

A fundamental flaw in the widely-held conventional model of requirements creates much of creep and other requirements difficulties. This flaw involves misunderstanding of the nature and role of REAL business requirements. The term “REAL” relates to requirements in two ways. The first way is widely recognizable and is represented in lower-case. People think they know what the requirements are and then learn differently and must revise their requirements definition. Thus, the “real” requirements are what one ends up with, as opposed to what one may have thought initially.

The second use of “REAL” warrants distinguishing with upper case because it represents breakthrough awareness that ***REAL requirements are business requirements, which are in business terms and are what must be delivered to provide value.***

Conventional Model

Common usage and practically every requirements book, typically in about the first paragraph, characterize “requirements” as the requirements of the product or system [which is to be created].

Therefore, for the purpose of this discussion, I’m going to refer to these as *product or system requirements, or software requirements*, which I’d suggest is an equally interchangeable term, since often the product or system being created consists largely of software. While some organizations do make distinctions between the terms “system requirements,” for products which don’t include software, and “software requirements” for products that do include software, for these purposes I think you’ll see that such a distinction is not relevant. Certainly, many people use the term “requirements” to mean software requirements and aren’t necessarily limiting themselves to software even when applying formal definitions such as the widely-used IEEE Standard 830 for Software Requirements Specifications.

Such product/system/software requirements often are stated in sentences beginning with the words, “The system shall” Frequently, such statements also are called *functional requirements, or functional specifications*, since they describe the functioning of the system from the perspective of an external observer. Authorities and standards recognize the need too to identify *nonfunctional requirements* of the product or system, such as performance and usability, which often also are called quality factors, attributes, or “ilities.”

Increasingly, organizations capture requirements in *use cases*, which are typically defined as “how an actor interacts with the system.” An actor usually is the user but could also be another system or piece of equipment. The heart of a use case is a step-by-step description, from the perspective of the actor, of the actor’s actions and the system’s responses. Because use cases are fairly concrete and understandable, they generally aid

communication and increase chances the developer will implement a system that works the way the user expects. To many, use cases are the *user's requirements*; and the user's requirements are most appropriately documented in use cases.

It also is commonly held that *creep*, changes to the (product/system/software) requirements after they ostensibly had been agreed upon, is primarily due to said system requirements *lacking clarity*.

In the testing community, the emphasis is almost entirely on the related concept of *testability*, which is the ability to create test cases that demonstrate the requirement has been satisfied in the delivered system. If a requirement is not testable, it's usually because the requirement is not sufficiently clear for the developer to know how to satisfy the requirement, which in turn creates rework and further requirements definition creep; and regardless, without suitable tests, there will be no way to confirm whether the developer's solution in fact has satisfied the requirement. One valuable by-product benefit of use cases is that they translate relatively directly into test cases. That is, a use case is demonstrated by carrying it out; and the use case's action-response format resembles a test case's input and expected result.

Many in the requirements community also use the term "business requirements," or something similar, which are widely accepted as being generally-vague, high-level requirements that decompose into the more detailed product/system/software requirements. People commonly consider the difference between business and system requirements to be simply a level of detail. That is, if a requirement is high-level and vague, it is a business requirement, whereas if the requirement is detailed, it is a system requirement.

While "high-level" can be interpreted in a variety of ways, many seem to use the term in the manner described by several authorities, including the International Institute of Business Analysis (IIBA) Business Analysis Body of Knowledge (BABOK, section 1.6 2.1.1, page 18), that business requirements are "statements of the goals, objectives, or needs of the enterprise."

REAL Business Requirements

REAL requirements are business requirements, which are in business terms and are what must be delivered to provide value.

This definition contains several important distinctions from the conventional model. The failure to recognize these distinctions perpetuates many of the requirements problems that people continue to experience and often therefore erroneously assume are unavoidable.

The conventional model says the desired benefits (objectives) are the business requirements. That's a certain recipe for creep because objectives are not requirements. Everyone can agree upon objectives and yet have no common understanding of what must be delivered to achieve those objectives. Rather, REAL business requirements are the deliverable business *whats* that when delivered (or satisfied, met, or accomplished) reasonably will accomplish the objectives and thereby provide value.

I think you'll find that common usage of the term "user requirements" also is mistaken and leads one's thinking astray. The use case format indeed can be applied to describe the user's requirements. However, as commonly applied, use cases almost always describe not the *user's* requirements but the *usage* requirements of a product, system, or software that the development process intends to create.

It's more appropriate to treat the terms "business requirements" and "user requirements" as synonymous. That is, business/user requirements are from the perspective and in the language of the business/user. The user's requirements are to do their business, which can be personal or organizational, involving money or not. Business/user requirements *exist* within the business/user environment and must be discovered. Business/user requirements serve business/user objectives and provide value by solving problems, taking opportunities (which generally solve someone else's problem), and meeting challenges. Usually there are many possible ways to accomplish the REAL, business requirements (henceforth, I'll save a little complexity by referring to these simply as "business requirements").

System requirements represent a *human-defined* product or system which presumably is one of the possible ways presumably *how* to accomplish the presumed REAL business requirements. As such, system requirements actually are a form of high-level design. The term "specification," which often is used interchangeably with "requirements," refers to human-defined design. Implementing a product or system that satisfies its system requirements does not of itself provide value. Value is provided if and only if satisfying the system requirements in turn satisfies the REAL, business requirements. That's why the word "presume" is so emphasized, because such presumptions often turn out to be wrong.

While clarity of the system requirements is important, *much of creep occurs* regardless of how clear the system requirements are, *because the system requirements don't satisfy the REAL, business requirements*. The primary reason products/systems don't satisfy the REAL business requirements is because the REAL business requirements have not been defined adequately. In turn, the primary reason the REAL business requirements are not defined adequately is because conventional requirements models mislead people into believing that system requirements are *the* requirements and thus warrant all the attention.

The difference between REAL business requirements and system requirements is not the level of detail. The difference is qualitative. Business requirements are *whats*. System requirements are design *hows*. Design doesn't have to be technical. A product or system is a presumed solution *how* to accomplish the presumed *whats*. *Whats* do not decompose into *hows*. *Hows* are a response to *whats*. All the *how* detail in the world won't make up for not knowing the *what* detail; and consequently focusing on all that system requirements *how* detail without knowing the REAL business requirements *what* detail is essentially guessing, which can't help being wrong and turning into creep. The conventional requirements model misses the boat by emphasizing attention to the system requirements *how* detail while failing to recognize that the REAL, business requirements *whats* need to be defined in detail first.

When REAL, business requirements are defined in detail first, they usually map fairly straightforwardly to a product/system which in fact will meet the REAL business requirements and thereby provide value.

But, Requirements Change So Much...

When people's only context is the conventional requirements model, they often easily leap to unwarranted conclusions about difficulties discovering the REAL business requirements detail first.

One such common conclusion is that time spent defining requirements early is wasted because everyone knows it's impossible to get the requirements right anyhow. Practically any experienced developer will assure you that requirements have to be revised repeatedly to respond to rapidly changing business situations and users who don't know what they want until they see something implemented. Let me suggest that for many, this becomes a self-fulfilling prophecy.

The fact is that REAL business requirements do change, but not nearly so much as people presume. It's not the REAL requirements that change so much, but rather it's the *awareness* of the REAL requirements that changes.

I agree there is little value in exerting effort capturing requirements which seem to have changed by the time the ink is dry. I also agree that some organizations have found ways to turn the documentation of detail into a seemingly interminable activity with a life of its own. I would suggest that each of these is a largely unnecessary, but predictably likely, undesirable outcome of the conventional model's focusing on the system requirements *hows* without first discovering the REAL business requirements *whats*.

The excessive requirements changes that people are accustomed to are changes to system requirements which have been guessed due to inadequate understanding of the REAL business requirements. Time spent laboriously documenting these *hows* indeed is often

wasted. Then, in a most inefficient manner, through successive labor-intensive code implementation iterations, the conventional approach eventually backs into a system which reasonably satisfies the underlying but probably still-not-articulated REAL business requirements.

Iteration indeed is the key to rapid delivery of value, but it's a different form of iteration based on the hierarchical nature of REAL business requirements. Nobody is saying that it's possible, let alone worthwhile, to define all the requirements *detail* perfectly up-front. On the other hand, it is both feasible and essential to discover the full set of truly *top-level* business requirements deliverable *whats* before proceeding to design and implementation.

We use the powerful Problem Pyramid™ tool to guide systematically disciplined, reliable rapid iterative discovery of top-level REAL business requirements. Iterations include applying special approaches and methods, both for discovering the REAL business requirements and for evaluating their accuracy and completeness. Many of the more than 21 ways we've identified for reviewing requirements go well beyond the conventional attention to clarity and testability. Realize that an incorrect requirement can be perfectly clear and testable; and clarity and testability are irrelevant with respect to a requirement that has been overlooked. These special techniques help identify ordinarily-overlooked and incorrect business requirements, as well as ones lacking testability and clarity.

Iterations continue until key stakeholder review, following clear evaluation guidelines, determines the Problem Pyramid™ is right and identifies the appropriate set of top-level business requirements that provide REAL value. With this grounding in value, estimates and tradeoff analysis become meaningful. Typically the full set of top-level business requirements is allocated to respective prioritized implementation projects.

Only those top-level business requirements which are to be implemented will be elaborated iteratively into detailed business requirements. Iterations usually will go down two or three levels at a time and continually apply various of the 21+ review techniques to assure accuracy and completeness. Then, system requirements and use cases are defined for a product/system that meets the detailed business requirements

It's really much quicker, cheaper, and more satisfying to learn how to discover the REAL business requirements in the first place and then design a product/system which indeed will satisfy them and provide value. It takes some different approaches and modification of familiar techniques, because the focus needs to be on discovering what the business needs rather than expecting the business people to design a product/system solution.

My book, *Discovering REAL Business Requirements for Software Project Success*, and related seminars (*Defining and Managing User Requirements* and *Evaluating Requirements and Design Adequacy*) further explain these approaches and techniques.

Taken together these methods enable dramatically improving development time, cost, and quality by discovering the REAL business requirements that prevent creep.

About the Author



Robin F. Goldsmith, JD

Robin F. Goldsmith has been President of Go Pro Management, Inc. consultancy since 1982. He works directly with and trains professionals in business engineering, requirements analysis, software acquisition, project management, quality and testing. He partners with ProveIT.net in providing ROI Value Modeling™ tools, training, and advisory services. He is author and Course Director for ASPE's two-day seminar, *Managing Real-World Processes and Projects with Metrics*.

Previously he was a developer, systems programmer/DBA/QA, and project leader with the City of Cleveland, leading financial institutions, and a "Big 4" consulting firm.

Author of numerous articles and the recent Artech House book *Discovering REAL Business Requirements for Software Project Success*, and a frequent speaker at leading professional conferences, he was formerly International Vice President of the Association for Systems Management and Executive Editor of the *Journal of Systems Management*. He was Founding Chairman of the New England Center for Organizational Effectiveness. He belongs to the Boston SPIN and served on the SEPG'95 Planning and Program Committees.

Mr. Goldsmith Chaired BOSCON 2000 and 2001, ASQ Boston Section's Annual Quality Conferences, and is a member of the ASQ Software Division Methods Committee and the IEEE Software Test Documentation Std. 829 revision Committee.

He holds the following degrees: Kenyon College, A.B. with Honors in Psychology; Pennsylvania State University, M.S. in Psychology; Suffolk University, J.D.; Boston University, LL.M. in Tax Law. Mr. Goldsmith is a member of the Massachusetts Bar and licensed to practice law in Massachusetts.