Software Requirement Patterns

1 Introduction: Better, Quicker and Easier Requirements

This is the first of two articles that describe two approaches to producing better requirements, more quickly and with less effort. This article introduces **Software Requirement Patterns**, which provide guidance on how to write specific types of requirements. The second article, "**Products and Requirements**", discusses how we can make better use of requirements when acquiring software products (and when selling them too)—and it suggests ways we can help each other by sharing requirements and related information.

Every time we define the requirements for a new system, we start from scratch. Well, almost. We usually figure out all the functions and write every single requirement afresh. It's certainly attractive to avoid pre-conceived ideas and to begin without unnecessary baggage, but we do seem intent on doing as much work as possible. Reinventing the wheel takes longer, provides more opportunities to make mistakes, and business analysts must be more skilled than they would otherwise need to be. It also suggests that business analysts could spend more of their time on challenging, interesting areas and less on the mundane and tedious. Maybe all this work creates employment for lots of business analysts, but I'm more inclined to feel it reduces respect for the profession and for requirements in general—and therefore has the opposite net effect.

I'm not decrying the need for better-qualified business analysts, but finding ways to avoid much of this labor altogether is likely to have greater impact. After all, it takes great dedication and pain for an athlete to shave even a fraction off their personal best—whereas running a shorter distance delivers dramatic improvements immediately. (I presume there are no psycho*business*analysis drugs that will help!) This isn't cheating, because we're not in a contest with rules, so we ought to use every means available to avoid slogging the whole distance on foot. But what means are available? I'd like to present two, in this article and a second to follow, although there are no doubt more. I suggest that early in every requirements process you ask yourself, "What resources might exist to save specifying all the requirements from scratch?"

2 The Purpose of this Article

The purpose of this article is to describe how requirement patterns can assist you to write specific types of requirements that you are likely to encounter in all kinds of system. It is perhaps a surprise to discover that a high percentage of the features found in any commercial system are common to any kind of system, and that only a relatively small part differentiates systems for very different purposes and industries. In some of the requirements specifications I studied, the patterns I identified are applicable to more than half of the requirements. This article is based on my book "Software Requirement Patterns" (Microsoft Press, 2007). Oh dear, you might well say, this won't be any use to me unless I buy the book. Of course I can't compress a whole book into a short article, but enough resources are freely available to get you started, including all the example requirements and requirement templates in the book. They can be downloaded from:

http://www.microsoft.com/mspress/companion/9780735623989

And you can download a sample requirement pattern, for scalability, from:

http://www.withallyourequire.com

3 What is a Requirement Pattern?

A requirement pattern is a guide to writing a particular type of requirement. It explains how to tackle that type of requirement, what to say, what to worry about, and it suggests additional requirements you might need to write too. Don't be put off by the word "pattern", because there's nothing mysterious or technical involved. You engage a requirement pattern when you want to write a single requirement, so each pattern contains detailed and precise low-level information. A pattern doesn't worry about the big picture of a whole system. Once you've used a pattern to help you write a requirement, its job is done. Then you can put it away and move on to the next requirement.

The aim of requirement patterns is to enable you to write higher quality requirements more quickly and with less effort—though I don't want them to sound like a miracle pill. They achieve these benefits by letting you take advantage of all the effort that went into writing them, because of the simple equation that says it's possible to devote considerably more time and thought to developing a pattern than you could ever afford to expend on writing a single requirement. So a requirement pattern represents *encapsulated expertise* that you can call on whenever you need it.

Requirement patterns help you write *better* requirements because they can point out all the issues you ought to bear in mind, so you don't overlook things. And a lot of care can be taken with the phrasing of examples and templates. Patterns let you write requirements *quicker* and *easier* by handing you everything on a plate: you begin with a substantially-written requirement, rather than having to start from scratch. Some patterns (especially the complex ones) give you step-by-step instructions to follow to gather the information you need. Occasionally a requirement pattern can try to steer you away from an obvious (but bad—or potentially bad) way of specifying a requirement, and encourage you to approach it in a different way. I call these **divertive patterns**. In fact, a requirement pattern is able to contain *any* kind of background information or advice or references to other resources that might be of interest to someone who's mulling over a requirement of this type.

I caution, however, that no matter what means you might find to make writing requirements easier, it is still *your* responsibility to take care that they express what they need to. What is a good requirement in one situation might not be good in *your* situation. Don't use requirement patterns—or, say, lists of requirements that you are tempted to copy and paste—as an excuse to be lazy.

Pervasive Requirements

Formulating requirement patterns involves figuring out the best way to specify certain types of requirements. This includes keeping an eye open for opportunities to make requirements more efficient, such as avoiding repetition (which involves more work—both to write and to read—and can lead to inconsistencies). One possibility that I found cropping up in several places is what I call **pervasive requirements**, which is where a single requirement imposes a systemwide demand on everything that it applies to, such as all functions of a particular type (or *all* functions). This saves you having to mention it every time: you need define it only once. This removes any possibility of inconsistency (that is, different requirements asking for it differently), allows you to define it thoroughly in that one place, and any change to it is made only once.

Pervasive requirements have nothing to do with requirement patterns per se. You can use them in

any requirements specification. If you find you're saying the same thing over and over in many requirements, stop and consider whether you can extract the repeated part into its own requirement. If so, work out how extensively you need to apply it across the system, and impose it that widely (but no wider than is worthwhile, in case it adds to the cost of implementing it).

Take care, though, that everyone's aware of all the pervasive requirements that affect them. For example, a developer implementing one requirement might not read the whole requirements specification, so they might miss a critical pervasive requirement. So make them as visible as possible, and publicize them.

4 How To Write Requirements Using Patterns

First know what patterns are available (they're introduced in the final section of this article), because they represent tools you can call upon—and every skilled artisan ought to know what tools are at hand, and when to use each one. Then when you want help to write any of these types of requirements, you can pull out the relevant pattern. Each requirement pattern contains an "**Applicability**" section that describes the situations in which that pattern can be applied—and can also point out situations when it's not appropriate to use it. You should read through the "Applicability" section of every pattern, as part of getting to know the tools available to you. Like any set of tools, you're likely to use some frequently and others rarely—but if you use a pattern only once, on that occasion it might prove invaluable.

Each requirement pattern contains one or more **templates**, and several **example requirements**. A template is a fill-in-the-blanks starting point for a requirement which identifies each item of information a requirement of this type might need (including optional items that can sometimes be omitted). The simplest way to use a requirement pattern is to copy-and-paste a template or example requirement and then modify it to say what you want. (The downloadable resources contain enough for you to get this far.) At the very least, this will be quicker than writing the whole of the requirement's definition yourself.

Templates and example requirements are of limited value on their own. The real engine of a requirement pattern is its "**Discussion**" and "**Content**" sections. The "Discussion" section explains how to tackle this type of requirement and how to gather information for it—going as far as providing extensive step-by-step processes, if that's what it takes. The "Discussion" can also contain any other material that could be of interest to someone writing one of these requirements. The "Content" section supports the requirement templates, by describing in detail what items of information a requirement of this type ought to contain.

Sometimes writing a single requirement isn't enough to express all that needs to be said. This is particularly true for complex areas (such as aspects of performance and flexibility), where you might start with a high level requirement that deserves to be broken down to spell out its implications. To deal with these situations, each requirement pattern contains an "**Extra Requirements**" section that runs through every type of extra requirement you might wish to consider; in a few cases there are dozens of them. Practical example requirements are given for each type of extra requirement, which you can copy in and use as the basis for your own requirements.

5 Other Ways to Use Requirement Patterns

Writing a requirement is only one stage in its long and (hopefully) fruitful life. Requirement patterns can help in other stages too. The same guidance that a pattern offers to help write requirements can be employed when interpreting them. Anyone who **reviews** a requirements specification can use requirement patterns to gauge the quality of particular requirements. Do they say all they need to? Have important auxiliary requirements been missed?

Requirement patterns can also potentially be used to help in **estimating** the size of a system and the effort it'll take to develop it. Using metrics calculated from previous projects for particular types of requirements is a sounder basis for estimation than the raw number of requirements.

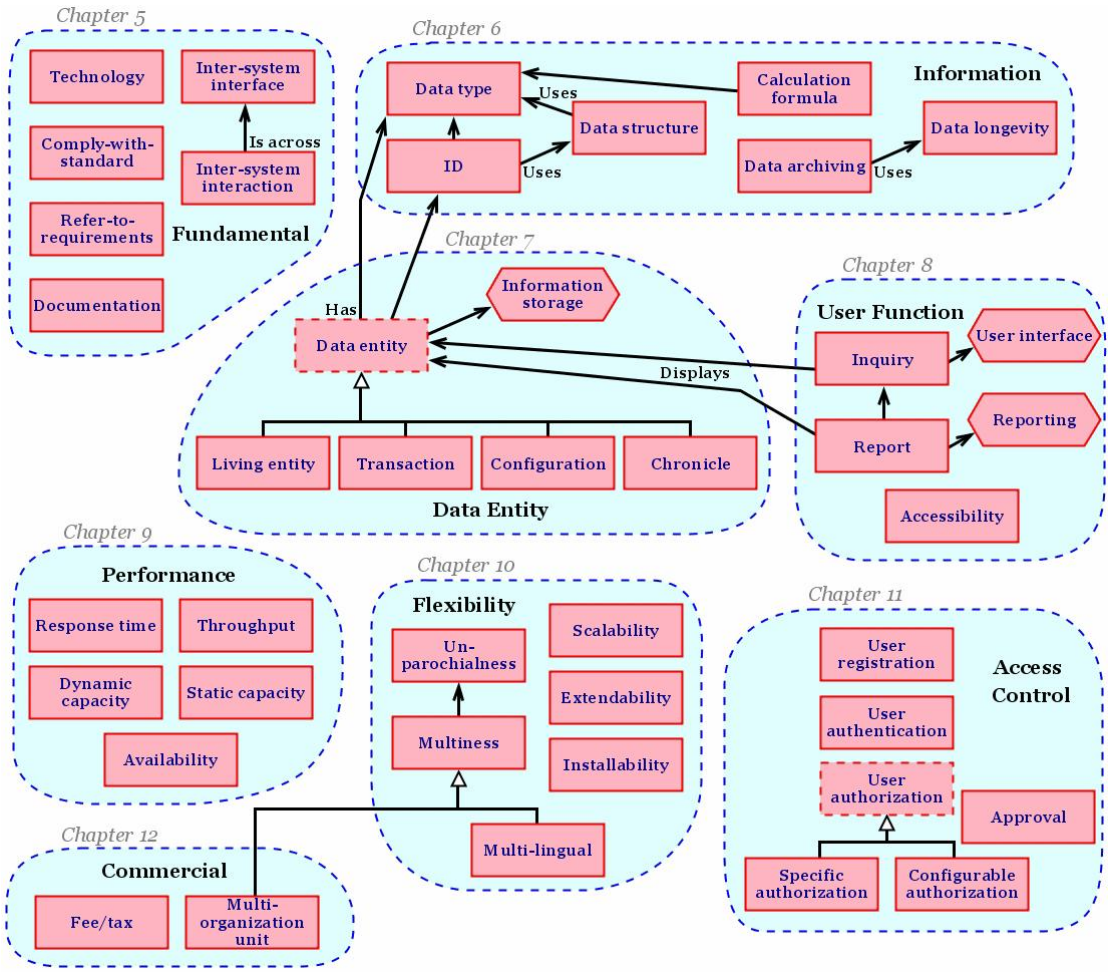
Each requirement pattern has a "**Considerations for Development**" section aimed at software designers and developers. It contains suggestions on how to implement a requirement of this type, and tips on issues to consider. Sometimes it also raises questions that developers should consider when reviewing requirements, to avoid impractical-to-implement requirements being written in the first place. Similarly, a "**Considerations for Testing**" section is present in each requirement pattern for the benefit of testers, to suggest how to test requirements of this type and to point out factors to look for that might be hard to test.

6 What Requirement Patterns Are Available?

There are thirty-seven requirement patterns in the "Software Requirement Patterns"book, for types of requirements that crop up in all kinds of systems. They are shown in the diagram on the following page. The material available for download (from http://www.microsoft.com/mspress/companion/9780735623989) includes a ready reference document that lists the "Applicability" of each pattern. I found that in practice these patterns sometimes apply to more than half of the requirements in a specification. However, it's hard to measure overall value from simple statistics like these, because help in getting one tricky requirement right might be of immeasurable benefit—while at the same time there will always be many unique, one-off requirements for which no pattern exists and there is no justification to write one.

These patterns were identified by dissecting requirements specifications for various kinds of commercial systems—although, as in any thoughtful analysis process, not taking them directly at face value. Some candidate patterns were dropped along the way because there wasn't enough of value to say about them (for example, if there was too much variety between one requirement and another), or because they were too obscure.

Studying requirements specifications in this way was an eye-opening experience: I found them to be of generally poorer quality than I expected (even ones I'd written myself that I was quite proud of!). So simply distilling and standardizing what I found in existing requirements wasn't enough: I sought ways to do better, such as ways to avoid the common mistakes I encountered. As a result, one or two surprising patterns popped out, especially for flexibility-related requirements. And I took a fresh look at requirements that define the information in a system, which led to separate patterns for four different types of information.



This diagram's notation is quite straightforward. The rectangles are patterns. (You can ignore everything else if you find it too complicated.) Lines denote relationships between patterns: triangles show where one pattern builds upon (or *extends* another), and arrows are for other kinds of references (though only the most important are shown; otherwise there would be arrows everywhere). Rectangles with dotted lines are "base" patterns, which are basically just a way to avoid repeating descriptions in multiple patterns. Lozenges (hexagons) denote infrastructures that need to be present to support the types of requirements that refer to them (for example, a database is the commonest type of information storage infrastructure, which you need if you have any requirements for data entities). The large regions represent groupings of related patterns.

I would like to see more requirement patterns become available—especially domain-specific patterns (for banking or accounting, say), and patterns in specialized fields such as security (in which it is hard for non-experts to write good requirements). I say a little more about this in my second article, "Products and Requirements", because domain-specific requirement patterns could improve communication between product vendors and their customers.