FROM TAXONOMY TO INTERVENTION

Non-functional requirements (nFRs) are critical to systems development yet remain problematic in practice. There is a gap between academic work and practice stemming from a difference in the question being asked. Academic traditions ask, "what is an nFR?", while practice asks, "how do I safeguard this property in my design?" An intervention-oriented framework is proposed that distinguishes three safeguarding mechanisms: functionalisation, budgeting and system-level validation. A decision logic based on three corresponding questions guides mechanism choice. This article does not present new practices but articulation of existing ones, providing shared vocabulary for knowledge transfer.

ERIK PUIK

Introduction

Non-functional requirements (nFRs) are widely recognised as critical to systems development. They often determine the difference between a system that technically works and a system that actually meets the expectations of users, clients and regulators. Properties such as safety, reliability, maintainability and cost are rarely optional; they constitute the constraints within which a system must operate and the criteria against which it will ultimately be judged.

Yet nFRs remain problematic in practice. Projects struggle with specifying, tracking and realising these properties. As systems grow more complex, the challenge intensifies. And despite decades of academic research, no broadly accepted approach has emerged for handling nFRs during design. This is surprising. The literature on nFRs is extensive. Taxonomies have been developed, classification schemes refined, and standards established. Academic attention has not been lacking. Nevertheless, a persistent gap exists between how nFRs are treated in the literature and how practitioners handle them in real projects.

The academic tradition asks: what is an nFR? This question leads to the above-mentioned taxonomies, definitions and classification schemes. It seeks conceptual clarity: what distinguishes an nFR from an FR, how different types of nFRs relate to one another, and how they should be documented.

Practice asks something different: how do I ensure this property is safeguarded in my design? This question leads to concrete design decisions about how and where an nFR is anchored in the development process. It is not about classification but about intervention.

These are not the same questions, and they do not lead to the same answers. Taxonomies help organise knowledge but offer little guidance on action. Practitioners, meanwhile, have developed effective strategies for handling nFRs, but these strategies remain implicit, unnamed, and difficult to transfer.

Here, an alternative approach is proposed: one that starts from what designers do with nFRs, rather than from what nFRs are. The result is an intervention-oriented framework that distinguishes three safeguarding mechanisms and offers a decision logic for choosing among them. By articulating existing practices, i.e., naming what experienced designers already do implicitly, a shared language emerges that enables transfer, comparison and improvement.

Background and problem statement

In academic research on nFRs two dominant traditions can be distinguished: taxonomic (classifying nFRs according to type, representation or origin) and goal-oriented (treating nFRs as soft-goals that can be systematically decomposed and operationalised). Both traditions, however, say relatively little about what a designer should do with an nFR once it has been identified and classified.

Empirical research demonstrates that in real projects nFRs are often treated ad hoc, carried implicitly rather than explicitly, or addressed late in the development process when options for change have narrowed. Practitioners handle nFRs by converting them into something tractable and rely on personal experience and tacit knowledge. Experienced designers have developed effective strategies. For example, safety is often functionalised: converted into FRs that lead to implementation as emergency stops, fault-detection mechanisms, and redundant subsystems. Cost is typically

AUTHOR'S NOTE

Erik Puik is professor of System Design & Realisation at Fontys University of Applied Sciences in Eindhoven (NL). This article is an abridged version of the paper he presented at the 2026 International Symposium of INCOSE (International Council on Systems Engineering), 13-18 June 2026 in Yokohama (JP). There, he also gave a tutorial, based on the book he published last year: "Beyond the V-model: Integrating Governance, Design Logic, and Iteration", ISBN 978-90-835996-0-1, Three Dot Fourteen Publishing, Eindhoven, 2025.

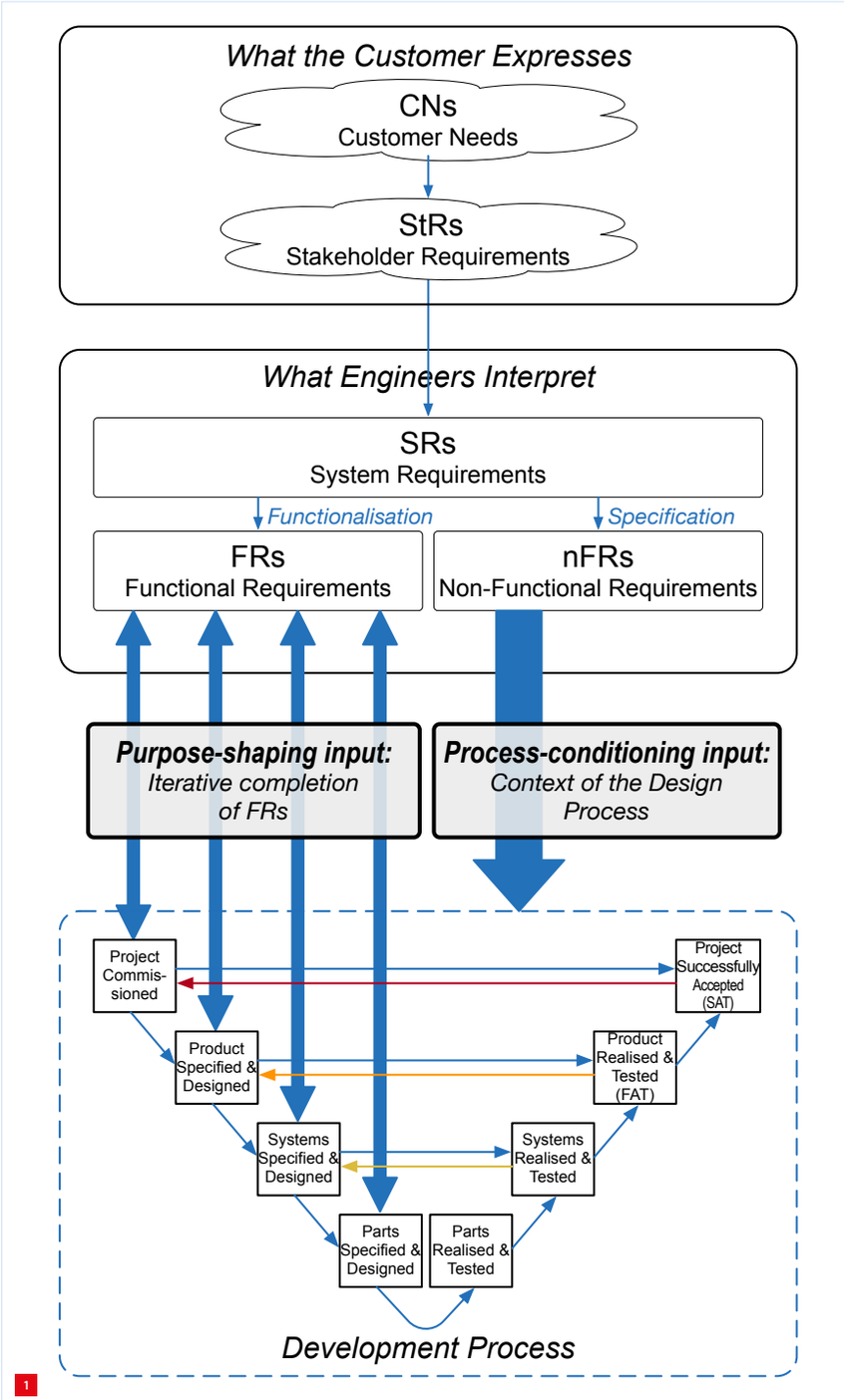
Please contact the author for the full paper, including references.

The author acknowledges Fontys, the NXTGEN Systems Engineering programme, and the NXTGEN Capability Booster initiative for their support of the broader context in which this work was developed. AI-based tools were used for limited editorial support. Responsibility for the content, interpretation, and final form of the full paper rests entirely with the author.

erik.puik@fontys.nl
www.incose.org

budgeted and allocated to subsystems. Properties like sustainability or electromagnetic compatibility are validated at system level during integration.

Figure 1 illustrates the different roles that FRs and nFRs play in system development. FRs serve as purpose-shaping input: they define what the system must do and drive the design process directly, while nFRs serve as process-conditioning input, specifying what the system must comply with and constraining the context within which design takes place.



FRs shape design through iterative completion across system levels, while nFRs condition the design context.

This distinction is consequential. FRs flow into the design structure; they are decomposed, allocated and realised through design parameters. While nFRs do not flow in this way; they bound the solution space, but the question of how they are safeguarded during design remains open. The consequence is a double gap. Academic work has limited impact on practice because it addresses a different question. Practical knowledge remains tacit and person-dependent because it lacks an explicit framework and proper documentation.

To resolve this situation, academic research should start from the question that practitioners actually face: not "what kind of requirement is this?", but "how do I safeguard this property in my design process?" This is an intervention question rather than a classification question, asking not for a label but for a strategy. An intervention-oriented approach would be inherently prescriptive, would connect the nature of the nFR to the strategy for handling it, and would provide a shared vocabulary for what practitioners already do.
Three specific limitations of the current situation can be identified:

1. Existing taxonomies are descriptive but not prescriptive.
2. The relationship between nFR and design action is implicit.
3. A shared language for safeguarding mechanisms is missing.

This article offers a practice-based conceptual contribution focused specifically on how individual nFRs are safeguarded during design. The goal is to make explicit what is currently implicit, and thereby to enable a conversation that cannot otherwise take place.

Approach

The framework presented here emerged from over three decades of experience in systems engineering practice and applied research, spanning domains including high-tech equipment, precision engineering, semiconductor manufacturing, and medical devices. Collaboration with industrial partners on complex multidisciplinary systems provided recurring exposure to how experienced engineers handle nFRs, and to the difficulties that arise when such handling remains implicit. The patterns described here were observed across these contexts; the abstraction into three mechanisms was a subsequent analytical step.

The central question guiding the analysis was: what must be safeguarded during design, and when can that safeguarding be delegated to the design structure itself versus when must it remain external? By applying this question systematically across diverse nFRs, three recurring patterns emerged:

1. Transformation of the nFR into explicit functions within the design structure.
2. Decomposition of the nFR into budgets or limits per subsystem.
3. Validation of the nFR at system level during integration.

These patterns were then abstracted into three safeguarding mechanisms: functionalisation, budgeting and system-level validation. The cases presented here serve as demonstration, not proof. They show that the framework can order practical situations and that the decision logic discriminates meaningfully between different nFRs.

The intervention-oriented framework

Central question

With nFRs specifying what the system must be or what constraints it must satisfy, the framework starts from a single question: how is the required property safeguarded during design? Safeguarding means ensuring that the property remains true across the design. An nFR is not satisfied merely by being documented; it is satisfied when the resulting system actually exhibits the required property. The question is when and how that assurance is established.

Three safeguarding mechanisms

Analysis of how experienced practitioners handle diverse nFRs reveals three fundamentally different approaches.

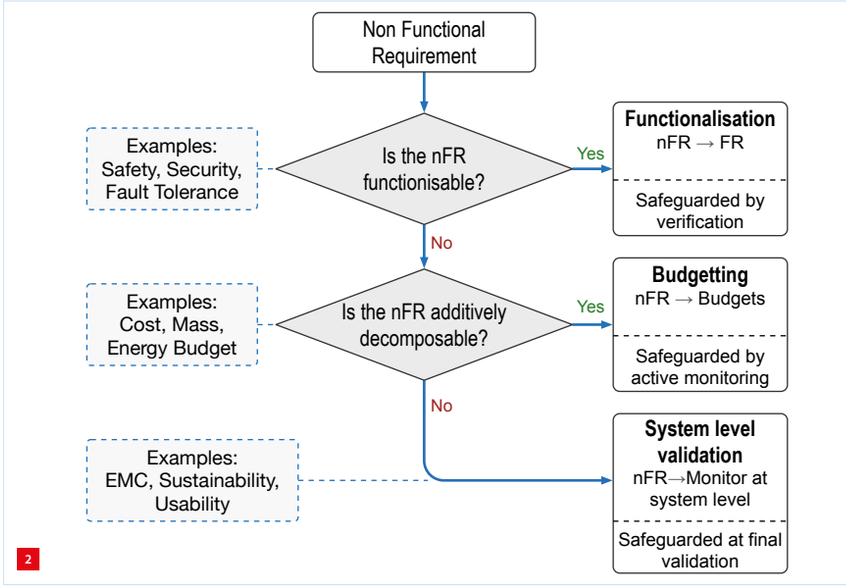
1. Functionalisation:
   - Working principle: The nFR is transformed into explicit FRs that become part of the design structure. Once these FRs are specified and implemented, the nFR is structurally secured. Safeguarding becomes verification: does the function perform as intended?
   - Example: Safety
     The nFR 'The system must be safe' is functionalised into specific FRs:
     a) The FR 'Execute the defined emergency response' may lead to implementation of an emergency-stop mechanism.
     b) The FR 'Detect fault conditions' may lead to implementation of a fault-detection system.
     Each FR has its own performance indicators, the measurable values that specify how well the function must perform. Stringent performance indicators, such as high reliability requirements, may necessitate redundant implementation.
   - Applicability: This mechanism applies when the nFR can be meaningfully expressed as FRs; not all nFRs permit this transformation.

2. Budgeting:
   - Working principle: The nFR is decomposed into budgets or limits allocated to subsystems. The constraint moves downward through the hierarchy, with each level receiving its own allocation. Safeguarding is active monitoring at each level: does each design choice stay within its allocated budget?
   - Example: Cost
     A total cost target is decomposed into budgets per subsystem. Each design team receives an allocation and is responsible for staying within it. The system cost is the sum of the subsystem costs. Performance indicators track realised cost at each level of decomposition.
   - Applicability: This mechanism applies when the nFR can be meaningfully divided into budgets, when the system property is a legitimate aggregation of subsystem properties. Cost and mass satisfy this condition. Mass budgeting in aerospace and automotive provides a clear illustration. A vehicle mass target is allocated across subsystems: chassis, powertrain, interior, electronics, with each team designing within its allocation. The system mass is the sum of component masses, tracked continuously through design reviews. Properties involving complex interactions between subsystems typically do not permit such decomposition.

3. System-level validation:
   - Working principle: The nFR remains at system level and is only verifiable at integration. The constraint stays high in the hierarchy; there is no meaningful decomposition. Safeguarding is final validation: does the complete system fall within the allowed bounds?
   - Example: Sustainability
     Environmental impact depends on the complete system across its lifecycle. The overall sustainability is not simply the sum of component sustainabilities: it depends on how components interact, how the system is used, and how it is eventually disposed of. The performance indicator measures the property at system level: carbon footprint, material circularity index.
   - Applicability: This mechanism applies when the nFR cannot be functionalised or budgeted: when it arises from interactions rather than aggregation, and when meaningful decomposition is not possible.

Decision logic

The distinction introduced in Figure 1 raises a practical design question. If nFRs do not drive the design process directly, but condition the context within which design decisions are made, how should a given nFR be safeguarded during design? Experienced practitioners address this question implicitly by anchoring different nFRs in different parts of the development process. Analysis of these practices reveals that the choice is not arbitrary but follows a small number of recurring patterns. Figure 2 makes this implicit reasoning explicit by presenting a decision logic for selecting an appropriate safeguarding mechanism based on the nature of the nFR.



Decision logic for selecting safeguarding mechanisms based on the nature of an NFR.

Three questions guide the choice among mechanisms:

1. Can the nFR be functionalised?
   Can the required property be expressed as explicit system functions? If yes, use functionalisation.
2. Can the nFR be budgeted?
   Can the required property be divided into budgets per subsystem? Is the system property a legitimate sum of subsystem properties? If yes, use budgeting.
3. Must the nFR be validated at system level?
   Does the property arise from interactions between subsystems? Is it only observable at system level? If yes, use system-level validation.

This is a heuristic, not an algorithm. The questions provide guidance, not determination. Context, domain knowledge, and engineering judgement remain essential.

Mixed forms

Real nFRs do not always fit neatly into a single mechanism. Mixed forms arise when budgeting helps but does not suffice. One example is electromagnetic compatibility (EMC). Local measures (shielding, filtering, grounding) represent budgeting at component level. But system-level EMC behaviour arises from interactions and can only be verified through integration testing. Both mechanisms operate simultaneously: budgeting where possible, system-level validation for final verification.

Summary

The choice of safeguarding mechanism (see Table 1) determines what kind of assurance activity is needed and at what level it operates: verification of functions for functionalisation, budget monitoring per level for budgeting, system-level measurement for system-level validation.

Table 1
Overview of the safeguarding mechanisms for nFRs.

Mechanism	What happens with nFR	Safeguarding
Functionalisation	Becomes FR	Verification
Budgeting	Becomes budget per subsystem	Active monitoring
System-level validation	Remains system-level	Final validation

Failure patterns

Each mechanism rests on assumptions that may not hold. Functionalisation assumes complete translation; budgeting assumes valid decomposition; system-level validation assumes correction space remains at integration. When these assumptions fail, characteristic problems emerge. Understanding the failure patterns enables diagnosis when projects encounter difficulties with nFRs (see Table 2).

1. Functionalisation – Failure through incomplete translation:
   - Core assumption: The nFR has been fully and correctly translated into functions.
   - Failure pattern: The functionalisation is incomplete. Some aspect of the nFR has not been captured as a function, or the functions do not fully cover the nFR.
   - Symptom: The system passes all functional tests but fails to exhibit the required property.

2. Budgeting – Failure through budget erosion or invalid decomposition:
   - Core assumption: The decomposition is valid and budgets are maintained throughout the process.
   - Failure patterns:
     a) Budgets are exceeded during the project without compensation elsewhere.
     b) The decomposition is invalid, the nFR is not actually additive.
   - Symptom: All subsystem budgets are met, yet the system fails the nFR.

3. System-level validation – Failure through lack of correction space:
   - Core assumption: There is sufficient room at the end to correct if the total falls outside bounds.
   - Failure pattern: Problems become visible only at integration, when there is no longer time, budget, or architectural freedom to make corrections.
   - Symptom: Late surprises; system tests fail without clearly localisable cause.

Table 2
Overview of failure patterns for safeguarding nFRs.

Mechanism	Core assumption	Failure pattern	Symptom
Functionalisation	nFR fully translated	Incomplete translation	Functions work, property fails
Budgeting	Valid decomposition, budgets held	Erosion or invalid sum	Parts ok, whole not ok
System-level validation	Correction space at end	Too late discovery	Late surprise, no recovery

Diagnostic use
When a project experiences problems with an nFR, the framework provides a diagnostic path: identify which mechanism was applied, check whether its core assumption held, and match symptoms to failure patterns. This helps locate the root cause, whether the wrong mechanism was chosen, or the right mechanism was poorly executed.

Discussion

The framework was designed to address three limitations of the current situation.

1. Descriptive but not prescriptive:
   The framework addresses this by starting from the intervention question. The three safeguarding mechanisms are not categories for sorting nFRs but options for handling them. Each mechanism implies specific design activities, specific monitoring approaches, and specific failure modes. The choice of mechanism is a design decision with consequences.

2. Implicit relationship between nFR and design action:
   The decision logic makes this relationship explicit. Three questions – (i) can it be functionalised? (ii) can it be budgeted? (iii) must it be validated at system level? – connect nFR characteristics to the choice of safeguarding mechanism. This connection was previously tacit; the framework articulates it. The logic is heuristic, not algorithmic, but the questions force conscious choice where previously there was often unreflective default.

3. Missing shared language:
   The three mechanisms provide vocabulary for strategies that practitioners already employ. Functionalisation, budgeting and system-level validation are terms that describe what engineers do. This vocabulary enables teams to discuss strategies, projects to be compared, and knowledge to be transferred to new engineers.

The framework relates to several existing approaches, such as Axiomatic Design (which distinguishes FRs from constraints but does not specify how constraints are handled during design), architecture tactics (which at a lower level of abstraction introduce concrete design decisions for achieving quality attributes), ATAM and QAW (which in a later stage evaluate existing architectures against quality attributes), and taxonomies. The three safeguarding mechanisms map naturally onto established systems engineering lifecycle activities:

1. Functionalisation aligns with architecture definition and functional decomposition; the nFR is resolved early by incorporating it into the functional structure.
2. Budgeting aligns with budget allocation and design-to-cost or design-to-mass practices; the nFR is managed through progressive decomposition and tracking.
3. System-level validation aligns with integration and verification planning; the nFR is addressed through system-level test and validation activities.

This mapping suggests that the framework does not introduce new activities but provides a rationale for selecting among existing ones. The decision logic helps engineers determine which lifecycle emphasis is appropriate for a given nFR: early architectural resolution, ongoing budget management, or late-stage system validation.

As a final note, the framework has limitations that should be acknowledged: it is conceptual, not empirically validated; it is heuristic, not algorithmic; it addresses one nFR at a time; and it is domain-generic.

Conclusion

The presented framework does not introduce new practices; rather it articulates existing ones. The strategies it names are already employed by experienced engineers. Here, these have been made explicit; giving them names, ordering them systematically, and connecting them to a decision logic. It enables practitioners to name what they do. The three mechanisms offer vocabulary for strategies they likely already employ. With vocabulary comes the ability to discuss choices, document approaches, and transfer knowledge.