Neil Maiden’s column “User Requirements and System Requirements” (IEEE Software, March/April 2008) makes the case that while deriving the latter from the former is a design activity, it isn’t generally recognized as such. I was left with the impression that the very term “system requirements” is part of the problem: If the presence of the word “requirements” leads people to conflate them with user requirements and thus to a widespread misunderstanding of their nature, we’d be better off with a different name, such as system or top-level design.

My view, apparently somewhat contrarian, is that design is the central activity in software development, more so than in any of the physical engineering disciplines. The conventional view of software design is very narrow, squeezing it from either end. On the one hand, programming is thought of as a manufacturing activity (that is, construction), even though programmers are constantly making design decisions. In his article, Neil remarks on designers’ reluctance to take on the task of defining a system that will satisfy customers’ needs. In my view, this narrow definition seems more like the software equivalent of engineering drafting than of real design.

So, in addition to the three approaches Neil describes, and beyond a mere name change, we must change perceptions, and to do that, we need to go back to how developers are educated. For one thing, we should do away with all those “Requirements and Design” courses. Over the years, I’ve taken several, both in school and at work, and in every case they’ve been about user requirements. At best, design appears in the last couple of sessions and is presented as if it were just a continuation of requirements gathering. Instead, design needs to be taught as an activity in its own right, with an emphasis on abstract representations and the verified transformation from one to another. Above all, it should emphasize design as a creative activity that first generates multiple solutions and representations and then chooses between them.

Getting developers to think in abstract terms is perhaps the hardest part, and I don’t know how to do it. Our profession talks about abstraction, but relatively few developers can envision systems as layered abstractions, each creating requirements for the layer below while simultaneously being implementations of the layer above. The problem Neil describes can be seen as “just” the top level of this deficiency.

Andrew Raybould
Securities Industry Automation Corp.
andrew.raybould@ieee.org

Neil Maiden responds:
Thank you for your comments. I agree with much of what you say. I was working with the term...
“system requirement” in the article because it’s well understood in the domain. You’re right to point out that there’s an inconsistency in the use of the term, because systems requirements often must be designed. An interesting case could be made for reclaiming the design activity in software engineering, one that I would support.

I, along with colleagues Suzanne and James Robertson, have been arguing that we must move away from requirements elicitation to concurrent requirements and design invention, in which you can apply creativity techniques to explore the problem and solution spaces early in a project. We’ve had some successes, reporting the use of creativity workshops in conferences and magazines such as *IEEE Software*. For example, take a look at “Provoking Creativity: Imagine What Your Requirements Could Be Like” (N. Maiden, S. Robertson, and A. Gizikis, *IEEE Software*, vol. 21, no. 5, 2004, pp. 68–75).

My interest in this direction was originally triggered by how I was teaching software engineering to undergraduate students. We tell them to follow processes to produce models, but not how to use the models constructively. In too many courses the model is the end, not the means. The provision of processes taught that the transformational processes often encountered in software engineering discourages creativity. I ask, where do use cases come from? Creativity is their source of many user requirements and design features.

So, I also welcome a debate about software engineering education and getting more creativity and invention into the curriculum. 😎

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.