From *Software Development*, October 2001.

Software: Product or Service?

By Bertrand Meyer

Herr Rektor, distinguished colleagues, Ladies and Gentlemen: I am honored that you have chosen to attend my Inaugural Lecture as Ordinarius Professor of Software Engineering. Today, I shall address a question that has troubled many distinguished people for a long time, a question that involves the very nature of what we do. Is software a product or a service? I intend to analyze the issue from three different sides: technical, commercial and legal.

Since I am a software engineer, or, to be more accurate, a software engineering professor, I feel less at ease with the legal and commercial elements, but I will start with them anyway, then move on to the aspect that is directly relevant to this new Chair. I don't see in the audience any of my distinguished colleagues from the Law Faculty and the Business School, even though they received the invitation, so there won't be anyone to watch my forays into these areas too closely.

Tell It To The Judge

Legally, it's evident that software is a product, not a service. Anyone who doesn't recognize this must be some kind of Walküre who's been riding a flying horse for the past 20 years. For one thing, software can be copyrighted, and even—much to the chagrin of some—patented. Has anyone ever heard of copyrighting a service? If it's a service, it starts anew each time you render it. No person to render the service, no service. Software doesn't fit in this definition. You deliver a piece of software to your customer, and from then on he can run it again and again, without involvement from you or anyone in your organization. Try to do that with a service. I was thinking of this last night when downing some schnapps—just a few drops, of course: I'm only into social drinking—with some of my good colleagues. I told old Hans, who was behind the bar, pouring the schnapps, that it was the same with him: I like his service, but I like his product even better. He waited until I had gulped my next glass and pointed out that without his service, I couldn't use the product. Tactically, I had to concede his point so that he could service one more copy of the product, but then I answered that … Actually, I'm not quite sure what I answered. What I do remember is that at some later point I was in my bed, thinking about products and services.

Where was I? Oh, yes. Software, then, from the legal perspective, is a clearly a service. This is obvious to anyone, save perhaps for a few Flying Dutchmen who have been sailing the globe on their lost ship for the past 20 years. Take patents, for example. Amazon patented their one-click purchasing technology successfully enough to deter Barnes and Noble from using something similar. But they never claimed that Barnes and Noble copied the product, which for all I know

might have been written in a different programming language, and might have used a completely different algorithm internally. It was enough for them to convince the court that the particular process as seen by a visitor at Barnes and Noble was too close to the visitor's experience at Amazon. This proves conclusively that legally software is a service. What else was the court letting Amazon patent? Case open and shut.

So from the legal perspective software is really a product. I mean, a service. Sorry for this lapse. My speech may be hesitant but my thinking is clear, as befits a Professor. It's just that on the way here we stopped at Hans's little Stube to celebrate my appointment, and his Moselwein was just a little too good. Try it when you have a minute. Make sure to ask for the *trocknen*, because the other one is really too sweet, no matter what Hans says. But no amount of the sweet or the dry would ever blur the cogency of my professorial analysis. Indeed, from the commercial perspective, software is a product, too. What else made Microsoft so big? They sell products: Windows, Office, Project, Visio. Or take Oracle, a great product company. The modern software industry started when the pioneers—not just Microsoft and Oracle, but also Borland with the original Turbo Pascal, Lotus with 1-2-3, and others—departed forever from the IBM model of the 1970s, the model of software as a service that's thrown in for free on top of a hardware purchase.

Of course, it suffices to look at some of the other successful software companies, EDS, Cap Gemini, Accenture and such, to understand that software is, commercially, really a service "Application Service Providers" is what they want to be. It's not by accident that the term for "software house" in French is *société de services*. These are software companies, but if you ask them what they do on a day-by-day basis, they'll tell you that programming is less and less their focus. Some would say it shows in their results, but that would be unfair, and we aren't here to criticize any of our esteemed industry partners. What matters is that their core business is services, and, in fact, many of them shy away from anything that would sound like selling services. I mean, products. Oracle, that great service company, understands this well; its executives will be glad to tell you that the future is in running integrated solutions for customers and shielding them from having anything to do with products and their versions, configurations, platforms, bugs and upgrades.

Cultural Imperialism

Another group, besides the old IBM, who has really understood that software is a service is the well-known monopoly of our industry. I mean, of course, the GNU folks. By providing compilers that anyone can download for free in all senses of the term, they have essentially left room for no other business model than services in the areas where they have succeeded. There's no market left for C/C++ compilers on Unix: It's all GCC. Good or bad, you will ask? Well, good and bad. Everyone has easy access to a compiler. But that compiler hasn't really progressed very much in recent years; why should a monopoly bother? In a service-oriented industry, it's in no one's interest to build a better mousetrap. In fact, it's bad for business.

The venture capitalists understand this well. Too well. Any talk of a business plan aimed at productivity tools will fall on deaf ears, as no one cares. This has reached absurd proportions. Even with the current economic crunch, companies are desperate for software developers. Yet a development tool that costs $3,000 will trigger outbursts of rage. That a good tool will pay for itself

many times over by reducing developer costs, debugging expenses and delays to market doesn't seem to impress potential purchasers or backers. In a service industry, productivity investments are an expensive luxury. In all those discussions of how to cope with the dearth of software developers, everyone talks about corporate psychology and retaining your people by offering them BMWs, but no one ever mentions the obvious: needing fewer developers thanks to better products.

An example of a product that's also a service is the conversion of Postscript to PDF, called *distilling* in the Adobe world. Very pleasant name, by the way. You can buy a Distiller product from Adobe, but you can also subscribe to Adobe's web-based service to distill individual documents. Our town used to have a distillery where you could likewise bring your own cherries and, for a fee, make your own *Kirschwasser*. It was much better than the commercial stuff, if you'll ask me, although I'll admit Hans has some decent bottles. Many product vendors could, like Adobe, make their products available as services. Economically, the interesting question is what prevents Peter, Paul or Ludwig from starting a similar service on their own sites legally, using purchased copies of the product. Of course they won't have the benefit of the original vendor's visibility, but they can provide a better price, and establish a Napster-like peer-to-peer system to let customers find them quickly.

And yet this shift toward services seems to be the latest fad for the most product-oriented among software companies. According to journalists—although, not necessarily the company itself—Microsoft's .NET is all about switching to a so-called "service model" for software. It's MSDN to the *n*-th power.

If Microsoft really thinks that way, they must have it really wrong, because there's this fellow from Australia named Szyperski (Clemens to his friends), who's been pushing this idea of software parts—component-based development, he calls it. Madame? I am not ready to take questions from the audience yet. In my time, no one would have interrupted an inaugural lecture. I can't hear you. What? Szyperski is not Australian? Too bad, I was going to compliment him on the nice Shiraz they have in the Connemarra valley. Oh, well. After all not everything that's good is Australian. Szyperski contends that we should be building our software out of self-contained, deployable components. This doesn't seem to fit at all with the fashionable view of software as services. He surely knows something Microsoft doesn't. I wish they would listen to him. Sir? Can't you please wait until the end of the lecture? What? You say Clemens actually *works* for Microsoft? Thanks for the information. Go figure.

The Final Resolution

I can see from the looks on your faces—you resemble the party guests in *Die Fledermaus* when they find themselves in jail for nothing else, really, than a trifle too much Sekt—that you must have noticed I have now moved on to our final territory, the technical. Here, my thesis is confirmed once again: technically, software is also a service. I mean, a product. I belong, after all, to a generation whose rallying battle cry has been "software engineering." Our whole effort, our whole focus, our whole professional life, our whole *Weltanschauung* has been devoted to establishing a sound technical basis for a true engineering discipline. If this term is to be more than a slogan, software engineering must be based, like all other engineering fields, on a solid repertoire of products. Of course, engineering has always used services too, but they assume this infrastructure of products

and constantly draw from it. To a certain extent, I believe we can say that we have begun to succeed in this multi-decade effort to emulate our predecessors, and that if software development today is markedly better than 30, 20, 10 years ago, it's due at least in part to the better quality of the tools we use. To try now to present services as the way of the future would be as criminal as to shout "*Pilsner Quell!* " in the middle of a crowded Weinskeller.

And so the services aspect will always be, as I was saying—I *think* that's what I was saying—the dominant one. Throughout the history of software, there have been attempts to present software construction as just nails and plywood, denying the part that's creative, even artistic at times. It started with ideas of "automatic programming," which first led to results as grandiose as assembly language and later COBOL. Then report generators and "fourth-generation languages." Then we had "egoless programming." Then object technology—what are classes if not standardized little products?—and component technology. Throughout all these dreams, it's the programmers and their services that have kept software alive and working, at least working some of the time, in spite of the managers' dreams of product lines and streamlined assembly production.

The conclusion, then, is clear. I don't need to expound on it much longer. This old debate is solved once and for all, as clearly as a glass of Tokay to get a fresh start on the day after. Speaking of glasses, our faithful Birgit is signaling to me that the drinks are ready, and I, for one, am getting thirsty, so please join me in the faculty club for a well-deserved celebration of the final resolution of this tricky question.

--

Bertrand Meyer is, despite what some might claim, a clear thinker on software issues. His latest book is *The .NET Training Course* (Prentice Hall, 2001).