Requirements Management and Project Lifecycle [1]

Pascal Bohulu Mabelo

Abstract

It is widely accepted that "Project Management is the application of knowledge, skills, [attitudes,] tools, and techniques to project activities in order to meet project requirements" (PMBoK, 2013). The 'application' is not merely in doing project works, but aims to "meet project requirements." This simple definition suggests that no project delivery would make sense until requirements are defined, documented, and agreed upon—and effectively implemented, especially in megaprojects.

A recurring challenge is that many experienced project managers caution against mentioning the term 'requirements' to clients, fearing it might complicate the project from the outset. This concern is compounded by the fact that project lifecycle methodologies typically expect the client to furnish a project brief in the form of 'Owner's Requirements Specifications' or 'Project Requirements.' However, some organisations may lack the capability or willingness to define and take ownership of such requirements. It soon turns into, "Tell us what requirements must be—for you are the experts!" The danger with this scenario is quite pernicious. Not only could the owner be handing out a blank cheque to the appointed team, but they are also being set up to deliver something the client will be quick to denounce and reject, saying, "This is not what we asked for; you have done your own thing!"

In the rare instances where clients provide 'copy-and-paste' requirements—adapted from previous projects—these may introduce confusion rather than clarity. Such inherited requirements frequently fail to align with the project's unique context, diverging unpredictably at the client's discretion instead of honing in on a coherent and elaborate baseline as the project evolves—from generic, high-level to detailed requirements. Still, successful projects must meet requirements and client's needs!

Necessity of Project Requirements

Many project managers tend to approach delivery linearly: (1) collect requirements, (2) define the scope, and (3) develop the Work Breakdown Structure (WBS), assuming the subsequent steps will naturally follow. However, understanding the essence of 'requirements,' including their sources, timing, and relevance (i.e., the *why, what, who, where, when,* and *how* of project requirements), is far from straightforward and calls for a more nuanced approach. Any missteps in identifying and managing requirements will trigger a ripple effect and compromise the system development process.

Systems Engineering (SE) is "[…] an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining Acquirer needs and required functionality early in the development cycle, documenting requirements, and then proceeding […]" (INCOSE, 2015) [*Underlining added for emphasis*]. In short, project management relies on *elicited* and documented requirements for the "*realisation of successful systems*." Indeed, as the PMBoK equally maintains:

> "Project Management is the application of knowledge, skills, [attitudes,] tools, and techniques to project activities in order to meet project requirements." (PMBoK, 2013) [*Underlining added*]

[1] How to cite this paper: Mabelo, P. B. (2025). Requirements Management and Project Lifecycle; featured paper, *PM World Journal*, Vol. XIV, Issue II, February.

Therefore, a project is "successful" only if the stakeholders' requirements are met and attained within the stated cost, time, and quality constraints—it is about meeting project requirements! Robert Buttrick eloquently captures the significance of requirements in his definition of a project:

> "An endeavour in which human, material and financial resources are organised, in a novel way, to undertake a unique scope of work, of a given specification [i.e., set of requirements], within the constraints of cost and time, so as to achieve a beneficial change defined by quantitative and qualitative objectives." (Buttrick, 2003) [*Underlining added for emphasis*]

Consequently, if a team fails to manage requirements, their project will fail to meet requirements—it will flounder and fail. The author has compared the causes of project failure per the Chaos Report. Not only do these *causes* remain almost the same during the 1994 to 2009 period, but at least half of those reasons (in numbers and percentages) pertain to project requirements as follows:

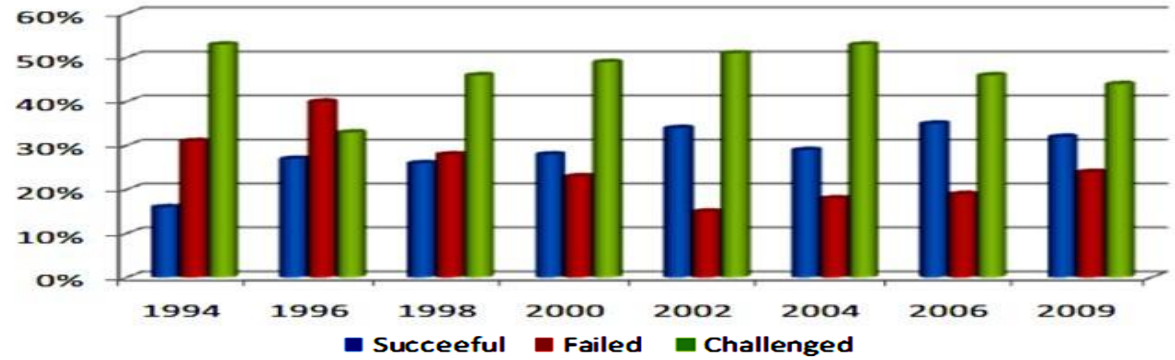


Figure 01 – Comparison of Project Outcomes from 1994 to 2009 (per Chaos Report)

Excluding the 1994 data of *successful* projects (16%), the tally gives a standard deviation $\sigma \approx 3.5\%$, suggesting no significant improvement was recorded during that period. The apparent reductions in failure rate from 1998 to 2002 could be attributed to a heightened focus on project delivery due to the 1997/1998 financial crisis. Indeed, there was a surge in the rate of 'challenged' projects over the same period (Mabelo, 2016). Interestingly, this *sluggishness* occurred during the 1990 to 2010 era when PM Software, Earned Value, process-based methodologies, global certifications, and Agile practices were tenaciously promoted. This gives reasons to believe that requirements issues (namely, lack of user input, incomplete requirements, changing requirements, unrealistic expectations, and unclear objectives) have been, and still are, the root cause of this chronic project failure syndrome.

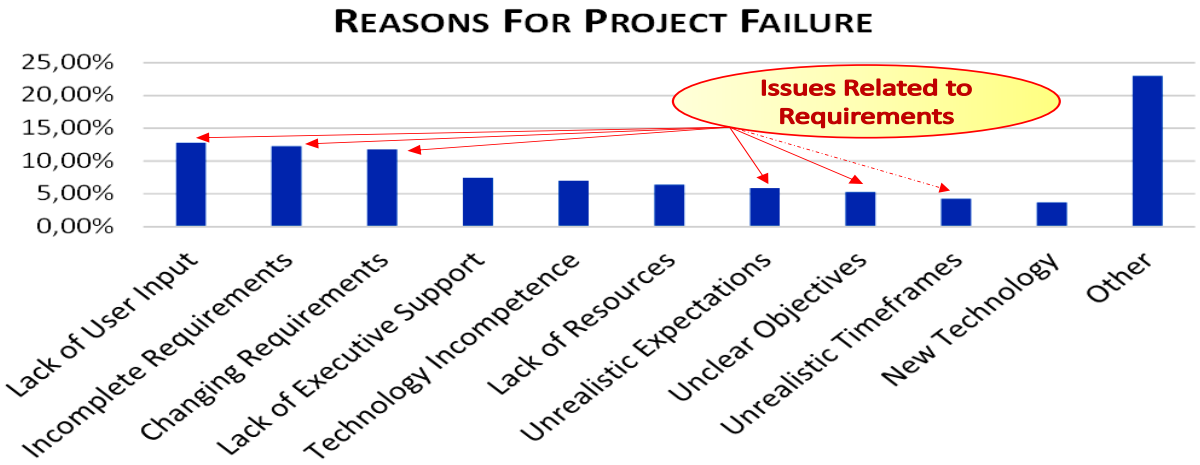


Figure 02 – Reasons for Project Failure (Chaos Report, 1994 and beyond)

Requirements are the single *thread* running through a project from conception to building, testing, and operations; the entire project scope should be constructed to meet requirements effectively. Thus, the project lifecycle methodology is about managing business and operational requirements throughout the project lifecycle, from inception to operations—one says, "*from womb to tomb*"!

Contrary to popular belief, projects do not fail due to cost and schedule overruns. Instead, since time (schedule) and money (cost) are spent on meeting requirements, those overruns indicate a "*failure to meet requirements*" on the project; in due course, this *failure* will translate into overruns (as symptoms) while ultimately leading to project failure. One ought to blame it on requirements, not on overruns for which project managers are often crucified. It is no wonder Kendall maintains:

> "By applying the practices and methodology outlined in the PMBoK, significant waste can be reduced. For example, most organizations tell us that they suffer from poor requirements definition. The result is that there is a lot of rework by project teams." (Kendall & Rollins, 2003)

Effective requirements management is key to project success; it lends potency to project delivery. To prove effective, project delivery methodologies for Large Infrastructure Projects (LIPs) should adequately reflect the "*interdependency*" of Project Management (on scope delivery) and Systems Engineering (on requirements) in managing requirements on the projects (Forsberg et al, 2003). Furthermore, the PMI's 'Requirements Management–A Practice Guide' stipulates, "Completion of the product scope is measured against the product requirements" (PMI, 2016); consequently, requirements should derive from "*the needs for the product*" and, thus, from business objectives.

> "When formal root-cause analysis is not performed, business people may 'jump to solutions' to solve their perceived problems. The result is often adding new capabilities that address only part of a situation and may also be more expensive than needed." (BA Guide, 2017)

Therefore, 'requirements' entail identifying the customer needs, conducting a market analysis to locate what is already available, and defining design goals. "Many of the difficulties encountered in design may be traced to poorly stated goals [i.e., poor requirements] or goals that were hastily written and resulted in confusion [i.e., wrong requirements] or too much flexibility [or ambiguity]" (Haik and Shahin, 2011)—From flawed requirements to poor design to bad products to failure.

Project management is all about meeting project requirements; eliciting, documenting, managing, and converting requirements into technical solutions to be "implemented" to address a problem. For that reason, many authors have made it a point to highlight the significance of requirements:

(i) "A factor present in every successful project and absent in every unsuccessful project is sufficient attention to requirements." (Robertson & Robertson, 2005)
(ii) "The cornerstone of SE [in large and complex projects] is the careful identification and analysis of stakeholder requirements." (Locatelli et al, 2014)
(iii) "A clear statement of requirements is arguably the most essential task in achieving [project] success." (BABoK, 2015)

The *Red Valley Bridge* (an alias) vividly illustrates these facts. The project faced massive delays and cost overruns due to poor requirements management. Inadequate attention to geotechnical studies led to construction challenges, and failure to address community and environmental concerns caused frequent protests and legal challenges. Vague requirement definitions, like "*minimise earthquake impact*," resulted in costly redesigns. No wonder, the project initially estimated at $ 1.4 billion and 5 years, took 9 years and $ 6.2 billion—highlighting the consequences of mishandling requirements.

Project Requirements Process

Nothing new can be produced, serviced, or maintained without detailed requirements or some set of standards. Therefore, each aspect of the desired item must be clearly defined (Tapke et al, 2001).

> "As the global environment becomes more complex, organizations [e.g., projects] that take a proactive approach to requirements activities will improve their competitive advantage by reducing waste [or overruns] and delivering projects that provide business value." (PMI, 2016)

Unfortunately, it was reported that "inaccurate requirements gathering [alone] was identified by 37% of organisations as a primary cause of project failure" (PMI, 2014). Further, a survey of the project delivery community indicated that "Only 49% of respondents have the resources in place to perform requirements management properly" (PMI, 2016)—while 51% destroy project value in the process.

To grasp the concept of requirements, one may need to consider this example of designing a chair:

> "Most of the time the customer provides a generic statement of need, and it is up to the engineer to identify the specific needs of the customer. For example, we are required to design a chair that can be used by a child. Clearly, all of us know how to sit on a chair, so in that perspective, we know how a chair works. A chair is used for sitting. Unfortunately, this description does not say how the chair is made. What material is used? Is the chair flexible or rigid? Does the chair rotate or is it fixed? [Is it collapsible?] What does it mean that the chair is to be used by a child? Is safety the biggest concern? How much will the chair cost? How old is the child? And so on." (Haik & Shahin, 2011) [*Underlining added for emphasis*]

Until such details (as underlined in this extract) are adequately elicited, there are no requirements; as a result, there is little chance of success, no matter how committed the project team might be. Therefore, unless one understands what constitutes "project requirements" and learns to appreciate their utility, there is little to no prospect that the project will satisfy the stakeholders' needs.

INCOSE and systems engineers rely on the definition of "requirements" provided by the IEEE Standard Glossary of Software Engineering Terminology (viz, IEEE Standard 610.12) as follows:

(1) A condition or capability needed by a user [or any relevant stakeholders, if the project owner or representative ratifies such a condition] to solve a problem or achieve an objective;
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; or
(3) A documented representation [in any due form] of a condition or capability as in (1) or (2).

Following Systems Engineering practices, "Project Requirements" could be described as follows:

(a) A requirement is a collection of capabilities originating from owner/users and other relevant stakeholders that all must be met by the solution (i.e., 'system' and 'delivery process') to solve the problem and achieve the objectives;
(b) Project Requirements define the bridge between the intentions and results of projects, between the 'Current State' and the envisioned 'Future State';
(c) A specification is an explicit set of requirements to be satisfied by the project.

As the above definitions imply, Project Requirements should involve two important components:

(i) 'System/Product' Requirements ("*What-to-Build*")
(ii) 'Process' Requirements ("*How-to-Build*")

Due to the increasing complexity of infrastructure systems, for instance, the design process often reflects a portion of the same; hence, the need to devise a viable process (i.e., "*How-to-Build*"). Thus, 'requirements' are not only about what the envisioned system shall do (i.e., "*What-to-Build*"). Both specifications (i.e., "*design-to*" and "*build-to*" requirements) are crucial to project delivery. Building a 45-meter-deck bridge on shallow waters is not the same as launching an identical bridge over a 180-meter canyon—one will need to devise a more advanced construction/works method.

Besides, since "*systems come in triplets*" (Scott, 2012) requirements ought to be elicited not only about (1) the Solution-System (i.e., the system intended to address the undesirable situation) but also from (2) the Realisation-System (i.e., the system established to create or develop the solution) and (3) the Context-System (i.e., the broader system where the intended solution will be deployed).

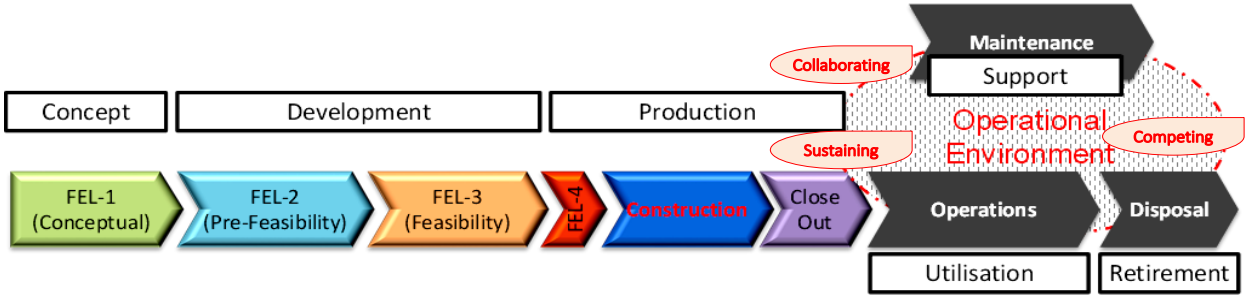


Figure 03 – Project Lifecycle Showing the Operational Environment and Related Aspects

Further, following Figure 03 above, the design of the Solution-System should consider and satisfy any 'requirements' arising from Competing-, Collaborating-, and Sustaining-Systems that might exist in the operational environment. Indeed, once deployed, the Solution-System must function despite or with the support of current or planned systems in its operational sphere. Such systems may compete with (i.e., vie for resources or space), collaborate with (i.e., share operations), or sustain (i.e., provide resources to support) the new system (e.g., mining shaft, powerplant, school). It must be noted that of the many requirements arising from the above realms, some would be "operative" (i.e., applicable permanently) and others "transitional" (i.e., ad interim, in transition). The two tables (in Annexure 1) provide the Requirement Classifications per INCOSE and SEBoK.

Focusing on the requirements of the (core) solution alone would always prove counterproductive. While the system may still work as a standalone, it will certainly fail to function adequately once deployed in its intended environment. This is because of the conflicts, dissonances, and other asymmetries arising from the missing, flawed, inappropriate, and counterintuitive interfaces with some adjacent systems discussed above. Imagine a system incapable of interfacing with adjacent systems intended to sustain it—its "as deployed" performance will surely suffer, leading to failure!

Consistent with Systems Thinking, system requirements are *interconnected* and should be treated accordingly. For example, modifying a single requirement—such as changing a crane's functional capability to lift four 20-foot containers instead of two—in the Solution-System domain can trigger adjustments in other domains. These may include increasing the motor's power supply, redesigning clamps to secure the containers during lifting, and updating crane driver training in the Realisation-Systems (i.e., transition) and Context-Systems (i.e., sustaining). Overlooking such interdependencies can undermine the new system's overall performance—once deployed in its intended environment.

If getting requirements right would be this *arduous* to project teams that at least attempt to identify and manage them adequately, what could the fate be of those oblivious to the process altogether?

Requirements Maturity and Project Failure

It has been said (see Figure 02) that at least half of the reasons for project failure *directly* pertain to requirements—failure to manage requirements leads to project failure. Thus, to enhance the success of projects, it is vital to understand the importance of successfully, (i) identifying, and (ii) managing project requirements. Systems Engineering (SE) concepts improve project success through better requirements handling—reducing *scope creep* or nasty surprises (e.g., delivering a wrong solution).

> "The basic idea behind capability models is the assumption that a high quality of processes [e.g., Requirements Lifecycle] will be reflected in a high quality of the results or artefacts of these processes […] It is the common belief of almost all industries that if the development processes are of high quality, the final products will inherit a fair portion of that quality." (Hood, 2008)

An IAG Consulting Report notes, "No matter the delivery method employed, the project success rate is strongly [and positively] correlated to the level of Requirement Maturity" (IAG Consulting, 2009). Indeed, Requirements Management Maturity [i.e., from levels 1 to 4, as per (Hood, 2008)] reflects the capability of an entity to define and manage requirements. The four levels indicate the maturity of processes, practices, technology, tools and techniques, deliverables/results, organisation, and staff competency. A steep *premium* is often paid for poor-quality requirements in Large Infrastructure Projects (LIPs); low-quality requirements (in either process or outcomes) cause project overruns.

Further, in discussing the 'Reasons Projects Fail', Keith Ellis of IAG Consulting argues as follows:

> *We have been tracking IT development projects over the last twenty years […] Failure rate is not going down; in fact, it is going up […] People are playing this blame game; blaming the scope, the business case, poor estimates, or lack of controls […] Failure is a result of lack of momentum (Momentum=Critical mass of understanding*Velocity) […] Out of 100 IT projects 94 will start all over again, at least once […] that 75% of projects spent 6 millions of dollars on projects estimated to cost 3 million dollars […] Failure comes from the inability to gather requirements well—70% of projects failed due to poor requirements*. (IAG Consulting, 2009)

If the requirements delivered during the Requirements Development stage are not of high quality, the solutions derived from such "conditions" have the risk of not achieving the intended objectives —It is cheaper to fix system errors at the requirements level by improving requirements handling! Although Figure 04 (see below) originated from the IT sector, it is relevant in Large Infrastructure Projects (LIPs). It indicates that the cost of fixing a "defect" is far less at the Requirements Stage (early phases) than at subsequent Testing and Maintenance Stages. Further, INCOSE (2015) notes that the small amount (8%) spent in the Requirements Phase can influence 75% of lifecycle costs.

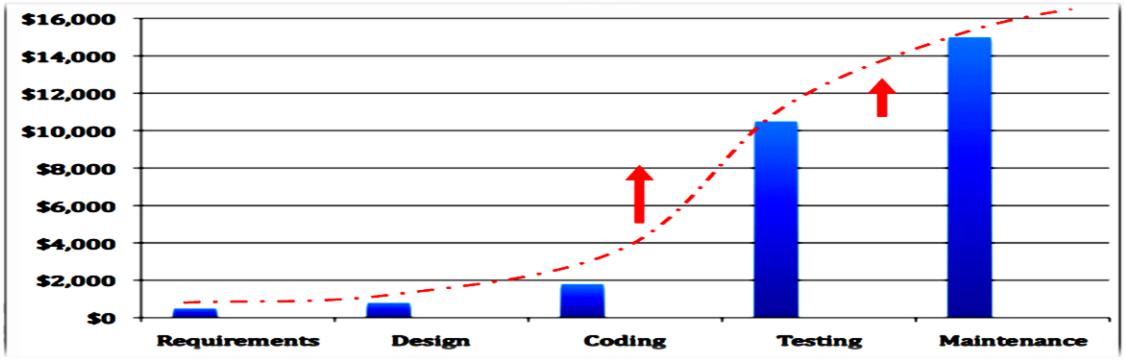


Figure 04 – Cost to Fix Defect in Various Stages (Adapted: INCOSE, 2015)

This observation may shed light on the all-too-common scenario where projects advance smoothly until they reach the infamous '*95% complete*' mark only to stall indefinitely, with further progress proving elusive—an endless struggle reminiscent of Sisyphus' curse. In Greek mythology, Sisyphus was *doomed*—condemned to roll a boulder up a hill, only for it to roll back down each time he neared the summit. Similarly, the *seeds* of failure in projects are often sown during the Requirements Phase, with these missteps resurfacing during testing or late in construction (e.g., due to systems integration problems). Under client pressure, project teams may "*tweak and twist*" the system to make it work; yet, worse woes such as operability failures and high-cost maintenance would manifest downstream, due to those upstream flaws. Oehmen (2012) argues, "Upstream activities must be held responsible for issues they cause in downstream activities"—Thus, addressing requirements flaws is imperative!

Construction and operations problems might need a recourse upstream, going back to planning and development for 'requirements' to be addressed in an *integrated* manner. Attempting to quick-fix a seemingly isolated faulty element of the 'system' may negatively impact other elements, but not necessarily in the proximity of time or space—usually, making the problem worse (Senge, 2006). This is a "*principle*" every business or systems analyst should consistently stress to the project team.

Moreover, Figure 05 (Mabelo, 2016; Mabelo, 2021; Mabelo, 2022) suggests that project success should not be measured narrowly in terms of cost, schedule, and quality (or the Iron Triangle). Projects are meant to "improve" operations (e.g., better products or services, lower costs, superior returns, faster delivery, higher customer satisfaction). Thus, goals or objectives in that direction should be the *yardstick* of project success. The Project Failure Grid indicates that both dimensions must prove satisfactory for a project to be successful. Experience and research have confirmed that 'wrong' requirements (with high quality, but low relevancy) engender Business Failure (i.e., white elephants) whereas 'poor' requirements (with low quality, but high relevancy) often lead to Delivery Failure (i.e., challenged projects)—both shall lead to Catastrophe (i.e., aborted projects).

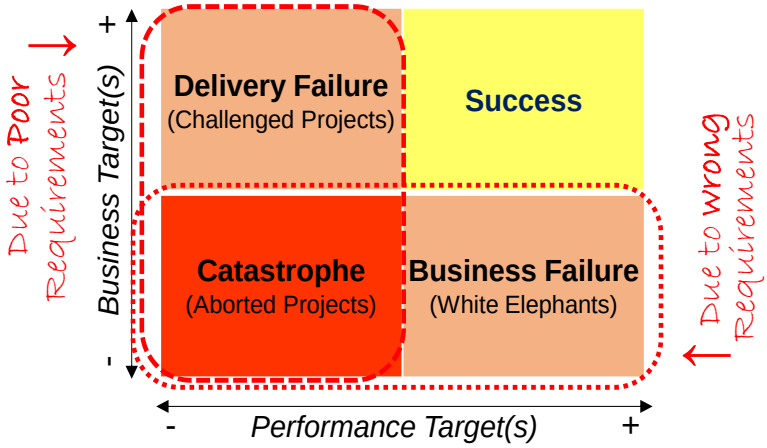


Figure 05 – Project Failure Grid and Requirements

It must be noted that this framework applies to any project provided that its definitions are upheld. Business Failures are finished on time, on budget, and to quality but will not yield business returns; Delivery Failures might have struggled to complete but are now likely to produce business returns. From this viewpoint, success can only be expected when no 'wrong' or 'poor' requirements prevail. In effect, 'wrong' requirements arise, for example, when a project team is tasked with developing a nuclear plant but defaults to using the specifications from a solar power plant. Conversely, 'poor' requirements may be incomplete, inconsistent, or unclear, even if intended for the nuclear plant.

The preceding discussions reiterate that requirements are the primary reason for project failure. Figures 04 and 05 eloquently capture this point; the *seed* of success or failure is in requirements. The question remains, "*What should be done to address the risk of 'poor' or 'wrong' requirements causing infrastructure projects (e.g., power plants, mining shafts, hospitals, shopping malls) to fail*?"

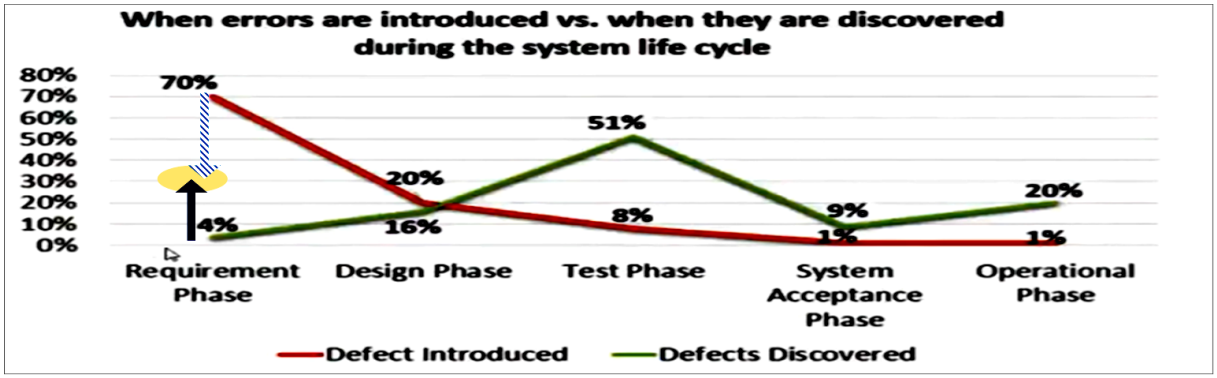


Figure 06 – Requirements and Project Failure (Source: IBM Business Research, 2017)

It was mentioned that rectifying 'wrong' or 'poor' requirements is *cheaper* in the early phases of the project lifecycle (e.g., in the Conceptual Phase). Figure 06 (above) further reveals that not only most requirement errors are introduced (purposely or unwittingly, it matters not) but they are often not discovered (and, thus, not rectified) at that stage. Therefore, an effective requirement process should focus on and target (i) preventing the introduction of 'wrong' and 'poor' requirements and, equally, (ii) enhancing the capacity to detect any flawed requirements at that stage and afterwards. As an industry, we need to "*flatten and lower*" the red curve and "*flatten and raise*" the green one. Today's insistence on *detailed* requirements from the get-go is counterintuitive; it entails guesswork and makes room for 'low-relevancy' and 'low-quality' aspects that creep in to impair requirements. Annexures 2 to 4 discuss examples of project requirements, their prevailing flaws, and the remedies.

To enhance the delivery of projects (large and complex infrastructure projects, LIPs, in particular), efforts must be invested and directed to effectively address the challenges of relying on wrong or poor requirements as input to the system development lifecycle process. Organisations involved in project delivery shall raise their requirement maturity towards the higher levels of the Hood Model. Lower requirements maturity levels entail higher overruns and maintenance costs. In addition, these are *exacerbated* by the rising complexity of LIPs—addressing maturity challenges is of the essence. Requirements process improvements (e.g., adopting V&V practices and Traceability Matrices) and comprehensive, teamwide training (e.g., SE workshops, elicitation techniques) could be a good start.

Moreover, Artificial Intelligence (AI) tools are becoming indispensable in managing requirements across the infrastructure project lifecycle. By leveraging sophisticated algorithms, AI facilitates the analysis of large datasets (and Traceability Matrices), streamlining requirements elicitation and fostering improved collaboration among stakeholders. AI's capacity to automate repetitive tasks, detect flaws in requirements, and predict potential risks enables organisations to minimise errors and align requirements more closely with project objectives. This adaptability ensures that requirements evolve alongside project needs, enhancing the delivery of infrastructure projects.

As the infrastructure industry evolves, project teams must embrace these advancements and take proactive steps to overcome the historical pitfalls of flawed or missing requirements. With today's advanced tools and methodologies, project failures owing to flawed requirements are no longer justifiable. This is a welcome opportunity to elevate standards and achieve sustainable success.

Requirements and Concept of Operations

Many project managers start their projects with '*copy-and-paste*' requirements from a previous project or a Work Breakdown Structure (WBS). Surely, when appropriately and adequately done, the WBS is a good technique for developing the project scope. However, its rationale and essence depend on "*how a community intends to use a contemplated 'system' to mitigate or suppress an actual or anticipated problem situation*" or the Concept of Operations (ConOps). Hence, defining a ConOps is essential in converging multiple stakeholders toward a common image and understanding of the intended system; this "*common image*" is what the WBS seeks to attain—but as to elements and work packages. Jumping to the WBS without *first* defining a viable ConOps makes no sense.

The ConOps document describes the desired or intended characteristics of the 'system' from the user's viewpoint and identifies the environment in which the 'system' will function. As an artefact, it serves as a vehicle to communicate the high-level quantitative and qualitative characteristics of the 'system' to the customer, users, buyer, developer, regulators, and other relevant stakeholders. Most Systems Engineering lifecycles, whether in a Vee-model format or otherwise, reflect the ConOps as the first, essential step of system development endeavour (e.g., megaproject delivery).

> "Concept of Operations (ConOps) is the first step in the systems engineering 'Vee' – while integral throughout the entire process, its most critical, and directly related, roles will be in the direct assistance to the generation of system requirements, and in system validation once it has entered an operations and maintenance phase […] the discussions among stakeholders during the Concept of Operations development were essential for the success of the system since all organizations came to a consensus on the vision for the project." (Smith, 2005)

The Concept of Operations answers the *why*, *what*, *who*, *when*, *where*, and *how* of the intended 'system', describes a *day-in-the-life* of the 'system' (i.e., normal and contingency operations), and provides context to all other aspects of such a 'system'. Unlike the WBS, the ConOps should reflect *out-of-the-box* thinking and, thus, not be concerned with immediate perceptions of feasibility.

At the practical level, identifying relevant stakeholders and their respective requirements arises from the ConOps. Whether the envisioned water treatment system will harvest rainwater and treat it before dispatching potable water to the city will entail a different set of "*stakeholders and their requirements*" than perhaps collecting sewage, treating it, and pumping it to the same community. Given its exploratory and speculative nature, as many ConOps as possible should be brainstormed.

Here is a simple example of a ConOps. A young boy was driving to school with his father. He often got annoyed that their fast and flashing-red sports car would only "roar and sprint" at 120 km/hour for a stretch of the highway, but then would come to a complete standstill every time they bumped into a traffic jam. The lad once looked around at the sea of cars stuck on the highway, closed his eyes for a few seconds as if to sink into deep thoughts, and in a burst of excitement, said to his father:

> "*Dad, I know exactly the kind of car you need to buy. It must be able to sense the traffic jam ahead and automatically shift to airplane mode to fly us beyond the damn jam, and then land back on the highway where you can drive at 120 km/hour again–you will still enjoy the driving and I will never be late to school again!"*

The boy is expressing a basic ConOps. He describes an *out-of-the-box* concept not concerned with *immediate* perceptions of the car's feasibility—telling the *why*, *what*, *who*, *when*, *where*, and *how*. Of course, other boys in this situation would probably envision different solution options or ConOps.

The *viewpoint* of a ConOps is from the "outside-in", describing the stimuli to which the intended 'system' is expected to respond and the effect the responses are intended to have on the situation. It depicts a 'system' of the future, yet to be designed; thus, it is necessarily speculative—a vision. Hence, meaningful requirements should derive from pertinent ConOps, lest the technical solution developed might not improve operations—a cure that would not remedy the malady in operations.

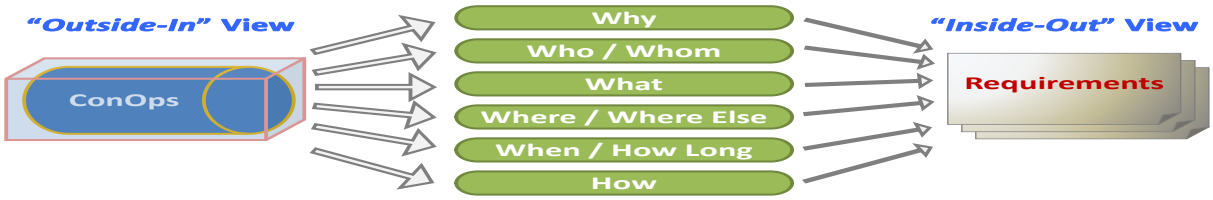


Figure 07 – Relationships Between ConOps and Requirements Outlooks

The generic outline of a *fully-fledged* ConOps that will evolve over the project lifecycle is as follows:

(1) **Scope** – 'system' identification, document overview, a high-level (or generic) overview of the intended system, and a brief description of the scope of effort required;
(2) **Knowledge References** – document identification numbers, titles, revisions, and dates of all documentation (and other artefacts) referenced in the ConOps document;
(3) **Operational Description** – the problem to be solved, and the 'system' or situation as it currently exists; the *why*, *what*, *who*, *when*, *where*, and *how* of the envisioned 'system';
(4) **System Overview** – reasons for developing the proposed 'system', meaning, (i) new or modified user needs, missions, or objectives, and (ii) dependencies or limitations of the current 'system';
(5) **Operational and Support Environments** – a high-level description of the proposed 'system' that indicates the operational features to be provided;
(6) **Operational Scenarios** – a *step-by-step* description of how the proposed 'system' should operate and interact with its users and external interfaces under a given set of circumstances and impacts from the user's perspective. It is a good idea to include failure events and anomaly handling.

Owing to its speculative nature, any ConOps should avoid assumptions about the internal content and structure of the eventual 'system'. It must avoid getting lost in details too early in the lifecycle, avoid premature feasibility (mis)-judgements, and preclude the early insertion of design concepts by "*force of habit*." Placing all observations about possible content and structure in appendices, not as part of the ConOps baseline, for later consideration by designers, achieves such avoidance.

Many in the industry still miss the fact that at the Conceptual (FEL-1) phase there could (or should) be many ConOps, each denoting a different concept or "operational option". In the water treatment plant example, harvesting rainwater as a concept differs from collecting sewage. Likewise, adding port capacity may involve expanding or upgrading an existing port facility or building a new one. These ConOps will be *optioneered* at the Pre-Feasibility (FEL-2) phase to select the most viable. Further, a ConOps may *initially* appear rudimentary or sketchy but shall evolve over the lifecycle; it should be judged solely on how effectively the needs of the intended benefactors are reflected.

As a matter of good practice, the ConOps shall reflect the owner's needs, and avoid these mistakes:
- Expecting external parties (e.g., vendors, contractors, partners) to develop the ConOps;
- Allocating inadequate resources (e.g., low-skilled staff) for its (initial) development;
- Postponing ConOps development until after the 'system' is designed or delivered;
- 'Cut-and-Paste' from another ConOps instead of devising a system-specific one;
- Neglecting to update the ConOps, not treating it as a *living* lifecycle deliverable.

Requirements and Project Lifecycle

Project requirements end only when users, customers, and other relevant stakeholders are satisfied. Systems Engineering (SE) treats the complete project lifecycle requirements as core elements. Thus, a specific step is built at each lifecycle phase to review whether the project requirements were met.

Getting requirements right at the beginning is critical because they run through the whole project and will be verified (i.e., proving each has been satisfied) and validated regularly with stakeholders (i.e., proving each is clear, correct, complete, consistent, and certifiable—in the project context). Project requirements are important and, thus, ought to be taken seriously throughout the lifecycle:

(i) The project team is putting efforts into satisfying requirements when developing the system;
(ii) Iteration of stakeholder consultations on requirements is crucial to secure "input and buy-in."

Stakeholders of a system-of-interest (SoI) may vary throughout the lifecycle. Therefore, to get a complete set of needs and the ensuing requirements, it is important to consider all lifecycle stages when identifying (classes of) stakeholders—from them, pertinent requirements are to be elicited. A list of stakeholders (those *interested* in the future 'system') must be identified at each project stage. The system delivery lifecycle relies on "requirements" to convert the 'Current State' into the 'Future State'. The goal is to get every stakeholder's point of view for every stage of the system life (i.e. *from womb to tomb*) to consolidate a complete set of stakeholder needs or issues that might be analysed, prioritised, and developed into the stakeholder requirements as exhaustively as possible.

The newly deployed system must function *properly* in its entire lifecycle—an inescapable condition:

> "A reliable system ensures mission success by functioning properly over its intended life. It has a low and acceptable probability of failure, achieved through simplicity, proper design, and proper application of reliable parts and materials. In addition to long life, a reliable system is robust and fault tolerant, meaning it can tolerate [minor] failures and variations in its operating parameters and environments." (NASA SEH, 2007) [*Underlining added for emphasis*]

For that, project requirements ought to evolve from Business Needs to (1) System Requirements ("*what the overall system shall do*") to (2) System Behaviour ("*How shall the system interact, both internally and externally*") to (3) System Architecture ("*What shall make up the system*", and to (4) System Measurements ("*How well shall the system work*"—viable shapes, sizes, physical/chemical and performance characteristics), in line with the "*Four System Domains*" framework (Scott, 2014).

This 'Requirement Lifecycle' process is about creating, maintaining, evolving, implementing, and delivering the requirements artefacts (as inputs into other project processes) throughout the system or project lifecycle. However, the responsibility to perform the "*works of the project*" rests with the owner and should only be transferred to a suitable *partaker* based on an Owner's Requirements Specifications (ORS). Therefore, neglect or error in requirements management could prove fatal.

> "A neglect of RM&E [Requirements Management and Engineering] in the starting phase of a project materializes only during the final phase of a project [...] A serious estimate [of costs and/or schedule, at any point in the lifecycle] can only be produced, if contractor and customer agree on the requirements of the system to be developed, at least roughly." (Hood, 2008)

It cannot be emphasised enough: cost and schedule estimates are contingent on good requirements. As requirements evolve and are increasingly elaborated, their quality and confidence improve too. A *holistic* set of requirements that talk to one another and the project context is a recipe for success.

> "If [system] requirements are defined without fully understanding the resources required to accomplish the needed technology developments then the program/project is at [greater] risk. Technology assessment must be done iteratively until requirements and available resources are aligned within an acceptable risk posture […] In the early phases of a project [FEL-1,2], risk and reliability analyses help designers understand the interrelationships of requirements, constraints, and resources, and uncover key relationships and drivers so they can be properly considered [in subsequent phases]." (NASA SEH, 2007) [*Underlining added for emphasis*]

This advice agrees with the lifecycle concept of "*progressive elaboration*." In these early phases, discussions and works should revolve around high-level concerns and contribute to reducing risks. Moreover, since 'infrastructure' is a technological system *nested* in socio-economic environments, the 'Primary Requirements' (i.e., the project objectives) should arise from socio-economic issues.

More to this point, an infrastructure system is generally a system-of-system that operates at the 'Business' layer of the HKM^2 hierarchy of complexity. As such, it contributes to improving operations in the 'Supply Chain', which then fulfils the objectives of 'Socio-Economic' realms. The capabilities and vulnerabilities of any infrastructure reflect the strengths and weaknesses of lower-level systems (i.e., constituent systems, products, components) and, in turn, contribute to the performance of higher-level systems (i.e., supply chain, socio-economic, regional, and global).

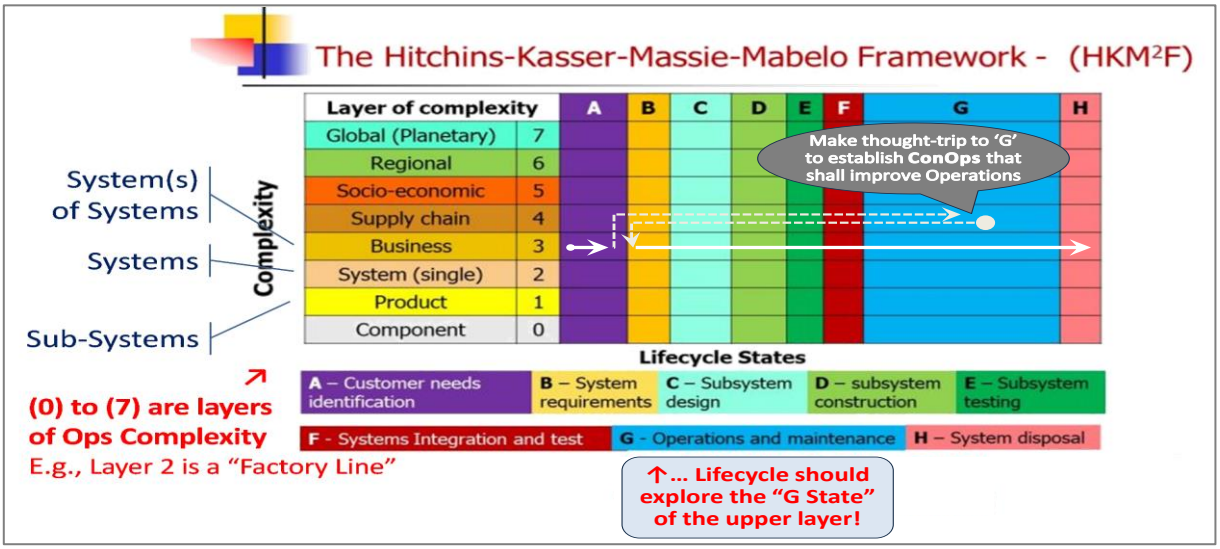


Figure 08 – HKM^2 Framework Depicting Layers of System Complexity and System Lifecycle

Regarding the ConOps, the HKM^2 Framework suggests concepts should be explored not in the system's operations but in the workings of the upper layer. For instance, Figure 08 illustrates that ConOps for Level 3 (Business, e.g., a seaport) should be *explored* at Level 4 (Supply Chain, e.g., transport policy, road/rail network). Infrastructure systems are designed to improve Supply Chain.

Yet, the author will insist on "*putting empathy in operations*" (Mabelo, 2020); based on the HKM^2, empathy (for the needs, ills, and challenges) should be put into the operations of the upper systems. From the identification and analysis of needs and their translation into ConOps (by exploring the respective upper level) during the A-Stage, 'requirements' will be documented during the B-Stage to lead to 'system' design, construction, testing, system integration, operations, and disposal stages. This approach ensures the new system fits and improves operations at upper levels, lest evolving requirements (i.e., requirements to behaviours to architectures to measurements) may turn futile.

Project Phase	Owner's Action	Project Team's Action	Phase System Output
FEL-1	Business Requirements (via SSM) Approved & Issued as [ORS $_{FEL-1}$]	System Requirements (*for each identified options*) Developed	Option-Solutions Generated
FEL-2	*Previous* + System Requirements (*for each identified options*) Approved & Issued as [ORS $_{FEL-2}$]	▪ System Behaviours Developed (*for each option*) ▪ System Architecture Developed (*for each option*)	Most-viable Option-Solution Identified & Recommended ... Selection Criteria/Matrix as per Objectives, ORS $_{FEL-2}$
FEL-3	*Previous* + System Behaviour and Architecture (*for selected option*) Approved & Issued as [ORS $_{FEL-3}$]	▪ (Sub-) System(s) Designed as per Behaviour & Architecture (*for selected option-solution)* ▪ Performance Measurements (V&V criteria, Plans) Developed	▪ Selected Option-Solution Designed ▪ Selected Option-Solution Planned for "Execution & Operations & Disposal"
FEL-4	*Previous* + System Measurements Approved & Issued as [ORS $_{FEL-4}$]	Performance Measurements (V&V criteria, Plans) fitted into Contracts	Construction Readiness Confirmed
Construction	Verification & Validation Plans (Interim/Final) Approved & Issued	(Sub-) System(s)/Components Built and Verified & Validated as per Plans	System (i.e. Facility/Asset) Integrated & Realised
Close-Out	Final Verification & Validation Reports Approved & Issued	Requirements & Performance(s) Lessons-Learned Collated & Issued	System (i.e. Facility/Asset) Handed over to Operations

Figure 09 – Project Lifecycle and Requirements Responsibilities (by Owner and Project Team)

Exploring a set of ConOps and translating the same into a set of requirements is necessary but not always sufficient to secure effective infrastructure delivery. For instance, a common *malpractice* in the industry entails expecting *detailed* requirements from the project onset. It is not feasible, practical, or even advantageous to discuss "*elaborate*" requirements in these early project stages. Nobody has yet attained "*sufficient mass of understanding*" on the project to engage at that level; they only get that by *copying and pasting* from another, somewhat similar project—to their peril!

Rather than suffer the proverbial chicken-and-egg situation (viz, "*I need to define requirements to start the project—but cannot get any until the project has started*"), the approach in Figure 09 breaks this vicious circle in line with the progressive elaboration principle. It ensures that hastily developed, or even precipitate requirements, which fail to align with operational needs, are no longer a concern.

From a high-level definition of Business Requirements (arising from Soft System Methodology), the project team would proceed to define a viable set of ConOps and their associated requirements. The owner will then be allowed to apply their mind to those specific phase outputs and revise the Owner's Requirements Specifications (ORS) for the next phase (e.g., FEL-2), and so on, to define the Systems Behaviours, Systems Architectures, Systems Measurements, and other deliverables.

The goodness of the proposed approach to Requirements Management and engineering is twofold:

(1) The owner (or an implementing agency on their behalf) is not overwhelmed with the demand by the project team (or the systems engineer) to furnish a detailed set of requirements at an early stage when no "*sufficient mass of understanding on the project*" is available to them;
(2) By tabling high-level (socio-economic or business) requirements, the owner may engage with the project team in a 'Develop—Review—Approve' cycle that evolves to gather a "*sufficient mass of understanding*" over the lifecycle to produce the right requirements at the right time.

Indeed, capturing a requirements snapshot that resembles a near-final configuration and using it at any other point or phase of the project lifecycle is inconsistent with Systems Engineering principles.

Conclusion

The successful delivery of large and complex infrastructure projects (LIPs) hinges on effective requirements management throughout the project lifecycle. Requirements constitute the bridge that connects the envisioned future state and operational realities, ensuring alignment with stakeholder needs and socio-economic objectives. Defined as a collection of capabilities derived from users and stakeholders, requirements must encompass functional, performance, and interface specifications—and prove conducive to verification and validation (V&V), especially in large infrastructure projects.

Research consistently shows that project success is strongly correlated with the level of requirements maturity. It is held that "*no matter the delivery method employed, the project success rate is strongly [and positively] correlated to the level of requirement maturity*." Lower maturity levels often lead to cost and/or schedule overruns and operational failures, particularly in large-scale infrastructure projects. Therefore, organisations involved in LIP delivery must strive to elevate the maturity of requirements to reduce risks and steadily improve outcomes—this is critical to long-term viability!

Systems Thinking plays an important role in this process by addressing the *interconnectedness* of requirements across technical, operational, and contextual domains. Iterative refinement, strategic stakeholder engagement, and progressive elaboration of requirements minimise ambiguities and ensure alignment with project goals. In contrast, misaligned, inconsistent, and incomplete, namely, 'wrong' and 'poor' requirements lead to inefficiencies, unwarranted cost escalations, and failure. Thus, project requirements should evolve throughout the project lifecycle, from 'business needs' to 'system requirements' to 'system behaviours' to 'system architectures' to 'system measurements'.

Advanced tools like Artificial Intelligence (AI) can further enhance requirements management by providing precision in decomposition, maintaining traceability, and fostering collaboration around requirements. Further, by creating and searching requirement repositories, AI tools improve project outcomes and contribute to resilient and high-quality infrastructure solutions over the long term.

The reviews of practical applications imparted herein (including cases such as the Red Valley Bridge, and one or two other examples in Annexures) confirm that ineffective 'requirements management' remains the root cause of cost and schedule overruns, culminating in persistent operational failures. Those real-life examples highlight the "*practical relevance*" of the notions discussed in this paper—exposing the reader to instances where 'effective requirements management' (or lack thereof) has made a significant difference. Project success entails "*right requirements being implemented right*," a precept project sponsors, business and systems analysts, engineers, and contractors should adopt.

Ultimately, every aspect of project management—from budget and schedule to scope, design, and construction—revolves around requirements. Without clear, exact, and well-managed requirements, the process risks losing its purpose and direction. It is akin to a game of golf: without the 18 holes defining the course, there is no game—no teeing off, no strokes, no caddy, and no purpose for the clubs and balls. Just as the golf course structure (i.e., layout, hazards, etc.) gives meaning to every element of the sport, well-defined requirements provide an effective foundation for successful project delivery. This perspective underscores the central argument of this seminal article: managing requirements is not just a component of project management, like cost control—it is its very essence.

For this reason, effective requirements management must transform infrastructure challenges into welcome opportunities for sustainable success. Further, by prioritising this discipline, organisations involved in infrastructure delivery can achieve improved operations, socio-economic and business value, and stakeholder satisfaction while raising delivery standards for modern infrastructure.

Annexures

Annexure 1: Requirements Classification

The INCOSE (2015) proposes a Requirements Typology to be elicited from various stakeholders:

Category per INCOSE	Description/Definition
Systems Requirements	It is the totality of requirements; they result from the analysis of the input Stakeholder Requirements, as required to meet project and design constraints such as Functional Requirements, Non-Functional/Quality Requirements, Performance Requirements and Architectural Constraints.
Functional Requirements	These refer largely to what the system should do. These should be "action" oriented and describe the tasks or activities the system performs during its operation. They often answer Performance Questions such as who, what, where, when, how many, and which ones. For example – E.g., the <bridge> must be capable of <crossing the waterway and accommodating> not less than <1,000 cars> per <hour>. These functions encapsulate the primary essence of the system and the reasons why it should exist. For instance, read fingerprints; emit "grey" light; and stop water flow.

… and many other types of requirements.

The SEBoK proposes a slightly different Requirements Typology to be elicited from stakeholders. It also includes operational aspects of requirements related to internal and external environments:

Example of Stakeholder Requirements Classification [Adapted from (SEBoK, 2015)]	
Stakeholder Requirement Type	Description
Service or Functional	Sets of actions to perform the mission or operation of the system-of-interest; enhanced by effectiveness or performance characteristics attached to the mission or operations.
Operational	This category may include: ▪ Operational concepts that indicate the operational features to be provided without specifying design solutions. ▪ Operational scenarios describing the sequence or series of activities supported by the system-of-interest. ▪ Operational and transition modes between states/modes of the system-of-interest during utilisation to include dynamic interactions between the system-of-interest (viewed as a black box) and the system-of-interest's interface with external components in the context of its use.
Interface	Matter, energy, or data flows are exchanged between the system-of-interest and its external components via physical interfaces in the context of its use.
Environmental	External conditions that affect the system when in operation.
Utilization Characteristics	The '*-ilities'* are used to indicate conditions of the utilization of the "system-of-interest" (e.g. usability, dependability, maintainability, safety, security).
Process Constraints	These requirements do not directly address the end-item system, but rather how the end-item system will be developed/provided.
Project Constraints	Limits to delivering the project or end-item system within cost and schedule.
Business Model Constraints	Constraints related to the expected business goal achieved by the system-of-interest, when this is relevant within the context of use, which may include: geographic position (local, national, international) of the future product, service, or organization resulting from the system-of-interest, distribution channels, alliance and partnership, and finance and revenue model, etc.

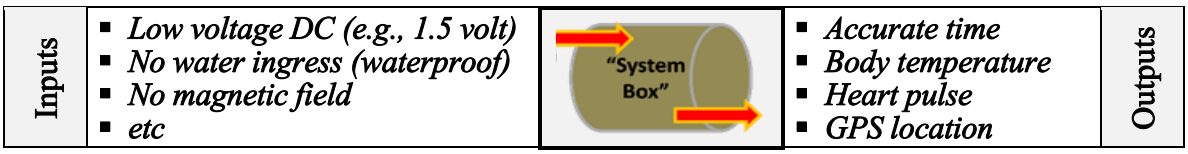
Annexure 2: Example of Four Domains of Requirements

The 'Requirement Lifecycle' is about creating, documenting, maintaining, evolving, implementing, and delivering requirements artefacts over the system or project lifecycle. It usually coincides with the system development lifecycle or design works; requirements and design outputs evolve together.

In devising a new 'system' designers should navigate through the 'four domains of requirements'—system Requirements, System Behaviours, System Architectures, and System Measurements. Most design engineers are *primarily* trained in Detailed Design, focusing on 'measurements'. As a result, they often skip intermediate steps, moving directly from 'system requirements' to 'architectures' or defaulting to familiar designs. This approach, however, leads to project inefficiencies and challenges.

Suppose a team is tasked to devise a *wristwatch*; they need to cover the four domains as follows:

(1) ***System Requirements:*** "*What the wristwatch shall do*", or else "*What shall come out*" and "*What shall go into"* the watch? Here, the desired *wristwatch* is considered a closed 'system box'; input and output items are mass/substance, energy, information/data, or combinations thereof.



(2) ***System Behaviours:*** "*How shall the wristwatch function and interact?*" Internal and external functions and interactions stated in verbs are needed to produce or support the desired outputs.

Sample of Selected Functional Behaviours for the New Wristwatch			
01	Display data	07	'Read' heart pulse
02	Convert DC to 'Eternal timer'	08	Display heart pulse
03	Count time on 'Eternal Timer'	09	'Read' GPS location
04	Display time, on an external screen	10	Display GPS location
05	'Read' body temperature	11	Stop/prevent water ingress
06	Display body temperature	12	Shield from any magnetic field

(3) ***Systems Architecture:*** "*What components/interfaces in the wristwatch would yield the desired behaviours?"* It is not only about 'components', but also about the interfaces and connections. Such 'objects' are vital to carry out and/or support the Functional Behaviours itemised in (2):

Architecture for the New Wristwatch			
01	Eternal Timer 'chip'	05	Display switch, associated with [04]
02	Temperature and Pulse 'readers'	06	Low voltage battery (e.g., 1.5 volts)
03	GPS 'locator'	07	Waterproof encasing
04	Multiple Digital Display (MDD)	08	'Magnetic Shield' sheet

(4) ***Systems Measurements:*** "*How well shall the wristwatch's elements conform and perform?*" In this ultimate 'design' step, "*numbers"* are finalised and allocated to *detailed* requirements.

This step, which pertains to Detailed Design, shall determine the "*shape, size, and technical and performance characteristics*" of each part, component, subassembly, etc—say, "*Item [06] shall be a 7 mm radius by 3 mm thick disk as a 1.55 V Lithium battery, with a 35 mAh capacity*"—and, by extension, determine the *overall* shape, size, and performance of the envisaged system.

Annexure 3: Example of Flawed Tender Requirements (The FlorLib Project)

The FlorLib Project involved upgrading a library, including renovating its bathroom facilities to modern standards. To that effect, a tender was issued to select and eventually appoint a contractor.

The author used patented AI software to review the 'Bathroom Requirements' and determine their quality status. The review report highlighted several 'requirements flaws' summarised as follows:

(a) 95% of requirements entries were categorised "Very High Risk". Many requirements lack an imperative, include multiple imperatives, or contain excessively vague terms —Consider rewriting the requirement or separating it into several concise requirements.

(b) 36% of items included 'vague words' (e.g., Ensure that correct water level has been set).

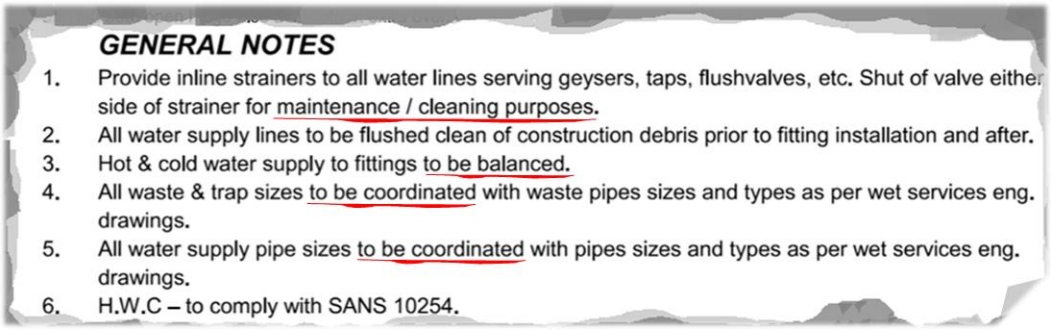
(c) 27% included "excessive continuances" (e.g., Operating overflow tube height conforms to and operates to local bylaws and SABS specification)—Verifying this string of disparate conditions or states would prove difficult and lead to disagreements.

(d) 9% involved "universal quantifiers" (e.g., Complete all plumbing; Always use silicone)

[Underlining is added by the author for emphasis]

Surely this bathroom tender scenario and the abovementioned 'requirements flaws' findings are not peculiar to the FlorLib project. This predicament is widespread in the infrastructure and many other industries, including the IT sector. Projects are doomed to fail due to flawed requirements. The ambiguities, vagueness, inconsistencies, and incompleteness in the requirements conveyed to appointed design consultants, contractors, and other project practitioners are enough to mislead their efforts; these predicaments can only lead to misunderstandings, complications, and conflicts.

In the case of this FlorLib project, the bathroom tender package was issued to solicit a contractor. If bidders raise no technical queries, the package is as good as an 'IFC' (Issued For Construction) specification. However, one can foretell the possible outcomes of this project: either (i) a string of technical/engineering queries would ensue, or (ii) the work done would result in operational failures.



GENERAL NOTES

1. Provide inline strainers to all water lines serving geysers, taps, flushvalves, etc. Shut of valve either side of strainer for maintenance / cleaning purposes.
2. All water supply lines to be flushed clean of construction debris prior to fitting installation and after.
3. Hot & cold water supply to fittings to be balanced.
4. All waste & trap sizes to be coordinated with waste pipes sizes and types as per wet services eng. drawings.
5. All water supply pipe sizes to be coordinated with pipes sizes and types as per wet services eng. drawings.
6. H.W.C – to comply with SANS 10254.

Extract showing Vague Words in Bathroom Requirements on the FlorLib Project

Adopting this level of requirements at the onset of the Feasibility (FEL-3) phase might not have caused the same or similar difficulties—at that stage, what matters the most is defining a plausible 'system architecture' (i.e., various components and their desired interactions). Therefore, while using such "*incomplete and unclear*" requirements is problematic at the IFC or Construction stage, the same stance may still be acceptable at the Pre-/Feasibility (FEL-2 or FEL-3) stages, provided they include a plausible listing of components with their intended arrangements and interactions.

Annexure 4: Example of Flawed Owner's Requirements Specifications (ORS)

The text below is extracted from a real-life ORS document issued by the owner (explicitly, by the Project Director from the contracting entity) to the project team at the Pre-Feasibility (FEL-1) Phase:

> *An intelligent dust-eliminating spray system shall be used on the stockpiles. Feed pipes shall be run above ground as far as possible. Spray system design shall consider access to spray all areas of the regular stockpiles.*
>
> *Stockyard design shall minimise settlement during operation by preloading of the area, and shall incorporate a proven style of drainage facilities. Stockyard equipment shall incorporate means of readily adjusting for any settlement that occurs. The berm and rail design shall consider and make provision to offset both vertical and lateral movement and the effect on the operation of the Stacker and Reclaimer machines.*
>
> *The stockyard base shall be constructed of consolidated fill to the design levels. To prevent contamination of manganese product with the compacted base material, it is recommended that a "carpet layer" of manganese ore material be provided by the manganese ore exports to the constructors prior to commissioning. The grades and quantities of material are to be determined prior to placement of this "carpet layer".*

This mere 165-word extract (of the ORS) already contains 12 weak or vague words, not to mention 10 "universal" quantifiers (i.e., all, and, any, both), which tells a lot about the quality of this ORS. Besides, how intelligent is "intelligent"? How does one establish that something was considered? Further, the bulk of this ORS (FEL-1) discusses 'Architecture', as opposed to "*what the system shall do*"—one would think this ORS was towards the Feasibility Phase (FEL-3), proceeding to 'Measurements' where Detailed Design (i.e., allocate shape, size, performances) must take place.

Moreover, the *prosaic* character of this ORS, with long and winded phrases, will not lend itself to effective (requirements) 'verification' (i.e., proving each has been satisfied) and 'validation' (i.e., proving each is indeed clear, correct, complete, consistent, and certifiable—in the project context). How would one confirm that "*both vertical and lateral movement and the effect on the operation of the Stacker and Reclaimer machines"* are offset as intended and satisfy the needs of the client?

This impediment could have been prevented by adopting an *atomic* rendition of the requirements:

> *Rq xy1: An intelligent dust-eliminating spray system shall be used on the stockpiles.*
> *Rq xy2: Feed pipes shall be run above ground as far as possible.*
> *Rq xy3: Spray system design shall consider access to spray all areas of the regular stockpiles.*
> *Rq xy4: Stockyard design shall minimise settlement during operation by preloading the area.*
> *Rq xy5: Stockyard design shall incorporate a proven style of drainage facilities.*
> *Rq xy6: Stockyard equipment shall incorporate means of readily adjusting for any settlement.*
> *Rq xy7: The berm design shall make provision to offset vertical movement.*
> *Rq xy8: The berm design shall make provision to offset lateral movement.*
> *Rq xy9: Et cetera, et cetera.*

Apart from their quality shortcomings (i.e., weak words; universal connectors) and "prochronism" (FEL-3 items prematurely applied at FEL-1), these requirements are structured as 'atomic' (i.e., individually numbered) and lend themselves to verification and validation. However, the above concerns have probably introduced *complications* to the multi-billion project that employed such an ORS, contributing significantly (if not primarily) to its substantial cost and schedule overruns.

References

01 BA Guide, 2017 — Project Management Institute, 2017. Business Analysis for Practitioners: A Practice Guide. Project Management Institute.
02 BABoK, 2015 — A Guide to The Business Analysis Body of Knowledge. IIBA. Toronto.
03 Buttrick, R. 2003, Project Workout. 3rd Ed. London: FT Prentice Hall.
04 Forsberg, K. , Mooz, H. and Cotterman, H., 2005. Visualizing project management. 3rd ed. New York: John Wiley & Sons.
05 Haik, Y. and Shahin, T., 2011. Engineering Design Process. Stamford, CT: Cengage Learning.
06 Hood, C., 2008. Requirements management. Berlin: Springer.
07 IAG Consulting: Reasons Projects Fail, 2009 [YouTube video from https://youtu.be/4a4ZxOAQifE]
08 IBM Business Research, 2017 — Dehghan, A., Neal, A., Blincoe, K., Linaker, J. and Damian, D., 2017. Predicting likelihood of requirement implementation within the planned iteration: an empirical study at IBM. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories.
09 INCOSE, 2015 — International Council on Systems Engineering (INCOSE). (2015). INCOSE systems engineering handbook: A guide for system life cycle processes and activities (4th ed.).
10 Kendall, G.I. and Rollins, S.C., 2003. *Advanced project portfolio management and the PMO: multiplying ROI at warp speed*. J. Ross Publishing.
11 Locatelli, G., Mancini, M. and Ishimwe, A., 2014. How can system engineering improve supplier management in megaprojects? Procedia-Social and Behavioral Sciences.
12 Mabelo, P. B., 2016. Application of systems engineering concepts as enhancements to the project life cycle methodology. Masters. University of Witwatersrand.
13 Mabelo, P.B., 2021. Managing Engineering Processes in Large Infrastructure Projects. Cambridge Scholars Publisher.
14 Mabelo, P. B. (2022). Post-Implementation Reviews - Benefits to project and operations teams; featured paper, PM World Journal, Vol. XI, Issue XII, December.
15 NASA SEH, 2007 — NASA, (2007). NASA Systems Engineering Handbook. 1st Ed. Washington D.C: National Aeronautics and Space Administration.
16 Oehmen, J., (ed.), 2012. The guide to lean enablers for managing engineering programs, version 1.0. Cambridge, MA: Joint MIT-PMI-INCOSE Community of Practice on Lean in Program Management. [online] Available at: http://hdl. handle.net/1721.1/70495 Accessed 24 Oct 2019].
17 PMBoK. 2013. A Guide to the Project Management Body of Knowledge (PMBoK Guide). Project Management Institute, 5th Ed.
18 PMI, 2014 — Project Management Institute, 2014. PMI's Pulse of the Profession: Requirements Management—A Core Competency for Project and Program Success. Project Management Institute.
19 PMI, 2016 — Project Management Institute, 2016. Requirements management: A practice guide. Project Management Institute.
20 Robertson & Robertson, 2005 — Robertson, S. and Robertson, J., 2005. Requirements-Led Project Management. Boston: Addison-Wesley.
21 Smith, B., 2005. Developing and Using a Concept of Operations in Transportation Management Systems. *Washington, DC, 20590*.
22 Scott, Z., 2012. 9 Laws of Effective Systems Engineering. White Paper. Vitech Corporation.
23 Scott, Z., 2014. "Engineers Behaving Badly". [Online Video]. 26 February 2014.
24 Senge, P.M. (2006). The fifth discipline: The art and practice of the learning organization. New York, NY: Crown Publishing Group.
25 Tapke, Jennifer & Muller, Allyson & Johnson, Greg & Sieck, 2001. Josh, House of Quality: Steps in Understanding the House of Quality.

About the Author

Pascal Bohulu Mabelo

Johannesburg, South Africa

Pascal Bohulu Mabelo *(MBA, MSc Industrial, BSc Civil, Pr. Eng, Pr. CPM, Pr. PMSA, PMP)*, has more than 25 years of professional experience and possesses a wide range of technical and managerial skills in large and complex infrastructure projects. He has worked on large infrastructure projects as a design engineer, project/programme manager, project consultant and project management executive. Pascal was honoured to serve as the national chairman of Project Management South Africa (PMSA), the leading Project Management professional association in Southern Africa.

Pascal has published the book: "*Managing Engineering Processes in Large Infrastructure Projects*" (Cambridge, 2021); he has also published, "*How to Manage Project Stakeholders—Effective Strategies for Large Infrastructure Projects*" (Routledge, 2020) and "*Operational Readiness—How to Achieve Successful System Deployment*" (Routledge, 2020). Through various publications, journal articles, and conference presentations, he assiduously promotes the application of Systems Thinking and/or Systems Engineering principles, concepts, and practices to unravel complexity in Large Infrastructure Projects (LIPs) to address their persistent risks of failure and their massive, even pernicious, cost and schedule overruns.

Pascal is currently a Director and Principal Consultant at E 6 Project Consulting or E6PC; for comments, further information, and clarifications he may be contacted at Consult@e6pc.com. His other papers can be viewed at https://pmworldlibrary.net/authors/pascal-bohulu-mabelo/