IT Project Guidance

Clear Thought in IT

Version: 1.3

Purpose

This document aims to address one of the key factors contributing to underperformance in government IT projects: the absence of shared, structured, and critical thinking in the earliest phases of problem definition and system design. It provides a practical framework to improve analysis and planning practices across agencies, while recognising the complex environments in which public sector initiatives operate. Rather than assigning blame, it highlights the opportunity to build clarity and rigour before any methodology, technology, or delivery model is selected.

Synopsis

Government IT projects are often challenged before they even begin—not primarily due to poor technology selection, resourcing gaps, or delivery methods, but because the core problem is not sufficiently understood or agreed. This document proposes that clear, structured analysis must precede design and delivery. It acknowledges the diverse pathways professionals take into business analysis, architecture, and leadership roles, and offers practical strategies to support more consistent thinking across disciplines. Drawing from systems thinking, information science, and enterprise architecture, the guidance presents a public-sector-suitable model of structured thought—emphasising domain clarity, temporal sequencing, information states, and stakeholder perspective.

Contents

Introduction

In the public sector, IT systems are almost always delivered to support essential services and operations. They carry weight beyond their immediate functionality. The expectation is not only that they work, but that they endure, integrate, and serve diverse user groups reliably over time. Despite this, large numbers of government IT projects fail outright, or struggle with significant overruns, missed expectations, or post-delivery abandonment.

The problem is rarely the technology. Nor is it usually the availability of capable developers or infrastructure. Instead, the root cause is typically a failure in the earliest phases of thinking: problem scoping, analysis, planning, and critical framing. What is required is not better methodology, but clearer thinking. Structured, methodical, rigorous analysis must underpin all solution activity, irrespective of the delivery framework selected.

This document responds to that need for clarity. Its purpose is to provide a structured, practice-oriented framework for improving critical analysis in the early phases of public sector IT projects. It aims to equip professionals—regardless of role—with tools to interrogate complexity, define purpose, and structure information before selecting methodologies or technologies.

Context

The solution lifecycle in government systems can be broadly described as traversing the following phases: Discover, Define, Design, Test, Deliver, Support, Operate, Maintain, and Replan. While Agile, Waterfall, and hybrid models interpret these stages differently, the presence of the stages themselves is unavoidable.

Clarity must begin at the beginning. Before a solution is even conceived, there must be shared understanding of the context, the desired future state, and the constraints that govern change. That understanding must be informed by structured inquiry and critical thinking. When it is absent, no project structure—no matter how robust—can deliver reliable outcomes.

A further trap is that many projects begin not with a vision of the future, but with an analysis of the current problem. This often anchors thinking to existing conditions, producing solutions that are small, incremental, and lacking ambition. Requirements elicited from current users in current roles tend to reinforce present-state constraints. Instead, projects must begin with a clear articulation of the future state—what should exist, not just what currently does. Working backwards from desired outcomes allows for a more transformative scope.

The alternative—practiced by Bruce Lee and others—is to begin from the outcome. In judo, this means placing the opponent on your back, then rewinding your body position to understand the movements required to achieve that result. In systems work, it means envisioning the future state first—then working backwards to identify the gap, and finally,

the appropriate starting point. This process—Future, Gap, Start—is far more reliable than the more common (and more dangerous) approach of Start, Gap… miss. Requirements gathered in that mode will inevitably reflect the past, not the future.

The Issue

Government projects fail at an alarming rate. International studies and audit reports consistently identify similar causes: ambiguous scope, changing requirements, low stakeholder engagement, unrealistic expectations, and poor alignment between business and technical teams. Beneath these symptoms lies a deeper, more pervasive issue: the lack of clear and shared understanding of the problem being solved.

In many government environments, Business Analysts play a central role in requirements development. Yet few operate with a consistently defined framework or shared professional standard. Many bring invaluable contextual knowledge from prior operational or support roles, but have had limited opportunity for formal training in structured analysis, domain modelling, or requirement classification. While their frontline experience is often essential, the absence of analytical training can reduce the rigour of requirements development over time.

Without formal grounding, analysts lack exposure to core analytical disciplines such as:

- Stakeholder analysis and the mapping of needs
- Differentiating between business, user, capabilities, qualities, system, transitional, and technical requirements
- Reliance on recognised frameworks to guide thought
- Recognising and evaluating assumptions
- Identifying underlying systemic patterns rather than surface-level symptoms

Additionally, the improper interpretation of Agile has led to a belief that planning is unnecessary. This is dangerously incorrect. Agile is a delivery process, not an analysis process, and presumes thoughtful discovery and definition of what can be defined has already occurred. It does not preclude structured upfront analysis, nor eliminate the need for progressive elaboration of known constraints and objectives.

Further complicating the issue, the domain complexity of most government services is rarely acknowledged. Education, health, justice, social services, and transport are all systems in which roles, responsibilities, funding, and outcomes are interdependent. To define a solution without recognising this interconnectedness is to court failure.

Government funding mechanisms reinforce fragmented visions by embedding disconnected initiatives. As a result, essential interconnectivity is left to emerge through

unplanned and unfunded efforts, which introduces risk and, more often than not, leads to failure.

Symptoms

The impact of unclear thinking becomes visible not just in delayed timelines or overrun budgets, but in the very structure of project behaviours. When discovery is rushed, or planning is treated as optional, a common set of failure patterns emerge. These symptoms are consistent across domains, delivery styles, and levels of project maturity: This absence of clear thinking manifests in multiple ways:

- Projects initiated without an agreed problem statement
- Solutions proposed before understanding user journeys or pain points
- Business Analysts unable to distinguish types of requirements
- Lack of transitional planning between current and future states
- Stakeholder disagreement on what constitutes "done"
- Failure to articulate measurable success criteria
- Misuse of MVP as a license to skip essential planning

Causes

Understanding why IT projects fail with such consistency requires more than pointing to symptoms. The underlying causes are structural — rooted in how organisations are formed, how roles are assigned, how planning is approached, and how information is handled. These causes span capability gaps, institutional habits, and misinterpretations of method. Recognising them is not about assigning blame, but about understanding the landscape in which failure repeats. Only by confronting these patterns directly can we begin to design projects that avoid them.

Analytical skills are not taught

Secondary and even tertiary education offer little in the way of structured critical thinking – and certainly not as it applies to systems design and service delivery. The absence of exposure to formal analysis frameworks leaves professionals unequipped to interrogate ambiguity.

Professional analysis frameworks are not adopted

The Business Analysis Body of Knowledge (BABOK) exists and is maintained by the IIBA, but is rarely referenced. Many analysts are unaware that requirements can be classified as business, user, system, technical, and transitional, or that transitional requirements deal with change logistics rather than system function.

Role confusion and perspective limits

Without proper framing, and partly due to their own title, analysts often assume their task is to consult only business representatives. Technical, support, and regulatory perspectives are ignored. This reduces the scope and clarity of analysis.

Misinterpretation of Agile

The myth that Agile projects do not require planning leads to skipped analysis, vague goals, and reactive delivery. In truth, Agile assumes the presence of a known goal and a plan to achieve it, but also a willingness to adjust based on ongoing discovery.

Clarifying the Purpose and Scope of MVPs

Agile's emphasis on Minimum Viable Product (MVP) often leads to the mistaken belief that discovery and analysis can begin with the interface or user experience. This reverses the logical order of design. Just as a watch is not designed starting from the dial but from the movement within, effective system design must start with an abstract model of functionality and behaviour—only later expressing this through user-facing components.

Product Owners must be supported in understanding that system success depends on defining internal structure, data flow, and state-change logic early—before engaging with visual or interaction-layer concerns. When design begins with the interface, there is a risk that visual and surface elements take precedence over the underlying structure. This can unintentionally prioritise what is seen over what is essential, reducing the clarity of logic, flow, and integration required beneath the surface.

A better metaphor comes from high-rise construction: one does not begin by choosing curtains. The process starts with boring into the ground for foundations, laying steel reinforcement, and installing core service access points—an effort-intensive and carefully sequenced phase that takes longer than most would prefer, and must be given time to complete correctly. Only after this groundwork is complete do walls rise, windows get fitted, and interiors are finished—steps that progress more quickly. Without this order, the building would be unstable. So too with systems.

Cognitive Distortion Through Stakeholder Bias

Stakeholders often conflate personal experience with systemic truth. Their utterances, while informative, must be interrogated and cross-validated. Without structured validation, subjective perspectives quickly solidify into unchallenged requirements, anchoring the solution in partial or distorted views.

Current-State Bias

One specific and powerful form of distortion is current-state bias. Stakeholders typically know only the systems, constraints, and workarounds they live with. Their framing of the problem is unconsciously anchored to what already exists—especially to what currently

frustrates or fails them. When discovery begins by documenting only these pain points, the vision becomes an attempt to fix the present, not to design the future.

Effective analysis requires resisting this gravitational pull. Future-state design must be treated as a distinct act—one that imagines what could be, not just what must be fixed. From there, gaps can be assessed and a transition planned. Without this separation, projects risk becoming just expensive incremental patchwork -- rather than transformational change.

Professional Capability and Role Boundaries

In many government environments, Business Analysts play a central role in requirements development. Yet few operate with a consistently defined framework or shared professional standard. Some arrive from operational or support roles—bringing invaluable contextual knowledge, but often without formal training in structured analysis, domain modelling, or requirement classification.

This is not a failing of individuals, but a system-level gap. Agencies have not always equipped BAs with a model of thinking that separates stakeholder perspective from systemic need, or user feedback from architecture. As a result, BAs may find themselves expected to articulate requirements across business, user, quality, system, and delivery domains—often without shared tools, consistent models, or clear expectations to support that work.

A similar pattern can be observed with many Solution Architects (SAs). They often arrive at the role through deep experience with the 'T' in IT—technology, infrastructure, and the implementation of system features. This technical strength is critical, but alone may not provide the grounding needed for full-system design. Structured exposure to the 'I'—Information—and to Integration principles takes time and support. As SAs grow in experience, their perspective typically broadens from individual systems to the wider ecosystems they exist within. This includes understanding cross-domain capabilities, lifecycle behaviour of information, and system interdependence across organisational, sector, and regulatory environments. Supporting this evolution helps ensure systems are not just functional in isolation, but resilient, interoperable, and aligned to long-term service goals.

Data Architects may also suffer from a similar bias. Many have risen through roles focused heavily on the management of data—emphasising storage, structure, analysis, and reporting. While this experience provides valuable rigour, it does not automatically equip them for systems design. Few receive structured training that supports abstraction, systemic modelling, or future-state planning. Nor are they often encouraged to consider storage as merely one component of a broader system rather than the central design concern. Without this shift, their designs may prioritise local schemas and access efficiency over long-term system evolvability, integration, and flexibility. To move beyond data warehousing and into strategic information architecture, Data Architects must loosen

their focus on storage primacy and shift toward abstraction and conceptual modelling—understanding information as a service, not data as a service—with lifecycles, domains, trust states, and access conditions. Information should be treated not merely as data to be queried, but as structured action-enabling knowledge within a dynamic system.

This guidance is intended to support that evolution. It does not aim to displace or reassign the responsibilities of Business Analysts, Solution Architects, or Data Architects, but to strengthen them—providing shared concepts, naming conventions, and sequencing logic that enable them to work with greater confidence and alignment across multi-disciplinary teams. When these roles are supported with consistent analytical models and structured thinking, they become among the project's most critical contributors—able to translate stakeholder needs into domain-bound, testable requirements; design systems that reflect strategic and cross-sector integration needs; and manage information as a lifecycle asset, not merely as technical data.

Critical Collection of Information

Clear thinking begins with trustworthy input. Before any structured analysis can occur, there must be disciplined collection of information from the right sources, and rigorous qualification of that information before it is used to guide design.

The first step is to identify the full range of relevant stakeholder groups. This includes not only business owners and operational users, but also lifecycle stakeholders—support teams, maintainers, testers, integration partners, governance bodies, and public representatives. For each group, a subject matter expert (SME) should be nominated to act as a qualified voice. Without this coverage, early design decisions risk being made from narrow or unchallenged perspectives.

Once SMEs are identified, information can be elicited through structured inquiry. However, the existence of a statement or data point does not make it useful. Information must be qualified. It should be tested for relevance, accuracy, and structure, and placed along a continuum—from unstructured utterances and discardable noise, through processable or storable data, to analysable insights and ultimately actionable knowledge.

It is equally critical that this process tests not only current perspectives, but future visions, by "writing back postcards from the future". Without deliberate inquiry into the envisioned end state—whether articulated by strategic leads, architects, or policy advisors—there is a risk that all elicited information simply validates or critiques the status quo. True critical thinking requires comparing stakeholder assertions against both real-world constraints and future-state aspirations. Testing whether the future state holds water is just as essential as clarifying the current state.

This qualification is not optional. The systems we build are themselves designed to handle information. They collect, process, store, and present information for others to use in decision-making. If the design process begins with flawed, vague, or unverified input, the resulting system will reflect that unreliability.

Additionally, clear thinking about information begins with treating all early inputs as provisional. Much of the data gathered during discovery is not yet verified; it represents perceptions, anecdotes, or institutional assumptions. If accepted uncritically, this material becomes embedded system behaviour. It must instead be interrogated—treated as hypothesis first -- and formalised only once tested.

Treating early inputs as provisional hypotheses protects against misalignment. Only once tested and contextualised should they be used to define scope or guide solution planning.

Improving Critical Thinking

With trustworthy information established, the work of analysis can begin. This next section introduces a structured approach to surfacing clarity, anchored in repeatable patterns of thought. It is not enough to gather data—we must apply disciplined reasoning to explore the structure of the problem, test assumptions, and uncover the relationships that will shape the solution. The following sections introduce a set of concepts and methodology aimed at improving critical thinking in the definition and delivery of IT systems.

Information as Foundation

Information is the core constant of information technology. It is the first letter of “IT” for a reason. While technologies—platforms, tools, formats, and vendors—change continually, the purpose of those technologies remains steady: to develop, collect, process, store, and disseminate information.

This makes information the foundation for all IT system design. Systems exist to manage the state of information, across time and across users. This is why one of the three key axes of thought is dedicated to it. Information itself is a formal discipline with decades of theory behind it. It is not a mystery, nor an improvisation—it is a structured, analysable domain. Mapping it correctly is the prerequisite for automating it effectively.

To understand information in any system, one must break it down into its parts: the entities involved (the domain-bound concepts), the states those entities pass through, and the roles that create, access, change, or approve that information. These are not arbitrary concerns—they are the pillars on which the reliability of the entire system rests.

Axes of Thought

Projects should be approached through intersecting axes of thought: domain, information, and role. These axes provide the analytical structure needed to explore problems and validate solutions.

- The Domain Axis is the cartography of the problem and emerging solution. It maps the structural landscape.

- The Information Axis captures how information changes state over time - from created, collaborated on, validated, qualified, trusted, changed, and used.
- The Role Axis reflects who acts upon information, when, and with what authority—across the information's lifecycle.

Together, these axes help surface the true structure of the submerged problem and form the basis for identifying the capabilities required. These capabilities, in turn, determine the functionality to be delivered. This sequencing—understanding the context and structure first, before identifying requirements of capabilities, qualities and functionality —is essential but often overlooked.

Framing Solutions through Domains (Axis 1)

One effective method for surfacing clarity is to break the problem down by domain. Rather than starting with roles or user stories, begin with the nature of the problem in relation to core general domains of human activity before completing what is absolutely specific about the current business context:

1. **System** – Diagnostics, configuration, integration points, data storage and caching, routing logic, users, permissions, identity, access control, notifications, queues, workflows, and settings.
2. **Social** – The People & their Personas involved, their Relationships, the Groups they form or belong to, and the Roles they take in each.
3. **Aspirations** – The goals, objectives, and milestones being worked toward or evaluated against.
4. **Artefacts** – The materials required, referenced, developed, or delivered in support of work and outcomes.
5. **Work** – The actions and collaborations that transform artefacts or produce progress toward aspirations.
6. **Assessments** – the qualify of completed work against aspirations.
7. **Organisation** – The coordination of resources, including people and time, across different places and periods.
8. **Use Case Specific** – Obligations, code sets, reference data, and specific workflow patterns that are unique to the domain context.

Each domain represents a form of foundational thinking—recognising that much of what we encounter in system design has been analysed and solved many times before. Human activity is not infinitely variable; it follows recurring patterns. The work of analysis is, in large part, the act of identifying which aspects of the current challenge belong to which well-understood domain. By doing so, we minimise what must be solved anew. This reframing shifts the problem from being novel and unconstrained to being largely structured and familiar, with only a small portion requiring invention or bespoke treatment.

Information State (Axis 2)

The second analytical axis concerns the state of information—how it behaves, evolves, and stabilises over time. Information is not static; it is created, reviewed, submitted, approved, rejected, versioned, archived, and ultimately retired. Each of these transitions constitutes a change in state that must be traceable, authorised, and often auditable.

State changes are not incidental—they define the operational logic of the system. For example, a learner application may move from 'draft' to 'submitted,' 'reviewed,' 'approved,' and 'enrolled'. These transitions are more important to system design than any single screen or feature. They anchor how workflows proceed and what actions are permitted at any given time.

These transitions are frequently governed by process constraints, timing rules, and validation logic. As such, understanding and mapping these information state models is not a business requirement—it's a design prerequisite.

Roles and Lifecycle Responsibilities (Axis 3)

The third axis focuses on who acts upon the information—and when. Information does not change state on its own. It is affected by people, processes, and systems that are granted specific responsibilities throughout the lifecycle.

From early planning, provisioning, and stakeholder engagement through to design, governance, testing, release, support, and retirement—each action depends on someone (or something) doing the right work at the right time.

This understanding of role-responsibility alignment is central to sound design. It ensures not just that functionality is delivered, but that responsibility for validation, exception handling, and assurance is properly distributed.

This aligns loosely with IIBA’s breakdown into Business, User, System, and Transitional requirements (BUST), though a more complete view may include Quality, Capability, and Technical aspects (BUQCSTT). Thinking clearly about roles and timing ensures we do not confuse stakeholder assertions with verified requirements, and that planning, testing, deployment, support, and governance are all represented and synchronised in delivery.

Beyond Axis – Systems, Standards, Purpose and Projects, Policies and Processes

Systems within systems

An automated system exists within business systems, which in turn exist within sector, professional, and legal systems. Each of these layers imposes its own obligations, expectations, and structures in exchange for interoperability and legitimacy.

It is therefore imperative to identify external obligations that lie beyond the project’s control but must be satisfied regardless. These include whole-of-government or jurisdiction-wide mandates such as privacy legislation, records management obligations,

official information acts, and financial reporting requirements. Others may be sector-specific—such as education, health, or transport compliance requirements—or operational mandates like cloud-first policies or cybersecurity frameworks.

Purpose versus Projects

Purpose is what drives transformation. It anchors the reason for change. Projects, by contrast, are the chosen vehicles for managing delivery and ensuring accountability. But while accountability can be diffused or delayed, purpose is binary—change happens or it does not. When delivery becomes detached from purpose, projects risk producing compliant outputs without meaningful outcomes. Anchoring project thinking in purpose ensures that the intended change remains central, even as delivery conditions shift.

Policies and Principles

Sliding from mandatory obligations to internal operating models, organisations must also distinguish between Policies and Principles. Policies embed specific processes and behaviours based on assumptions about a stable environment. Principles, in contrast, provide navigational intent—allowing groups to respond appropriately when conditions shift. Governance based on rigid policy enforcement often stagnates innovation. Governance based on conformance to well-framed principles enables adaptation while maintaining integrity. This difference is crucial for systems that must endure and evolve in changing environments.

Standards

While policies and principles are largely internal, standards represent cross-organisational agreements. They enable interoperability with external systems—by ensuring consistency in data formats, integration protocols, identity handling, and service expectations. Standards are not mandates, but they are invitations to collaboration. Ignoring them isolates the system; adopting them enables participation, portability, and future integration.

A building metaphor

Just as a house must comply with district plans, connect to city utilities, and meet national building codes, so too must a system acknowledge its broader context. External obligations define constraints but also create enablers. Standards allow systems to plug into shared platforms and services—offering reuse and interoperability. Purpose anchors the change being sought; Projects are merely the vehicles, and if detached from Purpose, they risk delivering activity without impact. Principles allow adaptive alignment to values; Policies, if overly rigid, may suppress discovery and adaptation. Together, they form the environment through which any IT solution must carefully navigate—not just to succeed, but to *matter*.

Structured Thinking in Practice

Discovery, Analysis, and Definition Stage

This stage forms the bridge between a problem's recognition and a description of its viable resolution. However, before a single stakeholder is identified or a requirement gathered, one concept must be made explicit: **Purpose**.

Purpose

Purpose operates at multiple levels. It begins with the organisation as a whole—why it exists, what mandate it serves, and what outcomes it is accountable for.

From there, purpose must be articulated at progressively narrower scopes: at the level of an agency, division or department, at the level of a particular business service, and finally at the level of the automation system being considered.

Without this hierarchy of purpose, core definitions fall apart. Terms like *Service Provider*, *Service Consumer*, and *Service Delivery* become ambiguous or circular. Projects risk optimising tools for local utility without understanding whether those tools contribute meaningfully to broader organisational objectives.

Value

Purpose is not just a vision—it is a framing of the change of value. It explains why a threat matters, why an opportunity is worth pursuing, and which aspects of service delivery must change in response. The system being commissioned is one possible instrument for delivering that change—but only after the purpose of the service is known, and only after the service is situated within its organisational context.

This is why a discovery phase must begin not by eliciting desires or listing actors, but by identifying **the purpose of the organisation**, **the purpose of the service**, and **how that service supports the broader mission**. Only then can discovery meaningfully proceed to identifying stakeholders, analysing needs, and defining requirements.

Stakeholder Identification

Before defining Business Requirements, a stakeholder map should be initiated. This map identifies all stakeholder groups across the full lifecycle of the system—business sponsors, operational staff, governance bodies, compliance teams, and support services. This early mapping is essential for ensuring that all perspectives are considered. For each stakeholder group, a representative subject matter expert (SME) should be nominated to articulate their specific needs and perspectives.

A comprehensive stakeholder map typically includes representatives from: the public; business service consumers (such as teachers, students, parents, school administrators, and principals); business service providers (such as sales teams, content creators, publishers, and product maintainers); customer support; operations (especially data quality roles); system maintainers; developers; testers and quality assurance specialists;

compliance and accreditation roles; project governance groups; and peak bodies or advocacy organisations such as unions or sector alliances. These last groups may represent the interests of service consumers or providers, without necessarily being users of the system themselves.

This initial stakeholder map often begins with high-level actors, but it must be iteratively refined as further information is gathered. The User Requirements phase will typically reveal new roles, processes, and service dependencies that were previously invisible. These discoveries must be used to expand and clarify the map. Stakeholder mapping is not a one-time exercise—it is a dynamic, evolving structure that ensures the discovery process remains grounded in actual organisational context, not abstract assumptions or isolated viewpoints.

Business Requirements

The first layer of structure begins with Business considerations—the 'why' behind the initiative. These are usually driven by opportunities to improve, threats to mitigate, or legislative or operational pressures to adapt. They establish the expected value of the system change, weighed in terms of cost and benefit.

Business requirements are not about features or workflows—they must clarify the purpose and benefit of the change. Typical business drivers include expanding service reach, improving service quality, reducing operational costs, increasing revenue, improving accountability, or extending the usable life of an existing capability. Each of these drivers must be clear enough to allow competing design options to be weighed against them.

Purpose

Despite being the natural starting point, a concise statement of the system's purpose is often omitted from business requirements. Yet without it, the project lacks a unifying anchor from which to evaluate trade-offs, validate scope, or determine whether the eventual solution remains on course.

The system purpose should summarise the intended operational outcome—not in terms of features, but in terms of what **value** the system is meant to deliver. One usable pattern is:

"The system exists to enable [target stakeholders] to [achieve X outcome] by [delivering Y capability or support]."

This purpose statement should be brief, specific, and enduring. It becomes the reference point for MVP decisions, acceptance criteria, scope validation, and overall governance. It ensures that business drivers are not merely noted -- but transformed into a clearly framed objective.

Audience

One of the most common analytical errors is the conflation of business and user requirements. Where business requirements describe *why* a system is being pursued, user requirements describe *how* people will engage with it. A requirement that blends the two often loses clarity and becomes impossible to validate. Each must stand on its own.

Principles

Once purpose is defined, the next foundation for decision-making is a shared set of principles. Principles exist to guide choices when the future is uncertain—without requiring constant deferral to governance bodies, escalation processes, or individual judgement calls.

Principles are not preferences or slogans. They are pre-agreed commitments about how decisions will be made when trade-offs, ambiguity, or constraints emerge during design and delivery.

For principles to be useful, they must be **concern-based** rather than **role-based**. That is, they should not reflect the worldview of a job title or department, but instead focus on real areas of recurring tension—places where decisions must be made, risks managed, or competing priorities resolved.

The right principles prevent conflict, reduce delay, and preserve design integrity—not by limiting creativity, but by ensuring decisions are made in alignment with shared values and long-term purpose.

Common domains of concern where principles should be established include:

1. **Legal Alignment**
   Defines how the system must meet the regulatory constraints of where it is consumed, hosted, developed.
2. **Purpose Alignment**
   Defines how to prioritise when system choices either reinforce or dilute the organisation’s overarching mission.
3. **Access & Security, Privacy Alignment**
   Guides how access, protection, and verification are handled—balancing openness with trust.
4. **Availability & Continuity Alignment**
   Determines what must persist across failure, change, or disaster, and how operational availability is prioritised.
5. **Scalability & Performance Alignment**
   Defines acceptable performance under load, and how capacity planning is approached.
   Example principle: “Systems should be sized for growth, but only optimised when demand justifies it.”

6. **Data, Information Value and Actionability Alignment**
   Frames what counts as accurate, complete, timely, and authoritative information—and how data responsibilities are shared.
7. **Change & Delivery Alignment**
   Articulates the approach to scope evolution, release cadence, and progressive delivery.
8. **Governance & Assurance**
   Clarifies who makes decisions under uncertainty, and how design intent is protected over time.
9. **Supportability & Maintainability**
   Addresses the ease with which the system can be diagnosed, updated, repaired, or extended.
10. **User Experience & Adoption**
    Places priority on trust, ease of use, and alignment with real-world behaviour, particularly when technical constraints conflict with user needs.
    Example principle: "If users avoid the system, it is not succeeding—regardless of feature completeness."
11. **Integration & Interoperability**
    Describes how the system will interact with others now and in future, including openness to external standards.
    Example principle: "Integration must favour stability and standards over vendor-specific shortcuts."
12. **Sustainability** *(optional in complex or public sector environments)*
    Provides direction when considering environmental, financial, and operational sustainability.
    Example principle: "Design choices should not impose avoidable cost or waste on future operators."

Each of these domains should contain just a few -- no more than five or six—clear, compact, and actionable principles. Overloaded, vague, or role-based principle sets lose meaning and cause friction rather than coherent clarity.

Most stakeholders will only need to reference the organisation's **purpose**, the **project's objectives**, and a small subset of **relevant principles** tied to their area of responsibility. This limited scope allows people to act with confidence while maintaining system delivery and outcome coherence.

Vision Statement

A key step often missed at this point is defining what *could* be. This stage is not just about recording problems or constraints—it is about actively engaging stakeholders in imagining what a better future might look like if the existing system did not exist. This

work is critical. Without it, the project risks being anchored too closely to the current system, process, or known limitations. Requirements gathered in that mode will inevitably reflect the past, not the future.

Importantly, this aspirational thinking is not the responsibility of the Business Analyst alone. While BAs are essential for eliciting, organising, and validating stakeholder input, the work of envisioning future-state systems requires input from system architects and strategic leads. These roles bring the structural insight and contextual framing needed to explore what is possible—within and across domains. In this mode, the analyst inquires into the vision of others, helping to make it specific and structured, but not originating it alone.

Discovery must make room for vision—not just verification. The task is to deliberately break free from current-state thinking and shape an aspirational yet feasible end state, which can then be reverse-engineered into specific, testable capabilities.

User Requirements

The next step involves identifying all potentially relevant stakeholder groups across the entire system and service lifecycle—not just those involved in delivery or business operations, but through to long-term support, governance, compliance, and decommissioning. For each group, a subject matter expert (SME) should be identified to represent their perspective. These SMEs are queried to capture the perspective on needs, constraints, and assumptions unique to each domain of responsibility. It cannot be stressed enough that their input is just that—input, not output. Each perspective must then be tested, validated, and synthesised into formalised capabilities and classified requirements that reflect not only what is needed, but by whom, and for what purpose.

User considerations follow, reflecting how individuals see themselves contributing to and benefiting from the change, using what functions. Their feedback helps define roles, usability expectations, and alignment with work practices.

Service Quality Expectations

The first and most important measure of quality is not whether a system is secure, fast, or maintainable—it is whether people actually use it.
No system delivers value unless it is adopted. And adoption does not happen just because a system works. It happens when people trust it, rely on it, and feel supported by it in the context of their real work.

This is where quality begins. Not with backend metrics or abstract standards, but with the lived experience of users. Unfortunately, this is also where most system projects go wrong—by starting at the bottom and working upward.

To correct this, quality requirements must be approached in three interdependent layers:

1. **System Use Qualities (ISO 25022)** – These define whether the system earns user trust and supports real-world success. They include effectiveness,

satisfaction, and context coverage, and they determine whether people continue to use the system over time.

2. **System Data Qualities (ISO 25012)** – These define whether the data presented to users is accurate, complete, current, and credible. Without good data, even a usable system will be avoided or misused.
3. **System Qualities (ISO 25010)** – These define how the system itself behaves: whether it is reliable, secure, maintainable, and efficient under real-world load. These qualities underpin data integrity and long-term operability.

These layers must be considered in order. A system that performs flawlessly but provides outdated or inconsistent data will not be trusted. A system with perfect data but confusing workflows will not be used. Adoption sits at the top of the quality pyramid—not the bottom.

This model provides both a practical checklist and a critical reframe: quality is not what the system *has*—it's what the system *enables*. Full descriptions of each ISO model are included in **Appendix F**.

Quality expectations shape the conditions under which systems must operate. They influence design, cost, risk, and ultimately user trust. Despite their importance, Quality Requirements are among the most misunderstood aspects of system design. They are often confused with Functional Requirements, conflated with Transitional Requirements, or reduced to vague aspirations with little practical value.

The development of Quality Requirements is not optional, nor should it be improvised. Like all requirements, they must be elicited by skilled Business Analysts. However, the SMEs involved in shaping them should come from specific stakeholder groups with operational and architectural responsibility. These include Enterprise Architects, Solution Architects, Operations, Support, Maintenance, Security, Development, QA, and Compliance and Accreditation specialists. While Solution or Enterprise Architects can offer proxy responses based on prior experience, they are not substitutes for directly engaging the source SMEs. If architects author Quality Requirements in isolation, the result is often aspirational or dense, lacking the pragmatism required for traceability and validation.

Moreover, Quality Requirements must not be invented anew for each system. Their omission is so consequential—and the scope of their relevance so wide—that their structure must be grounded in an established standard. The IIBA recommends ISO-25010 to elicit quality attributes of the system itself, ISO-25012 for the quality of system data, and ISO-25022 for the quality of system use. Starting from recognised models helps ensure completeness and consistency. Omitting this step introduces unnecessary risk and increases the likelihood of gaps or mismatches during delivery and assurance.

It is also crucial to resist the temptation to treat every system as a novel case. Nearly all systems inherit from a well-understood pyramid of quality expectations, where each tier

builds on the previous one, and the number of new or overriding requirements should diminish as the pyramid narrows.

At the base are qualities applicable to all systems: maintainability (diagnostics, configuration, error logging), operations (monitoring, control), security (audit trails, authorisation), customer support (account recovery, impersonation tools), and localisation (language switching, formatting standards).

Above this base are a very small number of country- or legal-jurisdiction–specific qualities such as non-negotiable legal privacy obligations, regulatory transparency, and accessibility compliance.

Above that are a tiny number of sector-specific qualities such as archival policies, reporting standards, and system-to-system interoperability.

Organisation-specific qualities follow, often involving integration patterns, branding expectations, or deployment constraints. These should be few and necessary.

At the top, and smallest in scope, are project-specific quality requirements. These should ideally only address quantifiable delivery-related parameters—such as intended volumes, performance thresholds, availability expectations, data classification, and RTO/RPO definitions.

The worst analytical outcome is to permit stakeholders to believe that their solution is so unique that it must invent its own quality model from scratch. This is almost never true. Robust systems emerge not from novelty, but from alignment with proven design practices.

An organisation that cannot differentiate between general, jurisdictional, industry, organisational, and project-specific Quality Requirements—treating them all as a single undifferentiated set—risks imposing unnecessary complexity, redundant checks, and mismatched obligations on every project. This not only increases project delivery risk, but leads to inflated cost, reduced agility, and systemic delivery fatigue. The result is a higher likelihood of failure—not because the solution was unachievable, but because the baseline expectations were poorly structured and inconsistently applied.

Capability Expectations

Human activity can be broadly understood as a cycle of purposeful, collective behaviour. People come together to form relationships and groups. They read, learn, imagine futures, express desires, and define improvements to their condition. These aspirations become objectives, which are translated into work. That work is planned, coordinated across time and place, and carried out through collaboration. Progress is assessed, outcomes are interpreted, and actions are adjusted accordingly. Most supporting systems—whether educational, economic, archival, or commercial—exist to enable,

regulate, or record these exchanges. They manage value, knowledge, and rights between individuals and institutions.

Within this context, Capabilities describe what the system must be able to do, independent of whether that is delivered manually, technically, or through a blend of both. They translate abstract requirements into actions the system must support. They reflect stakeholder needs, are framed by domain, and become the backbone of functional requirements.

From these three perspectives—Business, User, and Quality—emerges the first view of Capabilities. These perspectives shape not only what must be supported, but why it matters.

Domains bounding serve as the method to develop a cartography of the problem and solution space. They give structure to what might otherwise be overwhelming complexity. By mapping Entities, Roles, Capabilities, and State across domains like Social, Work, or System, we ground the solution in patterns already solved many times before. This reduces ambiguity, reveals omissions, and aligns the system to human experience, not just technical artefacts.

Capabilities are tightly aligned to Domains and their subdomains, and for clarity and traceability, they should be enumerated and validated within each Domain as per the non-exhaustive following list:

System Domain

- Manage Diagnostics and Monitoring
- Manage Configuration (e.g. settings, preferences, business rules)
- Manage Identity, Users, Authentication and Permissions
- Monitor Activity, Usage, and Health (e.g. logs, errors, usage tracking)
- Manage System Settings
- Manage Account Settings (ie similar to multi-tenancy, but more flexible)
- Manage Sessions of anonymous (public) and identified Users
- Manage Auditing of Operations during Sessions
- Manage User Settings/Preferences
- Support Operations and Maintenance (e.g. queue management, alerting, hotfix deployment)
- Manage Notifications (inside and outside of system)

Social Domain

- Manage People (e.g. their identity, name, date of birth, status, affiliation)

- Manage Relationships (e.g. person-to-person links such as parent-child or mentor-learner)
- Manage Groups (e.g. teams, classes, organisations, agencies)
- Manage Roles of Persons in Groups (e.g. teacher in a school, caregiver in a whānau context)

Artefact Domain

- Manage Artefacts of Different Media (e.g. documents, videos, forms, images)
- Manage State of Artefacts (e.g. draft, submitted, reviewed, approved, archived)
- Manage Roles Working on Artefacts (e.g. editors, contributors, reviewers, owners)

Aspirations Domain

- Manage Goals (e.g. learning outcomes, organisational targets)
- Manage Objectives and Milestones (e.g. timelines, checkpoints, programme benchmarks)
- Track Alignment of Work to Goals (e.g. contributions of actions to overarching objectives)

Work Domain

- Manage Tasks (e.g. assign, schedule, complete)
- Coordinate Contributions (e.g. sequence steps, define dependencies)
- Facilitate Collaboration (e.g. real-time editing, commenting, notifications)

Coordination Domain

- Manage Resource Scheduling (e.g. customers, providers, staff, locations, assets)
- Align Time and Place (e.g. calendars, sessions, term planning)
- Govern the Flow - of Participation, approval, conflict resolution (e.g. workflows, approvals, handovers)

Assessment Domain

- Assess Tasks against Aspirations (completions, milestones, quantities)

Use Case Specific Domain

- Apply Obligations and Rulesets (e.g. privacy rules, entitlements, exclusions)
- Maintain and Version Reference Data (e.g. country codes, subject codes)
- Execute Domain-Specific Workflows (e.g. application processing, moderation, certification)

Each Capability, controlled via system permissions, typically serves one or more stakeholder groups—Diagnostics may serve Maintenance Specialists, User Management

may serve Operations, and People Management may serve frontline Service Providers. Describing Capabilities in this structured and domain-bound way enables traceability from stakeholder need to system behaviour, and ensures the final solution aligns with both architectural integrity and operational need.

System Requirements

Once we know what has to be there to meet the expectations of Business and Users, and what level of Quality of Service is acceptable, we can begin describing what kind of system could support that. System Requirements are defined in two categories:

- Functional Requirements (FRs) describe what the system must do to deliver the required Capabilities.
- Non-Functional Requirements (NFRs) describe how well the system must do it, under real-world conditions.

Functional Requirements

Functional Requirements are the technical realisation of previously identified Capabilities. They describe specific functions the system must perform, typically expressed as actions on Entities within each Domain. A common pattern is CRUS: Create, Retrieve (single and group), Update, and State-change.

A critical risk in immature IT environments is that Business Analysts may document only those functions directly tied to high-visibility business objectives—overlooking essential functions in domains like System (diagnostics, configuration), Social (user identity and roles), or Artefact (submission state). This omission is often unintentional but can severely degrade system operability and maintainability.

To mitigate this, FRs should be mapped across each of the modelled domains, covering the full spectrum of necessary behaviour. This includes managing diagnostics, settings, identities, routes, people, relationships, groups, roles, artefacts, goals, tasks, assessments, and projections—ensuring that all foundational capabilities are technically delivered.

Functional Requirements should be reviewed for CRUD/CRUS coverage across all modelled entities (omissions here often stem from incomplete modelling, not system constraints).

Non-Functional Requirements

NFRs define the performance envelope in which functionality must operate. They reflect the Quality Expectations defined earlier and should be validated against recognised standards. The IIBA recommends ISO-25010 for system quality, ISO-25012 for data quality, and ISO-25022 for use quality. These standards help identify qualities such as maintainability, reliability, performance, scalability, auditability, accessibility, and supportability.

Any gap between stated NFRs and earlier Quality Expectations represents a potential design compromise or missed obligation. These gaps should be surfaced and addressed as part of system architecture or procurement review.

Transitional Requirements

The most commonly forgotten class of requirements is Transitional. These define what is needed to deliver the system—not what the system must do, but what the project must do. This includes resources, staff onboarding, development environments, training, documentation, licences, staging and release workflows, and everything required to move from concept to deployed solution.

Transitional requirements are not NFRs. NFRs are about the system. Transitional requirements are about delivery. Omitting them leads to project gaps, delayed onboarding, incomplete support plans, and weak alignment with business operations. Their absence is a common failure point, particularly in organisations lacking formal systems analysis or architecture training.

Importantly, not all required work can be automated by the system. Some aspects of the solution may need to be delivered through policy, process documentation, or manual intervention. If a required action, support step, or informational process is needed for delivery—but will not be handled by the automated system—it must still be captured as a Transitional Requirement. These elements are often internal or procedural and may fall outside the scope of vendor deliverables, but that does not diminish their importance.

A complete set of Transitional Requirements should include, but is not limited to:

- Resourcing plans and role allocation for discovery, design, delivery, and support
- Recruitment or reallocation of staff to support key delivery and operational roles
- Training plans for users, operations teams, and service support
- Communication and onboarding plans for stakeholder engagement
- Procurement of tools, environments, infrastructure, and third-party services
- Development of operational artefacts (e.g. forms, templates, manuals)
- Collection and reporting of project performance and assurance metrics
- Scheduling and ownership of governance milestones and reporting lines
- Risk mitigation activities outside the system (e.g. backup human processes)
- Delivery of policy and procedural documentation to cover any unautomated aspects

These requirements may may fall outside the scope of contracted system delivery, but they are essential for delivering an integrated and usable solution. Neglecting them places the entire system at risk of underperformance or outright failure.

Contextual Boundaries – Obligations, Policies, Principles, and Standards

The Quality and Transitional Requirements described above do not arise in a vacuum. They are often shaped—and in some cases dictated—by wider legal, sectoral, or organisational obligations.

A system never exists in isolation. An automated system exists within a people-based business system, which in turn exist within wider sector or professional systems, and ultimately within national or legal systems. Each layer imposes its own obligations, principles, and processes that must be respected in order for the system to be accepted prior to being integrated.

It is therefore imperative to actively seek out both external and internal obligations that exist beyond the project's control, as these must be met regardless of scope.

External obligations will include nationwide legal frameworks (such as privacy legislation), sector-specific obligations (such as public records, official information, or financial controls), and cross-agency mandates (such as cloud-first or open data policies).

Internal organisations will include policies and/or —preferably—principles. *Policies* constrain behaviour by encoding specific processes, attempting to embed a known past in the hope that the future can be conform to it. *Principles*, in contrast, provide navigational guidance—allowing groups to adapt their actions in unknown or emergent conditions.

Governance that insists on policy conformance to *process* inadvertently stagnate innovation and responsiveness. Governance that evaluates conformance to *principles* allows discovery and adaptation within shared agreed and accepted boundaries of choice. **This distinction is crucial to the health of an organisation or project.**

Standards are not laws or regulations, but cross-party agreements that enable interoperability and integration without prior contact or agreement. Adopting a standard is not mandatory, but ignoring it means others have no obligation to accommodate you. Standards offer the shared expectations that make integration predictable, cooperation easier, and replacement possible.

A useful analogy is urban development: just as a building must comply with neighbourhood zoning, city infrastructure, and national building codes, so too must a system integrate responsibly into its digital, organisational, and legislative landscape. These are not all constraints—they are also enablers. Compliance with standards allows reuse, sharing, security, and oversight. Ignoring them isolates the system, reduces its utility, and increases its operational risk and costs.

Outcome

The goal of this stage is not to deliver a solution, but to constrain the problem—to turn ambiguity into structure, and to ensure that what follows is grounded, not speculative. By the end of this phase, the organisation should have a structured understanding of

purpose, vision, stakeholders, quality expectations, capabilities, and requirement types—each traceable to the next.

This foundation ensures the system design phase begins from clarity, not confusion. A well-prepared discovery and analysis stage sets boundaries, aligns expectations, and defines success before a solution is proposed.

Activating the Axes – Structured Hypothesis-Driven Thinking

While the axes provide analytic structure, they must be activated by a deliberate mode of thinking.

One proven approach, intentionally irrespective of an IT lense, is The McKinsey Way, which applies a hypothesis-driven model to complex problem-solving. This method aligns closely with the needs of public sector system design.

Its steps are:

- Define the problem clearly, separating root causes from symptoms
- Break the problem down using MECE (Mutually Exclusive, Collectively Exhaustive) frameworks
- Develop testable hypotheses early and refine them through data
- Prioritise analysis where it will yield the most value
- Synthesise insights and communicate them clearly using structured storytelling

This process provides a disciplined lens through which to work with each axis:

- It treats each Domain as a hypothesis space to be structured and validated
- It encourages Information to be treated as evidence to test, not assumptions to confirm
- It supports Roles and Responsibilities being traced to specific needs, actions, and changes in information state over time

By applying structured reasoning early—before solutioning or delivery—the project gains alignment, traceability, and purpose. It is not the only approach, but it exemplifies the kind of thinking that allows complex system design to remain intelligible, testable, and explainable.

Design Stage

The work of developing clarity does not end with problem analysis. Once a future state is agreed and the domains and capabilities are broadly understood, the system must be designed. This work falls primarily to architects.

The Design phase is distinct from Discovery and Analysis. Where the earlier phases uncover what is needed, the Design phase defines how those needs are to be met. It is a

practice of future modelling—describing in structured form what a solution must become, before it is built.

A System Architecture Document (SAD) or equivalent blueprint is typically used to convey this model. It must be authored with different validation audiences in mind and must make interdependencies explicit. The key purpose is to ensure that each component or capability integrates with others, that constraints are respected, and that lifecycle considerations are not lost. The SAD provides enough structure to allow coordinated delivery, while allowing for incremental refinement.

Over-specification creates brittleness; under-specification leads to rework. Design is where balance is struck. It sets the stage for reliable, governable, and supportable implementation.

In architecture, this modelling typically takes the form of a System Architecture Document (SAD). A SAD is not a technical document for execution, but a blueprint structured for validation by different specialists. Each section is written with specific stakeholders in mind, ensuring that assumptions, constraints, and deliverables are visible and challengeable.

A complete SAD typically includes:

- **Technical Context** – the hosting, network, platform, and delivery assumptions
- **Information** – what data is being created, handled, or passed through, including classification levels
- **Constraints** – legal, environmental, cross-system, or institutional rules that shape the design
- **Domains** – the structural map of the system’s subject matter in terms of distinct domain-bound entities,
- **Capabilities** – what must be enabled or supported in each domain, even if not delivered as software
- **Functionality** – specific behaviours expected from the system, particularly those with business significance
- **States and Sequences** – key workflows and state transitions of information entities, anchoring process logic
- **Infrastructure and Dependencies** – what the system relies on and offers to others
- **Integration** – how the system supports other services (APIs) , beyond what it consumes
- **Security** – at rest, and in transit security as well as authentication and access control

- **Quality Assurance** – test strategy, validation points, and acceptance responsibility
- **Packaging and Deployment** – structure for quality assurance, static testing, packaging, deployment, provisioning, dynamic testing, data restoration – complex enough that it is best when referring to a *separate deployment SAD*
- **Support** – describes artefacts used to inform customer support
- **Operation** – describing artefacts used to inform system operation
- **Maintenance** – describing which artefacts support the maintenance of the stem.

This level of design requires more than listing features. It demands a structured blueprint, just as in physical construction. An apt analogy is the planning of a house: it begins with determining use case objectives (family size, room numbers, parking space, storage, kitchen and bedroom layout), then identifying constraints (zoning regulations, council consents, trade standards and dependencies, neighbourhood character, and legal covenants), and then sequencing delivery (earthworks, drainage, foundations, pad, framing, roof, plumbing, electrical, lining, cladding) looking to shorten deliver phases by scheduling trades in parallel *where possible and practical*, as opposed to in sequence.

A well-prepared building plan includes separate sections for each trade to validate and execute. The foundation team, plumbing subcontractors, electrical inspectors, and builders each rely on their own views of the same model. In doing so, the blueprint enables confident and interdependent delivery. Each plan permits a different stakeholder group to proceed confidently with their area of responsibility, knowing that the architect has verified location, and delivery coordinator (or foreman) have validated scheduling, such that their work will not interfere with others, nor be obstructed by others. Upon completion, it will also mesh seamlessly with the contributions of adjacent teams. This is what ensures, for example, that a plumbing riser is not placed in front of a window frame, or that electrical conduits don’t obstruct insulation flow (decrease quality) or breach firewalls (impact security).

Crucially, the plan must also acknowledge what infrastructure is already available—stormwater outlets, septic system access, mains power, cable runs—so that integration is intentional, not left to chance.

So too in systems. The same analysis in IT uncovers dependency on external services of secure and data storage, identity validation, external notification, system monitoring, integration hubs, operational analysis and reporting, data warehousing, archiving.

In system design, a Solution Architecture Description (SAD) must serve as a multi-view blueprint. Each section is authored for validation by a different delivery or assurance stakeholder—security advisors, platform engineers, integration analysts, and test leads. The goal is not to please each group, but to expose assumptions, encourage alignment, and enable rigorous delivery. Each plan permits a different stakeholder group to proceed confidently with their area of responsibility, knowing that both the architect and delivery

coordinator (or foreman) have validated that their work will not interfere with others, nor be obstructed by others. Upon completion, it will also mesh seamlessly with the contributions of adjacent teams. This is what ensures, for example, that a plumbing riser is not placed in front of a window frame, or that electrical conduits don't obstruct insulation flow or breach firewalls.

Omitting the less visible or non-functional aspects—like capability in non-technical domains, or information state transitions—yields brittle and misaligned systems, no less than a house without drainage or insulation.

While the three axes provide structure, they require a method to activate them. One proven approach is The McKinsey Way, which applies a hypothesis-driven model to complex problem-solving. Its steps are:

- Define the problem clearly, separating root causes from symptoms
- Break the problem down using MECE (Mutually Exclusive, Collectively Exhaustive) frameworks
- Develop testable hypotheses early and refine them through data
- Prioritise analysis where it will yield the most value
- Synthesize insights and communicate them clearly using structured storytelling

This method enables disciplined thought across all three axes. By generating and testing structured assumptions, the project gains clarity, traceability, and alignment—long before development begins.

Recommendation

To recover clarity in IT project definition, the following actions are advised:

- Establish a common analytical framework across all project participants
- Require clear articulation of the problem before solution scoping
- Adopt or adapt established models such as BABOK, MECE, and RASCII
- Recognise and structure requirements into appropriate categories, including transitional
- Use domain and temporal decomposition to understand the system fully before planning delivery
- Train analysts and project leads in structured critical thinking, not just stakeholder management

Conclusion

IT is **information first, not technology first**. Projects succeed or fail not on frameworks or code, but on how well a **purpose** has been defined, then how well they define,

manage, and transform **information**—its structure, its state, its access to it and its use – towards the purpose. Without this grounding in first purpose, then the “I” of IT, no tool, platform, or delivery model can rescue a poorly aimed or conceived solution.

Data alone is not enough. Data is storage—potentially useful, but inert. It may be noise, narrative, or records, but until it has structure, context, purpose, and a state-change it supports, it is not information. A system built on unqualified or unactionable data is not a solution or asset. It’s a liability.

To plan well, teams must first be clear on **BUQCST**—Business needs, User expectations, Quality requirements, Capability definitions, System behaviours, and Transitional delivery activities. These elements collectively define the service, the solution, and the project. Missing any one leads to drift, confusion, and eventual failure.

Second, projects must not begin from the current state. The current state is the problem. Instead, they must start from *postcards from the future*—a clear vision of how the world should look. From there, they can work backward. This is the path to meaningful transformation. Anything else risks reinforcing legacy dysfunctions.

Third, the problem must be broken into recognisable, solvable areas. These are the domains—the cartography of human and system experience. What cannot be mapped should be minimal and made explicit. Projects that treat every part as novel are building on quicksand.

Fourth, delivery on an existing platform does not inherently simplify anything. Platforms do not solve problems—they constrain how problems may be approached. Without clarity on the problem, platform selection simply adds friction.

Fifth, systems must be both envisioned and defined before they are delivered. Skipping that work is not strategic minimalism—it is **Missing Valuable Planning (MVP)**. It results in partial solutions, misaligned delivery, and chronic rework. Planning is not overhead. It is foresight made actionable.

In summary: IT projects do not fail from lack of effort or goodwill. They fail when information is unstructured, when problems are undefined, when domains are unrecognised, and when planning is skipped. The solution is not more process—it is better thinking.

This document lays out the structure and tools to do just that. What remains is to use them—with clarity, confidence, and discipline.

Appendices

Appendix A - Document Information

Authors & Collaborators

- Sky Sigal, Solution Architect

Versions

0.1 Initial Draft

0.2 Additions

0.3 Corrections

0.4 Expansion

0.5 Softening

1.0 Release

1.1 Addition of Project v Purpose

1.2 Articulation of Purpose and Value

1.3 Added Principles as a deliverable

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, Technical Manager	
Margaret-Anne Barnett, Strategic ICT Business Advisor	
Diane Simpson, Product Owner	

Duncan Watson, Enterprise Architect	
Gareth Philpott, Solution Architect	
Justin Nehemia, Infrastructure Architect	
Arthur Laclaventine, Senior Network Architect	
Carl Klitscher, Solution Architect	
Vincent Wierdsma, Lead Developer	

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.

BUST : Business, Users, System, Transitional [requirements]. See BUQCSTT.

BUQCSTT : Business [Objectives], [System] User, Qualities [of Service], Capabilities, System [FR & NFR], Transitional, Technical [requirements].

DDD : Domain Driven Design – a robust approach to system design.

FR : Functional Requirements – one half of the logical Solution Requirements.

GUI: Graphical User Interface. A form of UI.

MVP : an aspired Minimum Viable Product – unfortunately too often turns out to be Missing Viable Planning.

NFR : Non-Functional Requirements – one half of the logical Solution Requirements.

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.

SAD : Solution Architecture Description.

SMART : Specific/Singular, Measurable, Achievable, Relevant, Time/Resource bound

SME : Subject Matter Expert

UI : User Interface. Contrast with API.

Terms

Refer to the project's Glossary.

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

Capability : a specific, bound, area of functionality to manage a domain.

Domain : an specific, bound, area of knowledge.

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).

Appendix B – Recommended Reading and References

- IIBA Business Analysis Body of Knowledge (BABOK)
- McKinsey: The McKinsey Way, and related publications on problem structuring
- Ministry of Health NZ: National Oracle Solution Independent Reviews
- Auditor-General NZ: Lessons from public sector project reviews
- Peter Wachira: LinkedIn posts on structured problem solving
- MECE Framework: Mutually Exclusive, Collectively Exhaustive
- Double Diamond: Discover, Define, Develop, Deliver
- RASCII: Responsible, Accountable, Supporting, Consulted, Informed, Ignored

Appendix C – Local Case Studies (Suggested for Expansion)

- Novopay

- IRD Business Transformation
- Various Smart City and Regional Integrated Ticketing Projects

Appendix D – SMART Requirements

Requirements, whether developed for Waterfall or Agile work items have the following qualities:

- **Specific:** Clearly define the problem and its boundaries.
- **Measurable:** Identify and establish quantifiable criteria to assess the success of the solution.
- **Actionable (Attainable):** Ensure that the solution is realistic and within the scope of the problem and should point to the action that needs to be done to solve the problem.
- **Relevant:** Ensure that the solution aligns with the goals and objectives of the organization or situation.
- **Time-bound:** Set a timeline for implementing and evaluating the solution.

Appendix E – User Story qualities

- **Sponsor**: as a <role>
- **Change**: I desire <the following change>
- **Outcome**: to improve <outcome>
- **Acceptance Criteria**: the tests to qualify results.

Appendix F – System, System Data & System Use Qualities

ISO 25022 – System Use Qualities
These define how well the system supports trust and effectiveness from a human perspective—what users experience directly.

- **Effectiveness** – The degree to which users can achieve intended goals accurately and completely.
- **Efficiency** – How much effort (e.g. time, clicks, mental load) is required to complete a task.
- **Satisfaction** – The user's subjective comfort, acceptability, and confidence in the system.
- **Freedom from Risk** – The extent to which use of the system avoids harm to the user (including data, safety, and legal risks).

- **Context Coverage** – Whether the system works well across different use environments, user types, and operational conditions.
- **Trust** – The user's belief that the system behaves as expected and protects their interests, including privacy and fairness.

The above qualities in turn depend on the quality of the system's *data* as per below:

ISO 25012 – System Data Qualities
These define the essential properties of the data the system uses—how complete, reliable, and usable it is.

- **Accuracy** – Data correctly represents the real-world values it is intended to model.
- **Completeness** – All necessary data is present and sufficiently populated.
- **Consistency** – Data does not contradict itself across the system.
- **Credibility** – Data is trustworthy and comes from verifiable sources.
- **Currentness** – Data is up to date and reflects the real-world state.
- **Accessibility** – Data can be retrieved when needed, by the right people or systems.
- **Compliance** – Data adheres to legal, regulatory, or organisational standards.
- **Confidentiality** – Data is protected against unauthorised access.
- **Efficiency** – Data can be queried, stored, and transferred without undue resource usage.
- **Precision** – Data is recorded at the correct level of detail.
- **Traceability** – The origin, changes, and usage of data can be tracked over time.
- **Understandability** – Data meaning is clear and well-structured for its intended users.

These qualities in turn depend on the underlying behaviour of the system itself, as per below.

ISO 25010 – System Qualities
These define how well the system is built—its reliability, security, maintainability, and overall operational robustness (note that each of the following categories has sub-categories omitted here for brevity's sake).

- **Functional Suitability** – The system does what it is meant to do, covering all required behaviours.
- **Performance Efficiency** – The system performs responsively under expected load and resource constraints.

- **Compatibility** – The system can interoperate or co-exist with other systems.
- **Usability** – The system is understandable, learnable, and easy to operate.
- **Reliability** – The system runs without failure for its intended duration and usage.
- **Security** – The system resists unauthorised access and misuse.
    - **Confidentiality** – Information is only accessible by those with permission.
    - **Integrity** – Information cannot be tampered with without detection.
    - **Availability** – The system is up and functioning when needed.
    - **Non-repudiation** – Actions cannot be denied by those who performed them.
    - **Accountability** – Activities can be traced to responsible actors.
- **Maintainability** – The system can be updated, corrected, or improved efficiently.
- **Portability** – The system can be deployed or adapted to different environments with minimal effort.