Down With "Requirements"!

A plea from John Daniels, Syntropy Limited.

I really hate the word "requirements". We have got to the point now where it is an almost totally meaningless word when applied to software development. When someone offers to tell you the "requirements" for the software you're about to create it's anyone's guess what you're going to get.

Think for a moment about the old-fashioned, and much maligned, "Functional Specification". You knew where you stood with one of those. It was the developer's proposal for how the system would meet the "requirements". In many cases those requirements were never formally captured. The client would assess the Functional Specification and decide informally whether it met the requirements. But note that the Functional Specification **wasn't the requirements**, and was never claimed to be – it was a specification. In my view not much has changed really, despite all the words written about "Requirements Engineering".

Different kinds of requirements

In most projects the requirements process and work products are a total mess. There will be no clear distinctions between levels or kinds of information, and the notations used to record it will be inconsistent and imprecise.

In an ideal project you'd find the following information, all of which is often labelled as "requirements".

1. **Business goals**. If you are very lucky, you might be told what the business goals are. A goal here is some sort of attainment for the business, such as "become the leading online book retailer" or "reduce our insurance policy management costs by 50%".
2. More likely (but still not very likely), you will be told the **strategies** the business wants to adopt to meet its goals, such as "start selling books" (!) or "let the customer browse an online catalogue and place book orders directly" or "provide a mechanism by which customers can view their policies and request changes without having to phone the call centre".
3. When strategies are elaborated they become **business processes**, showing the steps or activities involved in executing the strategies. In theory business processes can be described without deciding how technology supports, or takes responsibility for, the various activities, but in practice these decisions are usually taken quickly because of the pervasive nature of software nowadays. Many different software systems might be involved in supporting a single business process.
4. Next up are **system interaction** specifications. At this point the boundary between human and machine responsibility must be decided, and responsibility for information provision and automation allocated across the systems involved. A system interaction tells you how external actors, including other systems, will interact with the system under consideration to achieve the result defined in a process activity. If this is a human-computer interaction then it might involve some complexity, worthy of a storyboard-like exposition. If the activity is simply triggered by the arrival of some information from another system then the actual interaction will probably be

trivial. But in all cases the interaction should specify, in terms of an information model, the before and after states of the system and the business rules being applied. Most of the so-called "non-functional requirements" are rooted in this category, in the form of numbers of interactions, limits on response times, and the likelihood that a stimulus will actually get the defined response.

5. Finally there are the **technical constraints**, such as "you must use J2EE" or "no changes can be made to the mainframe software". What do you mean, "those aren't requirements"? If I fail to heed them the result may not be deemed successful – that sounds like a requirement to me.

In fact, the "requirements" churned out in Use Cases and other UML artefacts are nearly always only in categories 3 and 4, and even then the "non-functional requirements" will be sketchy or absent. So for most practical purposes these days, the word "requirements" still translates to "functional specification", a dubious piece of double-speak if ever I heard one. The irony here is that, from the business perspective, the requirements which drive the whole project are in category 1, and possibly category 2.

Who writes them?

Who should be responsible for creating the five different kinds of "requirements" listed above? A useful way of resolving this is to think of the various kinds of requirements as contracts between different roles in the organisation (although one person may play many roles). Each kind of requirement has a "client" – the role that wants the result – and a "supplier" – the role that is obliged to deliver the result. Bear in mind, though, that the supplier will invariable require inputs from many sources – the supplier doesn't do it all themselves.

Contract	**Client**	**Supplier**
1. Business goals	Shareholders	Board
2. Business strategies	Board	Business strategists
3. Business processes	Business strategists	Business process designers
4. System interactions	Business process designers	Business analysts
5. Technical constraints	Corporate IT	Software designers

"Gosh", I can hear you saying, "what a lot of roles, and we've only considered requirements!" First, bear in mind that these are roles, not necessarily people (but more on that later). Second, these roles really do require particular skills and viewpoints.

A business strategist is someone who can figure out how a goal can be met. They need a deep understanding of the business domain, and a broad vision for how the business will operate. A business process designer can formulate end-to-end processes that deliver on the strategies. He or she will have the obsessive attention to detail that often isn't found in strategists, and an ability to model, which implies abstraction. A business analyst shares that attention to detail and modelling ability, understands the business domain or can learn it quickly, and, most importantly, can communicate effectively between technical and business people.

I think it's clear that business goals (category 1) are – or should be – set by the board members of the business. But what about business strategies, business processes and system interactions (categories 2, 3 and 4)?

Many popular modern development processes, including eXtreme Programming (XP), rightly stress the importance of a close relationship between the customer of the software and its developers. The separate roles of "business strategist", "business process designer" or "business analyst" are rarely acknowledged, so the strategies, processes and system interactions have to come from the "customer", goaded by the developer. The trouble is that most of us in the IT industry are neither trained, experienced or, in many cases, interested, in helping a company with its business strategies, nor can it be assumed we recognise a good business process or system interaction when we see one. An additional problem is that the "customer" in XP is frequently really an end user of the software, and not necessarily someone fully in tune with the business goals. This does, though, have the clear benefit that the defined processes and system interactions are at least likely to be practical and useable.

Technical constraints are set by the corporate IT function ("Enterprise Architects") who, rightly or wrongly, take it upon themselves to put in place standards and policies that they hope will lead to technical convergence, streamlining and, ultimately, reduced technical risk. Having worked in that role myself, I'll simply say that such things are very hard to do well, and developers are well advised to push back strongly against constraints they feel will significantly hinder their progress.

Notations

What form should these five contracts take? I think it's fair to say that the UML has nothing to say about representing business goals or strategies; these are usually delivered as documents and spreadsheets, as are technical constraints. Business processes are best represented using process modelling notations, such as UML activity diagrams. The best representation for system interactions is less certain.

Some people like to represent system interactions with Use Cases, which when based on templates that include pre- and post-conditions, and backed up with supporting business rules documents and, possibly, storyboards, can certainly do the job. Others prefer to avoid the tension between the procedural steps of Use Cases and the declarative nature of pre- and post-conditions by adopting a purely declarative style, perhaps using a language such as OCL, and writing "OpSpecs" (Operation Specifications). Personally I quite like Use Cases when used properly, but overall I'd say they have been a force for evil rather than good in software projects. Use Cases are more often than not imprecise, unclear, not integrated with other models, and hazily positioned in the development process.

A radical and intriguing alternative is to represent system interactions as executable system tests, where the implication is that the system is required to do whatever is necessary for the test to execute correctly. I predict that this kind of test-driven development will become very popular because it makes "requirements" come alive. It correctly focuses attention on stating fully and clearly what is required, and then directly implementing against that specification.

Roles and people

Now let me return briefly to the question of roles versus people. The old-fashioned Functional Specification was traditionally written by technologists because they

understood what was – and wasn't – possible with the technology. But they relied heavily on domain knowledge quickly acquired or provided by others. The notion of Functional Specifications as the primary requirements artefact harks back to days when systems tended to be smaller and simpler, and the industry was populated by generalists – people who could turn their hand to most tasks in a development project.

After a decade or two of trying but failing to understand the specialised roles needed in software development, and also failing to understand how they should fit together, recent trends in development process have gone the other way, stressing instead the need for each team member to have knowledge of the entire system and to be able to take on pretty much any task. XP is a good example of this. Each XP team member is taking on several roles, but the specific roles are not made explicit.

This may be pragmatic and practical but it doesn't scale. For larger undertakings it is essential that we define roles explicitly and understand how a large team with defined roles can work together effectively. In my view the future of software development, as has been the case in other creative fields, lies with specialists, not generalists.

For me the term "requirements" is a red herring. Instead, there is the set of things about which I, as software developer, have no choice – the givens – and those things about which I am allowed to make an informed proposal. Wherever I have a choice I'm doing design, wherever I don't I'm meeting constraints, whether they be technical or business strategic.

So please don't bandy around the word "requirements" as if its meaning is clear to all. Instead, focus on the five contracts described above – then perhaps we software designers can all get on with building the right system.

I'd like to thank Bruce Anderson, Dave Cleal, Jason Gorman and Rob James for their help in preparing this article. However, any screw-ups are entirely my responsibility.