

Software Product Lines

What's the Difference Between Product Line Scope and Product Line Requirements?

Paul Clements

To set the scope for a product line, decision makers consider what each system must be able to do to be part of the product line. Requirements tell engineers how a system will behave. Though scoping and requirements engineering are similar activities, it is important to realize that they are done by different people, stop at different points, and are used for different purposes by different stakeholders.

Setting the proper scope of the product line is a step of fundamental importance. Make it too large and the core assets will have to be too hopelessly generic to ever work; make it too small, and the market demand for your product suite will be too small to recover the up-front investment. Make it the right size but encompassing the wrong systems for your market and you'll have no customers.

What does the representation of a product line scope look like, exactly? That is, how do you write one down? It can be very vague, or very precise—any statement that will let a decision-maker decide whether a proposed product is in or out will do. In practice, to make such a decision requires a fairly precise statement of what the “in” systems will all have in common. One product line scope definition that we once saw talked about systems for special-operations helicopters. These systems were supposed to do three things: “aviate” (that is, fly), “navigate” (move from point A to point B), and “communicate” (talk to other systems). While this description ruled out, say, software for toasters, it is clearly insufficient to tell whether the software for another kind of aircraft would be in or out of the product line. Something more detailed was needed. During the next round, some specific behaviors and features were articulated, and this began to position the product line squarely in the realm of special-operations helicopters.

This is usually how it goes. Scope starts out as a vague description of a set of systems by naming some functions they provide. After that, then it's refined to be written in terms of the products' observable behaviors and their exhibited quality attributes such as performance or security.

But that is also how writing down requirements for a system goes. So why are the scope and the requirements covered in two separate practice areas? Aren't they in fact the same activity? The answer is “Theoretically, yes.” But “theoretically” is often a euphemism for “not really.”

In part a of Figure 1, the rectangle represents every possible software system that ever has been, ever will be, or ever could be built. This is the starting point, albeit a not very useful one, for determining the scope of a product line. Typically, a product line manager is able to start describing systems that are definitely outside the product line (toasters) and a few products that are definitely in (a small list of specific special-operations helicopters). Part b of Figure 1 shows the system space divided into three parts: systems that are out (mottled), systems that are in (white), and systems we aren't sure about yet (black). The process of defining the product line scope is the process of narrowing down the “not sure about” space by carefully defining more of the “out” and “in” spaces, until conceptually it resembles part c.

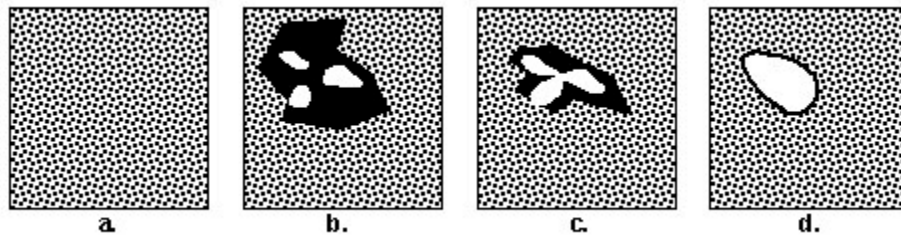


Figure 1: The Evolution of a Product Line Scope

Now think about requirements. A good requirements specification should tell us everything that must be true about a system for it to be acceptable, and no more. The “no more” part means that many systems could be built to satisfy a given requirements specification, since a statement of requirements does not and should not mandate a particular implementation. Requirements for a product line specify things that are true about every system in the family, and specify a set of allowable variations exhibited by individual systems. Given any system, we should be able to compare it to the requirements specification and see whether or not it conforms—that is, whether it satisfies all of the common requirements and an allowable combination of the variable requirements. It either does or it doesn't: There is no in between. Part d of Figure 1 is the visual rendition of this situation, which is just part c pursued to the point where the “not sure about” space has been squeezed to nothingness. This is why we said that requirements engineering was theoretically akin to product line scoping—a completely precise scope is, in fact, a requirements specification for the product line.

At least theoretically. Often, the scope includes aspects of the system that would not appear in even the most complete requirements specification, aspects related to business goals or construction constraints. For example, “any reasonable system commissioned by our most important customer” might appear in a scope definition as something you're willing to build, but not in any requirements spec because it's not a testable condition of the software.

In addition, the scope and the requirements are defined at different times. The scope is defined early enough so that a business case can be built and used to see if the product line is economically viable. The requirements are specified as a prelude to actual development. The two products have different consumers. The scope is written for people like marketers, who need to see what they will be asked to sell but do not require full statements of product behavior. The requirements spec is used by the architect and the developers of core assets and products who *do* need to know exact behavior. (The scope definition is also used by the architect to begin planning architectural means to provide the commonality and variability defined there.) Finally, there is no mandate for the scope to be completely precise—the situation in part c of Figure 1 is just fine. The vast majority of systems are ruled out, a useful number of systems are ruled in, and a class of systems remains on the cusp, meaning “If asked to build one of those, we’ll think about it.” If, over time, you’re asked to build a *lot* of those, it may be a sign that your scope missed the target a bit, and needs to be adjusted.

In fact, the situation in part c is *more* desirable for a scope than the one in part d. If you’re asked to build a system that lies oh-so-close but just outside the anointed set of “in” systems, then “I’ll think about it” is probably a better response than a flat “No.” Rejecting it out of hand might cause you to miss a good business opportunity that you had not previously considered.

So if you thought that setting the scope and setting the product line requirements sounded like similar activities, you’re right. They are. In practice, however, scoping and requirements engineering are done by different people, stop at different points, and are used for different purposes by different stakeholders.

The views expressed in this article are the author’s only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

® Capability Maturity Model, Capability Maturity Modeling, Carnegie Mellon, CERT, CERT Coordination Center, CMM, CMMI, and OCTAVE are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

SM Architecture Tradeoff Analysis Method; ATAM; CMM Integration; COTS Usage Risk Evaluation; CURE; EPIC; Evolutionary Process for Integrating COTS Based Systems; Framework for Software Product Line Practice; IDEAL; Interim Profile; OAR; Operationally Critical Threat, Asset, and Vulnerability Evaluation; Options Analysis for Reengineering; Personal Software Process; PLTP; Product Line Technical Probe; PSP; SCAMPI; SCAMPI Lead Assessor; SCAMPI Lead Appraiser; SCE; SEI; SEI-Europe; SEPG; Team Software Process; and TSP are service marks of Carnegie Mellon University.

TM Simplex is a trademark of Carnegie Mellon University.