Alex Groce (agroce@gmail.com), Northern Arizona University

Donald Schön's *The Reflective Practitioner: How Professionals Think in Action* is a guide to being a thoughtful, decent, fully human, *philosophical* software engineer. It does this by way of looking at the question of how professionals can examine their own practice and behavior, while practicing a profession. Case studies in architecture, psychology, systems science, materials science, town planning, and business management are used to explore this question and present the idea of a *reflective* practitioner, who thinks about action while acting. Let us pause a moment, for an epigraph:

> *Epictetus: …* Every man wishes his sons to be philosophers while they are young; but takes especial care, as they grow older, to teach them its insufficiency and unfitness for their intercourse with mankind. The paternal voice says: 'You must not be particular; you are about to have a profession to live by; follow those who have thriven the best in it.' Now, among these, whatever be the profession, canst thou point out to me one single philosopher?
> *Seneca:* Not just now; nor, upon reflection, do I think it feasible.
>
> - "Epictetus and Seneca," Walter Savage Landor (from *Imaginary Conversations*)

Schön's book was in my mind a great deal recently, in part because I actually read it for the first time, but mostly because it seemed to suddenly appear before me constantly. First, I encountered a mention of it in the course of some other reading that I cannot now recall, which brought me to acquire and start reading the book. Second, Mary Shaw, who I was interviewing for a forthcoming book of conversations with eminent scholars and innovators of software engineering, mentioned *The Reflective Practitioner*, and directed me to a particular part of the book concerning the distinction between solving and setting a problem. Next Daniel Jackson, interviewed in the same context, prompted by my mention of the book, made a comment to the effect that transforming students into reflective practitioners is what is required if we are to enable them to be effective collaborators with AI, rather than yet another category of weavers displaced by a new, and doubtless very different, Jacquard loom. Then, finally, I read the above passage in Landor's book of imaginary conversations between historical figures (one of the unique neglected classics of the nineteenth century, as rewarding, and as little read, as the work of Thomas Love Peacock).

Is the software engineer a "professional" practitioner? Are you a professional, dear reader? Am I a professional (ignore that my category could be software engineer, or teacher, or even writer)? In Schön's book, his canonical categories of professionals are engineers, physicians, lawyers, architects, teachers, and social workers (the ministry, dentistry, agronomy, and urban planning also appear from time to time). At first glance, this settles the matter: software engineers are engineers; engineering is the first-listed and, other than law and medicine, perhaps the most traditional of the professions. On the other hand, I think everyone knows that software engineering is, as a field, aware that it lacks some of the claims to "engineering discipline" present in a field like civil engineering, which dates from prehistoric eras, has a fairly clear and coherent legal and scientifically-based practice, and requires professional licensing before most of its functions can be performed. Most software is not built the way bridges are built, both for good and for ill, and the rarity of cases in which software is rejected as “not up to code” is such as to make any such event a news item.

Schön’s book does not mention the word “philosophy” often, and when it does it is in the context of a brief introduction to logical positivism as the source of the dominant model of professional practice, that of Technical Rationality. The word “philosophy” does not appear at all in my interviews with Mary Shaw and Daniel Jackson. However, I think it is a most relevant word when examining what Schön’s book has to offer the practicing or researching software engineer.

Schön’s book covers a very wide variety of topics, and concludes with some discussions about the politics of expertise (critical of both radical, which now would cover both left- and right-oriented critiques of knowledge and expertise, and traditional, “we know best,” status-quo centered viewpoints). But at heart, I think Schön’s final section, even in the middle of a discussion of social and political realities, summarizes the core idea of the book in a way that goes even beyond societal issues: “reflective practice is an alternative to the traditional epistemology of practice.” That is, it proposes a considered, examined, approach to professional knowledge that neither dismisses the idea of specialized knowledge nor evades the responsibility of the professional, in our case the software engineer, to reflect on, test, and improve that knowledge in the course of practice. Clients are not children to be patronized, but participants in the design and evolution of a software system; engineering requires not only problem-solving (“how do I make this function that does this very particular thing?”) but problem-setting (“the goal here is very broad; how do we choose a particular kind of software, and how will that shape the reality of users of the software, including what they see as the problem they are using software to solve?”). Fundamentally, what Schön’s reflective practitioner must do is *not take expert knowledge for granted*, but consider the sources, uses, and development of that knowledge. Good software engineers must be epistemologists, concerned with what they know, and how they know it, both with respect to software development in general, and about the particular task at hand.

Practicing in any field (for us, designing and creating software systems) involves some question of how we know what we are doing; this question can be shoved aside as irrelevant, in the context of credentials (a degree in computer science from an ABET-accredited university, for example), past success (jobs at famous centers of software development), and the rote nature of much of what we all do when creating software. However, in any substantial practice, Schön points out, moments of “uncertainty, instability, uniqueness, and value conflicts” arise. In these moments, an unreflective approach to practice will be in trouble, and only the epistemologist can proceed without paralysis. Epistemology, I had long thought, was central to my own subfield of software testing and verification; I thought about having my university-provided business cards include the subtitle “Applied Epistemologist.” Schön’s book shows that the centrality of thinking about how we know what we know, and what, in fact, we can know, is not limited to software testing, but is critical to effective software engineering practice in general.

The particular point Mary Shaw referred to Schön's book as examining is a good case in point. Computer science education, for a variety of reasons, focuses on the case where one is given a fairly narrowly defined problem (e.g. a homework exercise, test question, or class project) to solve. Even classic "capstone" classes where a real customer proposes a problem, tend to focus on unusually narrowly well-defined software problems. But the highest levels of software design tend to take place in settings where the problem itself must be set; engineering of all kinds has to concern itself with defining the appropriate problem as much as with solving a well-defined problem. Even in highly technical design domains, such choices have to be made. The designers of hardware and software for a Mars Rover, a problem unusually lacking in political/economic concerns, have to choose between models where a rover allows for many low-quality scientific instruments, giving many scientific communities "something, but not much" and models where a rover completely neglects some categories of instrument in order to allocate more weight, budget, and scientific power to particular categories of scientific data. When software is addressing less technical, more societal needs, the problem-setting element of software design becomes even more central. Exploratory, epistemological, *reflective* software engineering practice is more applicable in these settings than narrow, positivist, "application of a set of scientific theories to highly well-defined theories." For one thing, increasingly, the narrower "solve for the code-block x in this equation, given your CS education" problems may be left to various artificial intelligences to address, while humans apply their attention to larger-scope problems for which AIs are more likely to prove inadequate.

The quotation from Landor above suggests an underlying theme of Schön's book: it is not trivial to be an epistemologist. Institutional, bureaucratic, educational, political, and societal beliefs and opinions can stand in the way of reflective practice. There may be times when career and reflection contradict each other, and retreating to a stance of un-reflective practice seems to be the best way to thrive. *The Reflective Practitioner* offers an eloquent argument for proceeding in spite of such obstacles. This is important, and a replacement of "professionals" with "software engineers" in one of Schön's final statements, I think, captures the full scope of the issue: "Under the perspective of reflective practice, [software engineers] are neither the heroic avant-garde of the Technological Program nor a villainous elite who prevent the people from taking control of their lives." Avoiding that hubristic, arrogant Scylla and despairing, paranoid Charybdis sounds like a good idea, to me.