. Drives system design.

🔹 6. INSIGHT: LOGICAL PROBLEM-SOLVING

> 💡 Sometimes, a problem contains the seeds of its own solution.

In one exercise, by analyzing the phrasing of each statement and identifying keywords (e.g., *financial* → *economic*, *resources* → *technical*, *time* → *schedule*), I was able to deduce the correct matches **without relying solely on memory**.

This technique can be generalized to many real-world problems: if we read carefully, analyze logically, and break the elements down, the solution often becomes self-evident.