IT Project Guidance

On Analysis and Requirement Documentation Development

Version: 0.2

Purpose

This document provides structural guidance and recommended formats for developing comprehensive requirement documents in IT projects. It serves as a reference for Business Analysts (BAs) to ensure clarity, completeness, and alignment of project objectives.

This document is intended for Business Analysts and Solution Architects involved in IT project planning and execution. It is a specialized extension of the more general underlying '*IT Project Guidance - On Professional Document Development*' and assumes familiarity with its contents.

Synopsis

Requirement documents are foundational to contractual work in IT projects. This guidance outlines the minimal traditional BUST model (Business, User, System, Transitional) but also the more comprehensive SBBUQCTST[1] model (Stakeholders, Business, User, Qualities, Capabilities, Transitional, System, Technical). It emphasizes the importance of capturing all relevant requirement types to reduce project risk and improve delivery outcomes. The document is part of a broader series and will be split into targeted versions for BAs and SAs, ensuring each group is aware of the full documentation set.

[1] Admittedly unpronounceable, however, valuable.

Contents

Context

Analysis leading to Requirement documents are the basis of contractual work in IT projects. They begin with capturing a business's external and internal drivers, including the current state, and progress through documenting business intent as Business Requirements. These are followed by User Requirements that contribute to the desired outcomes, Technical Requirements for systems to automate contributions, and specifications for qualities of service.

Traditionally, the International Institute of Business Analysis (IIBA) refers to a minimal set of requirements summarized as BUST: Business, User, System, Transitional. However, given the high failure rate of IT projects on key axes such as cost, features, and timelines, this minimal set is often insufficient.

A more complete analysis set is recommended: SBUQCTST, which includes Stakeholders, Business, User, Qualities, Capabilities, Transitional, System (Functional and Non-Functional Requirements), and Technical details. While this document does not delve into how to develop each type of requirement document, it outlines the recommended structure and guidelines for each category.

This document is part of a broader series, including '*IT Project Guidance - On Business User Requirement Development*', '*IT Project Guidance - On User Requirement Development*', and others.

Alignment

This guidance is grounded in internationally recognized frameworks and standards, including:

- **IIBA BABOK Guide**: The document reflects the BABOK's core knowledge areas and requirement classifications, including business, stakeholder, solution, transition, and non-functional requirements. It supports traceability, stakeholder engagement, and structured elicitation techniques such as SWOT, PESTLE, MoSCoW, SMART and BUST.
- **PRINCE2 Project Management Methodology**: The emphasis on business case development, governance, version control, and documentation structure aligns with PRINCE2's principles of justification, control, and product-based planning.
- **ISO/IEC 250xx Quality Model**: The treatment of service quality and non-functional requirements is consistent with ISO/IEC 25010/12/22, including categories such as performance, reliability, usability, and maintainability.
- **Agile Principles**: The document clarifies Agile's stance on planning and analysis, supporting incremental and iterative requirement development while maintaining discipline and traceability.

- **Domain-Driven Design (DDD)**: The use of domain modelling and capability decomposition reflects modern system design practices that align IT systems with business concerns.

This guidance is suitable for use in both structured and adaptive delivery environments. It supports contractual rigor, stakeholder alignment, and delivery traceability—making it appropriate for enterprise, public sector, and regulated project contexts.

Planning Is Not Optional: Clarifying Agile's Intent on Requirements

A widespread and unfortunate misconception is that Agile methodologies discourage upfront analysis, leading some teams to mistakenly believe that system and transitional requirements can -- or even should -- be developed entirely on the fly. This misunderstanding often results in a MVPs, unfortunately the wrong kind: *Missing Valuable Planning*.

Agile, in any of its forms, has never advocated for skipping analysis. While it rightly cautions against excessive upfront analysis and design, it also emphasizes the importance of capturing what is knowable and necessary early on. Agile simply encourages deferring decisions about uncertain or evolving requirements—not ignoring what is already clear and required. In fact, also widely misunderstood is that traditional "waterfall approaches" -- albeit less vocally-- *also* recommend allowing for iterative later analysis for initially unknown aspects.

This document does not promote Agile or Waterfall. It promotes **quality requirements**—whether developed upfront or incrementally.

Quality means using structured frameworks to elicit broad, thoughtful consideration of what is needed, and organizing those needs for prioritization and delivery. Whether requirements are documented formally or captured in a Kanban tool is beside the point. What matters is that they are achievable, pragmatic, identified, traceable.

In practice, a requirements document might reference a live report generated from a Kanban board instead of a spreadsheet. The format is flexible. The discipline of analysis is not.

Tools Are Not the Method: A Warning on Modelling and Documentation

Just as methodology is not a substitute for analysis, tooling is not a substitute for documentation. A common misstep in IT projects is over-reliance on modelling tools for requirement traceability and management—often at the expense of producing proper, contractual documentation.

While modelling tools can facilitate traceability across requirement types (e.g., Business → User → Capability → System), many practitioners use them only for linking requirements, without understanding the broader discipline of documentation. This becomes especially problematic when tools are chosen for their modelling features but lack support for versioning, formatting, review workflows, and contextual preambles—all of which are essential in contractual environments.

Requirements documents are **contractual artifacts**. They must be versioned to preserve fidelity across draft, agreed, and in-progress states. They must include context, background, and rationale that modelling tools cannot adequately support. Text boxes in modelling software are not a substitute for structured documentation—they lack formatting depth, are difficult to review collaboratively, and do not support commenting or change tracking.

Presentation matters. While internal reports may tolerate minimal formatting, contractual documents—especially those subject to public scrutiny or government obligations (e.g., OIA)—must meet professional standards. Poorly formatted tool-generated reports can become liabilities.

That said, it is entirely acceptable—and often beneficial—to reference a traceability table or report generated from a modelling tool, as long as versioning is rigorously managed. This is no different from referencing an external spreadsheet. The key is to ensure that the document remains the authoritative source, with the tool output serving as a supporting artifact.

This principle is revisited in the Requirements Traceability Matrix (RTM) section later in this document, where traceability practices are discussed in detail. For now, remember: don't fall in love with a tool so much that you lose sight of the contractual and analytical purpose of requirements documentation.

Business Analysis Document Types

We're not going to define what a Business Analyst *does*—but we will outline the core types of documents they typically produce. At a minimum, every BA should be familiar with the BUST model: **Business**, **User**, **System**, and **Transitional** requirements.

However, in practice, a more complete view includes:

- Stakeholder Analysis / Map
- Problem Statement / Opportunity Analysis
- Business Requirements
- Business Case
- User Requirements
- Capability Requirements

- Quality Requirements (non-functional)
- Transitional Requirements
- System Requirements (functional refinements of capabilities)
- Technical Requirements (implementation constraints)

As described in *IT Project Guidance – On Professional Document Development*, these documents often follow a general structure: front matter (purpose, scope, audience), followed by a body section that lists and elaborates on the requirements—grouped by type. Appendices may include traceability matrices, models, or supporting data. We won't be re-covering those parts in detail again here.

Stakeholder Analysis / Map Document

After the Front matter, the document proper's structure to use for a stakeholder analysis is as follows:

1. Purpose and scope of the analysis
2. List of stakeholders (individuals, roles, or groups)
3. Stakeholder interests, influence, and expectations
4. Engagement strategy or communication plan
5. Visual map or matrix (optional)

This document sets the stage for all requirements work. Be specific about roles and their influence on the project. Avoid vague labels like "users" or "management"—name actual roles or personas. A visual map helps clarify relationships and power dynamics. Keep it updated throughout the project, as stakeholders shift.

The document will include a stakeholder map. A stakeholder map visually represents the individuals or groups who have a stake in a project, organized by their level of influence and interest. It helps BAs identify who to engage, how often, and in what way.

Within the document, a Stakeholder Map is a visual tool used to categorize stakeholders based on their **power** (ability to influence decisions) and **interest** (level of concern or involvement). Unlike RASCI, which defines roles and responsibilities, stakeholder mapping focuses on engagement strategy. The classic 2x2 grid helps BAs determine how to manage each stakeholder group—whether to engage closely, keep informed, or monitor. This map should be updated regularly as stakeholder dynamics evolve.

Note that while RASCI is about roles, it's not the same as influence. So here the matrix will show

- **High Power / High Interest** – Manage closely
- **High Power / Low Interest** – Keep satisfied
- **Low Power / High Interest** – Keep informed

- **Low Power / Low Interest** – Monitor

Business Case

The Business Case is the formal justification for initiating a project. It defines the rationale, expected benefits, costs, risks, and alignment with strategic goals. It is not a requirements document, but it is deeply informed by them—especially Business Requirements and early stakeholder analysis. The Business Case is often the first document reviewed by decision-makers and funding bodies, and it must be clear, concise, and defensible.

A well-structured Business Case ensures that the project is not only desirable but also viable and achievable. It provides the foundation for governance, prioritization, and accountability throughout the project lifecycle.

Recommended Structure for a Business Case Document:

1. **Introduction**
    - Purpose of the business case
    - Background and context
    - Link to strategic goals or organizational priorities
2. **Problem Statement or Opportunity Analysis**
    - Description of the current state
    - Pain points, inefficiencies, or missed opportunities
    - Supporting data or evidence
3. **Proposed Solution Overview**
    - High-level description of the proposed change or system
    - Summary of how it addresses the problem or opportunity
    - Alternatives considered (if applicable)
4. **Expected Benefits**
    - Tangible and intangible benefits
    - Business outcomes (e.g., cost reduction, improved compliance, increased customer satisfaction)
    - Alignment with Business Requirements
5. **Costs and Resources**
    - Estimated budget
    - Resource requirements (internal and external)

- Funding source or model

6. **Risks and Mitigations**
    - Key risks (delivery, adoption, technical, legal)
    - Mitigation strategies
    - Dependencies and assumptions
7. **Timeline and Milestones**
    - High-level delivery schedule
    - Key phases or decision points
8. **Governance and Approval**
    - Decision-making roles
    - Review and approval process
    - Link to stakeholder engagement strategy

Notes on Format and Usage

- The Business Case should be version-controlled and treated as a formal document.
- It may reference other documents (e.g., stakeholder analysis, business requirements) but should remain self-contained.
- While it may be supported by modelling tools or spreadsheets, the document itself must be readable, reviewable, and presentable—especially in contractual or public sector contexts.

Business Requirements Document

The structure to use for a business requirements document is as follows:

1. Introduction (purpose and business context)
2. High-level business goals or drivers
3. Business rules or policies
4. Business requirements (numbered list)
5. Dependencies or constraints

These are not *system* features—they describe what the *business* needs to achieve. Use language that reflects business outcomes, not technical solutions.

Business requirements describe what the organization needs to achieve—not what the system should do. They should be expressed as universal outcomes that are meaningful

across industries and technologies, expressed in change direction (decrease, increase) by a percentage if possible. Common examples of these include: Decrease time,

Decrease dependencies, Decrease resources required, Reduce resource usage, Reduce cost, Increase speed of delivery, Increase qualities (see ISO-25010/12), expand market reach, expand market groups, strengthen compliance, improve interoperability, expand customer satisfaction (see ISO-25022), etc.

To help elicit these outcomes, consider using the **PESTLE** framework (Political, Economic, Social, Technological, Legal, Environmental) to explore external drivers, and **SWOT** (Strengths, Weaknesses, Opportunities, Threats) to understand internal motivators. These aren't requirement models per se, but until there is one, they're excellent for surfacing the forces behind business needs.

Each requirement should have an id (to support traceability back from user requirements developed next), a singular statement, potentially explanation and supporting rationale.

Avoid mixing these up with user or system requirements — stay at the business outcomes level.

While Business Requirements should be simple statements of intent "Increase|Decrease X by Y by Z", it is maybe still worth recommending avoidance of the use of "And", "All", "Shall", "Will". Use "If" sparingly. If Obligations and Prohibitions are required to be defined use MoSCoW terms - avoiding Recommendations and Permissions as they rarely survive work effort triage.

Refer to the companion *IT Project Guidance – On Developing Business Requirements* for further information, advice and examples.

Service Quality Requirements

Quality requirements define the qualities of service of the *business*, without specifying the *system* itself.

The structure to use for a quality requirements document is as follows:

1. Introduction (why quality attributes matter, what framework is being used to elicit and organise them)
2. Categories (e.g., performance, security, availability)
3. Specific quality requirements (numbered list)
4. Measurement criteria or thresholds
5. Risks or trade-offs

“Open for service 24/7, 99.99% of the time”. “Provide written response to queries within 2 days”. “Maintain 99.99% accuracy in records collected.”, “If interrupted, service will be restored within 24 hours.”, etc.

Warning: the development of these qualities can be informed by ISO-25010, however, not must not stray into becoming system based. They must remain business qualities, devoid of systems used to automate them.

Refer to the companion *IT Project Guidance – On Developing Service Quality Requirements* for further information, advice and examples.

User Requirements

User requirements define what users need to do to contribute to the business objectives being achieved. It is important that they are developed without reference to a specific system, or even a system.

The structure to use for a user requirements document is as follows:

1. Introduction (who the users are and what they need)
2. User personas or roles
3. User goals or tasks
4. User requirements (numbered list) categorised and organised by use case then user group
5. Usability or accessibility considerations

Write from the user’s perspective. Use plain language and avoid system jargon. If possible, tie each requirement to a persona or role. Think in terms of what the user wants to do—not how the system will do it. Include edge cases or accessibility needs where relevant. Most importantly, think of all user groups throughout the service’s lifespan: public, consumers, service providers, admins, support, operations, delivery, monitors, maintenance, development.

Work through default use cases that may involve one or more user stakeholder groups, including thinking about signing in, configuring, provisioning, inviting, onboarding, identifying, authenticating, authorising, routing, creating, contributing, describing, publishing, reviewing, categorising, linking, measuring, consuming, rating, commenting, correcting, maintaining, merging, replacing, removing.

Each requirement should have an id to later facilitate traceability from functional requirements, a title or singular statement, an expansion or explanation, and a rationale.

For traceability reasons, each requirement should have a footer that references the Business Requirement(s) it is contributing to.

Requirements should be developed as SMART requirements as much as possible. Singular,

Specifically, avoid the use of “And”, “All”, “Shall”, “Will”. Use “If” sparingly. If Obligations and Prohibitions are required to be defined use MoSCoW terms - avoiding developing Recommendations and Permissions as they rarely survive work effort triage.

It is relatively common for documents to be developed that embed the requirements as a table within the document or refer to an external spreadsheet of requirements. Both approaches are acceptable (documents offer more editability at the cost of a sortability).

Refer to the companion *IT Project Guidance – On Developing User Requirements* for further information, advice and examples.

Capability Requirements

System capabilities requirements are often overlooked. We strongly suggest omission is a key reason for poor system requirements.

Capabilities define what a system has to have like capabilities to support the User’s actions. Capabilities are areas of concern, and are not to be confused with the finer grain functions that make that capability available to users.

The structure to use for a capability requirements document is as follows:

1. Introduction (scope of capabilities)
2. Capability list (grouped by domain or function)
3. Description of each capability
4. Success criteria or KPIs
5. Dependencies or constraints

Capabilities describe what the system must capable of doing to some effect—not how it does it. They are broader than functional requirements and often map back to user requirements outcomes. Use consistent naming and avoid overlap. Capabilities should be stable even if the implementation changes.

It is relatively common for documents to be developed that embed the requirements as a table within the document or refer to an external spreadsheet of requirements. Both approaches are acceptable (documents offer more editability at the cost of a sortability).

Using *domain driven design* analysis processes is fundamental here. Domains to work through - in almost all cases - will include:

- system (diagnostics, configuration, storage, caching, discovery, access, routing, authentication, authorisation, search, description)
- resources (registries, records and their metadata, nestable collections)
- social (people, personas, relationships, group memberships),

- social activities (aspirations, coordination, projects and tasks, progress and completion assessments),

Refer to the companion *IT Project Guidance – System Capabilities Requirements* for further information, advice and examples.

Transitional Requirements

Transitional requirements define the requirements for the project to deliver the automation desired. These requirements are specific to the *project*, not the *service or system*.

The requirements are intended to be adhered to by the organisation delivering the project or business service, not the system vendor or implementor.

These are temporary project change management specific, but essential. They describe what's needed to move from the current state to the desired state. Think data migration, training, or phased rollouts. Be clear about when each requirement becomes obsolete. Don't bury these in other documents—they deserve their own space.

Transitional requirements are intended for more than one responsible organisation. Some are for the project sponsors itself, to make available resources and data to the vendor, others can only be implemented by the vendor.

The structure to use for a transitional requirements document is as follows:

1. Introduction (context of transition)
2. Current vs. future state comparison
3. Transitional requirements (numbered list)
4. Timing or sequencing considerations
5. Risks or dependencies

It is relatively common for documents to be developed that embed the requirements as a table within the document or refer to an external spreadsheet of requirements. Both approaches are acceptable (documents offer more editability at the cost of a sortability).

Refer to the companion *IT Project Guidance – On Developing Transitional Requirements* for further information, advice and examples.

Warning: Transitional requirements are often conflated with System Non-Functional Requirements. They are not the same. The service implementor can agree to service functional and non-functional requirements, and a portion of transitional requirements, but not all of them.

System Requirements

System Requirements represent a logical grouping of two distinct but complementary sets of specifications: Functional Requirements, which describe what the system must do, and Non-Functional Requirements, which define how the system must perform. These may be documented separately or combined, depending on organizational standards and project complexity.

Functional Requirements

Functional Requirements are discrete, testable behaviours that define how the system supports each capability identified earlier. These requirements are actionable for developers and should be expressed in a consistent format, using clear numbering and traceability to user needs or business goals. They must avoid design bias—focusing on *what* the system must do, not *how* it will do it. Functional requirements often expand into a long list -- especially when decomposing broad capabilities into specific operations such as Create, Read, Update, and Delete (CRUD).

These are the detailed, actionable requirements for developers. Each should be testable and traceable back to a capability or user need. Use consistent formatting and numbering. Avoid mixing in design decisions—focus on what the system must do, not how it will do it.

As such, the functional requirements become a longer set of requirements.

The structure to use for a system requirements document is as follows:

1. Introduction (background, scope and context)
2. Functional requirements (refined from capabilities)
3. Use cases or user stories (optional)
4. Data requirements or models
5. Interfaces or integrations

Examples of these are:

“The system MUST allow Administrators to create new User accounts.”

“The system MUST allow Administrators to disable User accounts.”

“The system MUST allow Administrators to retire User accounts.”

“The system MUST allow Administrators to restore User accounts.”

“The system MUST NOT allow users to permanently delete User accounts.”

It is relatively common for documents to be developed that embed the requirements as a table within the document or refer to an external spreadsheet of requirements. Both approaches are acceptable (documents offer more editability at the cost of a sortability).

Warning: it is all to common for analysts to confuse Functions for Capabilities and only provide System Capabilities Requirements (the System MUST manage Users."). This often proves insufficient, and source of disagreement as to scope of delivery.

Non-Functional Requirements

Non-Functional Requirements define the system's quality attributes—such as performance, security, usability, and reliability—that enable the service levels expected by stakeholders. Organized using the ISO/IEC 25010 framework, these requirements influence architecture and design decisions and are critical to system acceptance. They must remain distinct from Transitional Requirements, which relate to project-specific implementation logistics.

The structure to use for a non-functional requirements document is as follows:

1. Introduction (background, scope and context)
2. Non functional quality requirements (following ISO-25010 organisation)

It is relatively common for documents to be developed that embed the requirements as a table within the document or refer to an external spreadsheet of requirements. Both approaches are acceptable (documents offer more editability at the cost of a sortability).

Warning: do not conflate *project* Transitional Requirements (resourcing, migration, onboarding, accreditation, etc.) with *System* Non-Functional Requirements.

Technical Requirements

The structure to use for a technical requirements document is as follows:

1. Introduction (technical context or constraints)
2. Platform or environment requirements
3. Integration or API constraints
4. Security or compliance requirements
5. Deployment or infrastructure needs

These are often driven by enterprise standards or architectural decisions. Be precise — "must run on AWS" is clearer than "cloud-based." Include any constraints that affect design or delivery. If these are negotiable, say so. This document often overlaps with architecture work—coordinate accordingly.

Requirements Traceability Matrix (RTM) Document

The Requirements Traceability Matrix (RTM) is a structured mechanism for tracking the lifecycle of each requirement—from its origin (business or user need) through to its implementation (system or technical specification) and validation (test case or

acceptance criteria). It ensures that no requirement is lost, misunderstood, or left unverified.

However, this is also where a critical misstep often occurs.

While modelling tools can facilitate traceability by linking requirements across layers (e.g., Business → User → Capability → Functional → Technical), many practitioners use these tools superficially — focusing only on linkages without understanding the broader purpose of traceability: accountability, validation, and contractual clarity.

A common error is to prioritize the tool's modelling features at the expense of proper documentation. This is a mistake. Requirements documents are **contractual artifacts**. They must be versioned to preserve the fidelity of draft, agreed, and in-progress states. They must include context, rationale, and introductory framing—elements that modelling tools typically do not support well. Text boxes in modelling software are not a substitute for structured documentation. They lack formatting depth, are difficult to review collaboratively, and do not support commenting, change tracking, or formal approval workflows.

Moreover, **presentation matters**. While internal reports may tolerate minimal formatting, contractual documents—especially those subject to public or government scrutiny (e.g., under OIA obligations)—must meet professional standards. Poorly formatted tool-generated reports can become liabilities.

That said, it is entirely acceptable—and often beneficial—to reference a traceability table or report generated from a modelling or requirements tool, **as long as versioning is rigorously managed**. This is no different from referencing an external spreadsheet. The key is to ensure that the document remains the authoritative source, with the tool output serving as a supporting artifact.

Recommended Structure for an RTM Document

1. **Introduction**
    - Purpose of the RTM
    - Scope of traceability (e.g., which requirement types are included)
    - Tooling or sources referenced (e.g., spreadsheets, modelling tools, Kanban boards)
2. **Traceability Table**
    - A matrix showing relationships between:
        - Business Requirements
        - User Requirements
        - Capability Requirements
        - Functional Requirements

        - Non-Functional Requirements
        - Technical Requirements
        - Test Cases or Acceptance Criteria
    - Include unique IDs, requirement titles, and status indicators
3. **Change History and Versioning**
    - Track updates to requirements and their traceability links
    - Indicate when requirements were added, modified, or removed
4. **Assumptions and Limitations**
    - Clarify what is and isn't covered by the RTM
    - Note any known gaps or deferred traceability
5. **References**
    - Link to source documents, spreadsheets, or tool-generated reports
    - Ensure referenced artifacts are version-controlled and accessible

Conclusion

This document has outlined the structural guidance and rationale for developing comprehensive requirement documentation in IT projects. It has emphasized the importance of clarity, completeness, and traceability—regardless of methodology or tooling preference.

Whether requirements are captured in formal documents, spreadsheets, or live reports from modelling tools, the discipline of analysis must remain central. Requirements are not just technical specifications—they are contractual commitments, strategic enablers, and the foundation for successful delivery.

The SBUQCTST model presented here offers a more complete framework than traditional approaches, helping Business Analysts and Solution Architects ensure that no critical requirement type is overlooked. By following the structures and principles outlined in this guidance, teams can reduce project risk, improve stakeholder alignment, and deliver outcomes that are both technically sound and business-relevant.

Above all, remember: tools and methodologies are means, not ends. The goal is quality—achievable, pragmatic, and traceable requirements that support the development of work contracted to deliver the solution required to serve the business.

Appendices

Appendix A - Document Information

Authors & Collaborators

- Sky Sigal, Solution Architect

Versions

0.1 Initial Draft

0.2 Some corrections

Images

No table of figures entries found.

Tables

No table of figures entries found.

References

There are no sources in the current document.

Review Distribution

The document was distributed for review as below:

Identity	Notes
Russell Campbell, Project Manager	
Carrie Buckmaster, Senior Analyst	

Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

Structure

Where possible, the document structure is guided by either ISO-* standards or best practice.

Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

Acronyms

API : Application Programming Interface.

DDD

: Domain Driven Design

GUI: Graphical User Interface. A form of UI.

ICT: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

IT : acronym for Information, using Technology to automate and facilitate its management.

MoSCoW : MUST, SHOULD, COULD, WON’T

MVP : Missing Valuable Planning

MVP : Minimum Viable Product

OIA : Official Information Act

RTM : Requirements Traceability Matrix

UI : User Interface. Contrast with API.

Terms

Application Programming Interface : an Interface provided for other systems to invoke (as opposed to User Interfaces).

Business Analyst : an analyst of a business, a business service, and a business system. To ensure wider coverage, a more balanced term might be *Stakeholder Analyst*.

Capability : a capability is what an organisation or system must be able to achieve to meet its goals. Each capability belongs to a domain and is realised through one or more functions that, together, deliver the intended outcome within that area of concern.

Domain : a domain is a defined area of knowledge, responsibility, or activity within an organisation or system. It groups related capabilities, entities, and functions that collectively serve a common purpose. Each capability belongs to a domain, and each function operates within one.

Entity : an entity is a core object of interest within a domain, usually representing a person, place, thing, or event that holds information and can change over time, such as a Student, School, or Enrolment.

Function : a function is a specific task or operation performed by a system, process, or person. Functions work together to enable a capability to be carried out. Each function operates within a domain and supports the delivery of one or more capabilities.

Person : a physical person, who has one or more Personas. Not necessarily a system User.

Persona : a facet that a Person presents to a Group of some kind.

Requirement : a statement that defines of the following: an Obligation, Recommendation, Permission, or Prohibition.

Quality : a quality is a measurable or observable attribute of a system or outcome that indicates how well it meets expectations. Examples include reliability, usability, and performance. Refer to the ISO-25000 SQuaRE series of standards.

Service (IT) : an **IT Service** is a digitally delivered capability provided by an IT System. It supports or automates part of a *Business Service*, but is not the business service itself. IT Services are typically defined by their functionality, availability, and performance characteristics, and are managed to meet agreed service levels.

Service (Business) : a Business Service is a value-creating activity or offering delivered by an organization to its customers, partners, or internal stakeholders. It may be supported by one or more IT Services, but its definition and success criteria are rooted in business outcomes, not technology.

Stakeholder Analyst : refer to *Business Analyst*.

System (IT) : a collection of software, hardware, and data components that automate or support the delivery of one or more IT Services. It enables the execution of business processes by implementing specific functions and capabilities, often aligned with enterprise architecture standards.

System Capability : a high-level ability that an IT System must possess to support business or user needs. It is realized through a combination of functions and services, and remains stable even as implementation details evolve.

User : a human user of a system via its UIs.

User Interface : a system interface intended for use by system users. Most computer system UIs are Graphics User Interfaces (GUI) or Text/Console User Interfaces (TUI).