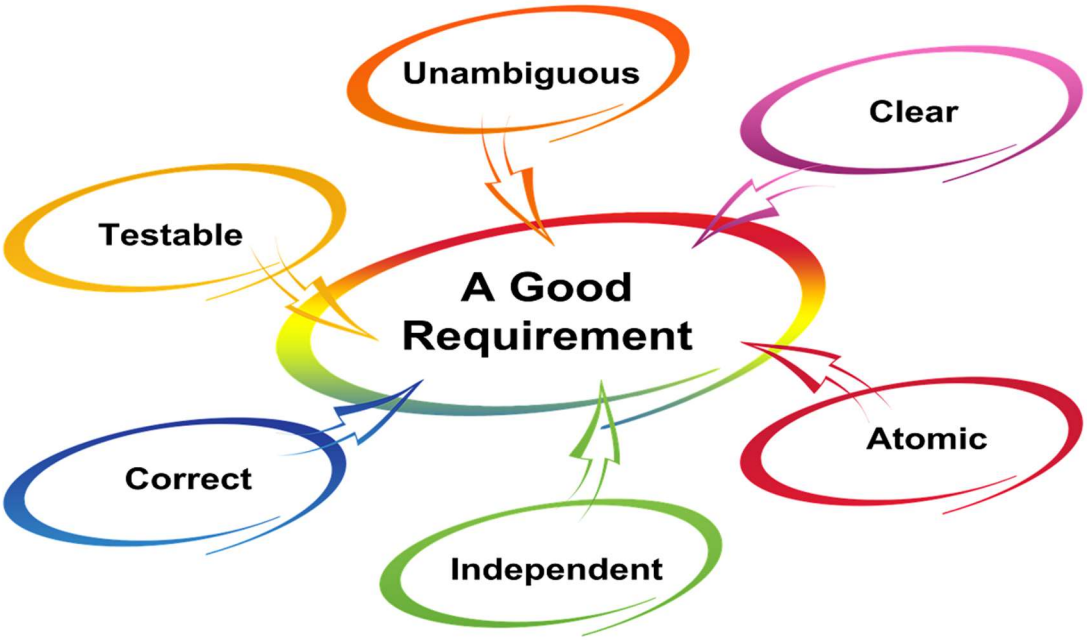
TESTABLE REQUIREMENT



The “testable” character of a requirement sounds like a pleonasm. How could it be otherwise? How should anyone come up with some requirement if he is not able to assess whether his expectations have been met? Untestable requirements are not requirements, but wishes, prayer or fantasies.

Nevertheless, all over our diversified experience, we have gone mostly through untestable requirements as soon as something beyond geometry or energy-based physics was involved, typically something dealing with business analysis, “high level” system engineering or embedded ECU specification. So, we’ll focus hereafter only on requirements that doesn’t deal with classical physics.

How can it be that organizations develop millions worth engineering system “approaches” and tools without any objective criteria to assess their actual added value and their relevancy? Beyond “hello world” illustrations, no such approach to complex development has ever come to produce a fruitful and general solution to the need of mastering complexity and of connecting design and IVVQ processes. The huge investments made to master requirements and their traceability has till now amounted to setting up Indian totems, hoping that dancing around will make the rain fall. Just totem manufacturers have found their advantage.

How can it be when anyone has been calling for “Testable” requirements for years? Two main factors may explain:

- There is a fundamental and genetic paradox that hampers System engineering from being developed as a scientific discipline (link): science is supposed to describe the real, as it is, while system engineering deals with human finalities. Such a scientific character would provide, among others, a framework to

guarantee, by construction, that any requirement be testable. Overcoming this obstacle amounts therefore to formally including human finalities in scientific corpus, at the most fundamental level.

- Many people keep on thinking that requirement engineering is a matter of common sense. “Approaches” implementing that “common sense” and coming up with magic tools must obviously make it for. It’s just a matter of “correct” application. This naïve thinking despite systematic failures persists and prevents any fundamental questioning as hereabove mentioned.

Maybe a third unmentionable factor: at a time when soft skills are the Graal to success, what does matter to some decision makers is to prove their commitment, to justify that the best is done to keep critical developments under control and, above all, to be in line with the conventional wisdom to develop their network. Tools that serve that purpose are on the roll. Actual efficiency is secondary.

For most tools, a requirement boils down to a textual paragraph, sometime loosely link to some model through dependency link. But, as already exposed in our article on “Unambiguous Requirement”[1], natural language cannot guarantee testability, by nature. Sometimes to look-like being “scientific”, some pseudo-methods go up to defining strict lexical rules, keywords and even, on such a basis, to automatically generating such models as sequence diagrams. But such rigorous look-an-feel doesn’t mean requirements are testable.

Confronted with the situation hereabove mentioned, most serious organizations try to manage a correspondence between requirements and test specification through matrixes. Most often, such a matrix formalizes a n-n relationship: a set of requirements is supposed to be assessed through a given test and several tests contribute to validate a given requirement. We cannot qualify such requirements as testable, but only the set they belong to, provided this set is endowed with an unambiguous connection to the experiment process and the resulting factual descriptions.

Even when a one-to-one association is possible between a requirement and some test, testability only relies on a consensual interpretation of the requirement with regards to the testing process. At best may we speak of “loose testability”. Factually, when some implicit or explicit consensus arises as to the tests representativity of the expected, it is eventually the integration and validation process that becomes the referential, without feedback to the “requirement design chain" for such a connection is unmaintainable.

Let’s not forget moreover that, typically in functional analysis (usually used outside the classical physic domain), no connection of “high level” requirements to facts is possible, no criteria exists to guide the functional breakdown structure. Functions float in the air without any physical referential rooting that would endow them with a testable character.

Eventually, requirements may be somehow connected to facts as mentioned only when “function” are allocated to “components” to be regarded just as software component specification... Moreover, side effects between different roles conferred to

[1] See https://www.linkedin.com/pulse/unanbiguous-requirement-henri-boulouet

the same resource come as internal “flow” that prevents any verifiable “black box” specification without interfering with that resource design. At the product level where stand the “services”, the acceptance criteria have a live of their own, relying on common usages and people expertise, far away from functional analysis.

We’ll now just introduce hereafter some features of the solution we propose with Relativized System engineering[2], based on Relativized Systemic[3] - to give a taste and, maybe, for the few ones really concerned, to get them to visit our website and, for the more adventurous, to contact us.

Requirement testability requires a formal connection with fact description. The formal structure that makes it possible (initial conditions, generative/qualification process, observables) together with the protocols that bridge the gap between factuality and the expression of the finality has already been mentioned in our past article “Unambiguous Requirement”.

Beyond, not to inextricably mix the different finalities and maintain the development process, testability calls for an organization in which each physical entity in the development architecture emerges out of the different roles it is supposed to play, each one endowed with its own “requirements”. This amounts to adopting a twofold constructive approach, one relativized, need and constraint driven - system engineering, the other resource driven - the development architecture. Then can arbitration and trade-off be mastered through testable requirements and shared resources rather than more or less imposed, as a consequence of a development process spinning out of control.

Requirements testability at any stage of the development product architecture entails to define a system as a physical entity of its own, whose design (“inner” causality) relies on a relevant composition of items in the development architecture. They may potentially be regarded as component of multiple systems. Such a system may be specified as precisely as required through testable requirements without interfering with its design, while architectural resource driven approach focuses on putting into compatibility in an optimized way the different roles resulting from the different systems design (shared resource).

Requirements themselves, within each specification, must be organized according to the lifecycle that is specific to the standpoint a system formalizes, what leads to manage several lifecycles for a given resource. They form consistent chains that turn them into the different steps of wider scope scenarios addressing the finality at stake.

The relativized nature of RSE formalism makes it possible to specify separately in a “black box” way the different roles conferred to a resource, even when these roles are somehow correlated, what used to be formalized in functional analysis with internal “flows” interfering with design.

We must still mention that testability entails to formalize the correspondence we construct between our models, stable by nature, and ever-changing factuality: there

[2] See https://mersyse.com/en/isr_features.php

[3] See https://www.amazon.fr/Syst%C3%A9mique-Relativis%C3%A9e-Essences-conceptualisations-R%C3%A9el/dp/6138478835

stand probabilities. Kolmogorov himself has put forth the impossibility to consider Probabilities, as they are defined, as a scientific basis to make the job (Kolmogorov, N.A., (1983) "Combinatorial foundations of information theory and the calculus of probabilities", Russia Mathematical Surveys, 38, pp. 29-40). In practice, they are nevertheless universally used for that! Let's just assert that we have reached that goal with the formal development of the concept of "physical probability law" (Relativized Systemic - Chapitre VIII). It introduces a conceptual and formal distinction between probabilities and statistics and opens new perspectives especially for testing and certification purposes in critical system development.

There would be so much to say... We can just here testify that Relativized System Engineering (RSE) has met the goals that have motivated its development and much beyond. Its scientific character, coming with its construction on top of a physically meaningful algebra, guarantees by construction the testable character of any requirement without "magic": RSE has been successfully validated in quite diverse operational applications dealing as well with human usages as with technical design.