

**Two Books for Thinking About Software:**  
*Great Software Debates* (Alan Davis)  
*The Laws of Software Process* (Philip Armour)  
Review by Scott Duncan

I like books that expect the reader to think about and react to the content in some substantive way. Even if you cannot accept an author's premises wholesale, a book can still be thought-provoking. Both of these books are like that. *Debates* is a collection of, mostly previously published, essays designed "to make you think about the things that you may have taken for granted up to now about software." *Laws*, similarly, is a collection from "The Business of Software" columns in *Communications of the ACM*. Both nicely consolidate and expand on the previously published material.

One example of the content addressed in each book has to do with a common theme over the years in software: whether software development is art or engineering. Davis compares software to custom home design/construction in which "it is recognized that different skills are required at different times." For some things you need art, for others, engineering, and a mature discipline knows which is needed at what times without arguing either/or. Armour says software development is not fundamentally about product development but about knowledge acquisition and suggests that four kinds of skill are needed at different times on most projects: *learning* during requirements gathering, *creative* during design, *tactical* in implementation, and *problem-solving* during debugging and testing.

If knowledge acquisition is a key to software's creation (as some research in the 80's suggested) that could explain a difficulty going from requirements to executable code: incompleteness in transcribing knowledge from one form to another. Gathering requirements is about gaining expertise in what the system should do. However, customers often aren't completely sure what a system should do and cannot completely describe what it is they do know they want it to do. Over the years, "agile" approaches to development have evolved to address the problem of incomplete knowledge at the beginning of software development efforts.

Both books cover more than this, of course, since the materials span many years of writing by each author. But this ought to give you an idea about the nature of some of what I found interesting to consider in these books.