WHITEPAPER

The Dirty Little Secret of Software Pricing

Stan Schneider

Mr. Customer, our price is $13,349 dollars per floating development seat. Larger teams need more support, so we charge 20% maintenance and support. Runtimes start at $800 per core and decline through a series of levels with volume, but runtimes don't take effect until you're successful. You will also need the optional tool package. That's $7,500 for each user who needs it.

It sounds well thought out and justified, now doesn't it? Sorry, it's garbage…100% pure fiction.

Software pricing has always been controversial. As the evidence mounts that open source does not control costs, it has become even more critical. How much is software worth? Who should pay for it? What's fair? Should vendors charge per floating license? Per user? For service and support only? Runtime royalties?

Many business models have evolved over time. Any successful policy must accomplish one key goal: the revenue must cover the costs – and then some – of providing the software. You have to pay somehow. However, nobody seems to be upfront and reveal The Dirty Little Secret of Software Pricing. And not understanding that secret costs people, companies, and governments a lot of money.

As a long-term vendor of embedded infrastructure software, I feel it's time everyone knew. You see, the secret is (shhhhh!):

Software pricing today makes no sense. Vendors make it all up. The real goal is to charge in proportion to how much money the customer has.

All the policies are just ways to approximate this rule. Most pricing plans take any metric, or set of metrics, that correlate to the money you have and charge for that. Most arguments about cost and value are pure fertilizer.

Why is this? It's because there's a fundamental business conflict. Software costs a lot to produce. However, the incremental cost of another customer is very low. Vendors need to cover the entire cost, but users want to pay that low incremental cost. This gap must somehow be bridged, and positioning that bridge is not trivial. So what's a vendor to do? The natural thing to do is invent justifications. This is fundamentally impossible. So, the poor vendor jumps off the real-cost bridge onto the very slippery slope of incremental cost justification. (See Figure 1.)

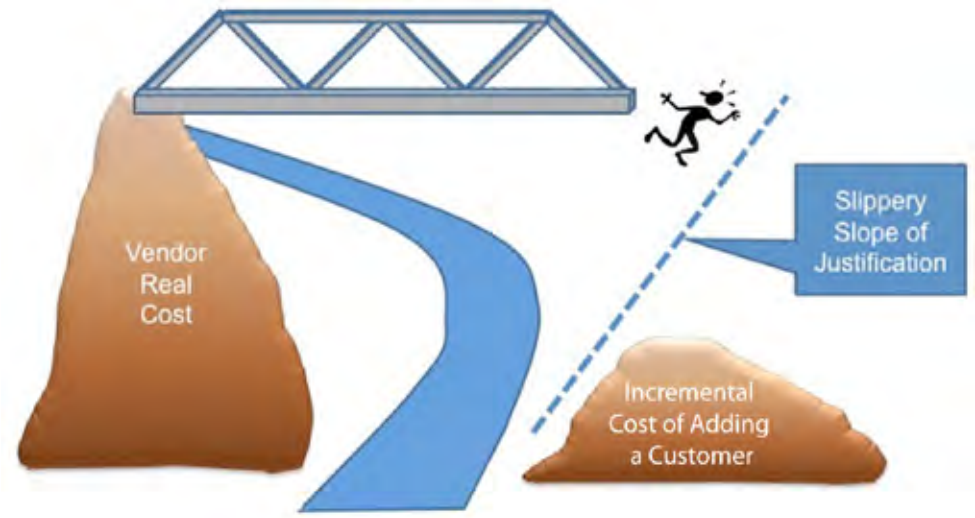

Figure 1. The gap between the vendor's real software cost and the incremental cost of an adding a customer cannot be bridged. Justification attempts fail.

Costs Justifications

The most common arguments are about cost to the vendor or value to you. Let's first look at cost.

Software is a strange beast. It costs money to develop and maintain. It requires ongoing work to keep it current and on track. Development isn't even the largest cost; it costs even more to find and match people to usage. Typically, marketing, sales, and presales support outweigh engineering. These are all considered fixed costs, because they don't really grow with each new user.

While fixed costs are high, variable costs are very low. It costs nothing to produce. It costs nothing to ship. It costs nothing to provide it to everyone. It only costs a little per installation to help people use the software.

So, the real costs have nothing to do with you, the user. The real costs are dominated by maintaining an engineering team, figuring out what code that team should develop, telling the world what the code does, and then matching the code's solution to customers. These functions require a significant ongoing investment in engineering, support, product management, marketing, sales, and services.

Unfortunately, vendors can't charge customers for that. Customers demand justification. Accountants want countable beans. Vendors often strive to bridge the gap by pricing based on some incremental cost metric. The results are not usually good.

Here are some common cost-based explanations:

- **Everyone has to pay for support, because each user adds to support load.** This is a very common cost justification. Companies claim that you pay support fees for everyone because larger teams generate more support load. That sounds reasonable, but it's not true. In reality, support costs are roughly inversely proportional to the number of people at a site. Large sites develop experts who answer internal questions and submit great bug reports. The lone clueless guy will call your support line every day. Charging support for everyone makes no sense.
- **Licenses cost money.** Another tactic is to imply there's a cost of providing you with a license. Runtime, development, source, and use licenses may carry big price tags. But, let's be real; that piece of paper costs nothing. It's just something to count.
- **We charge you for what you use, because everything you use costs us money.** This is the black art of bundle pricing. Bundling is often implicitly justified because "more software costs more." However, since shipping costs are zero, it doesn't cost more to ship sixteen pieces of software than it does to ship three. There are no additional support costs for the shelfware the user doesn't open. You can even ship the same package to every customer, an efficiency that actually saves money. Bottom line: it doesn't cost the vendor anything to ship you everything. Bundle cost arguments make little sense.
- **We charge only support and services.** This is the pure-play open-source business model: we give away the software and charge only for direct costs…hours. It's the most straightforward of incremental cost justifications.

  This sounds good in theory; you pay only for what you cost the vendor. In practice, however, it's often terrible. Each user is striving only to solve its own needs as cheaply as possible. There's a lot of "the other guy will pay" mentality, resulting in incomplete solutions and their unfortunate consequences. Many users start blindly down the free primrose path, only to run into these issues and find they need far more hours than expected. This surprise can be nasty, because many users start without even checking into upgrade or support pricing models. Often, open source vendors end up depending on a few customers, many of whom originally thought they were getting a deal.

  Following the few of those users who are willing or forced to pay is hardly a way to navigate a dynamic industry. While it works in some cases (mostly sophisticated-user-maintained or mature technologies), revenue hours are a poor driver for strategy and a poor basis for a healthy vendor relationship. Bottom line, paying for hours simply does not reward excellence. It ignores most everything required to support an emerging technology.

Value Justifications

So, if cost isn't the right basis, how about value?

Software does bring real value. For instance, RTI real-time middleware helps you build distributed mission-critical systems. Our software implements a high-quality, reliable, fast, flexible communications infrastructure. It ties a system of diverse applications into a working whole. We thus solve a challenging problem at the core of your success.

Our software is the result of our combined experience with hundreds and hundreds of problems. We've walked your path before. We have a great team of engineers making sure the product is the best in the world. We professionally track and prioritize requirements of entire markets. We do a very thorough job, through services and sales, of mapping our solution to your problem. We deliver real value, and we do that by reducing your risk, by lowering your development costs, and by guiding your success with the bright light of our experiences.

So, software delivers very real value. But vendors can't list value in this form on a price quote.

What do users expect on price quotes? They expect metrics like developer counts, runtimes, support, and product bundles. Unfortunately, the justifications for these value charges aren't usually relevant. Some typical value arguments:

- **We charge per development seat, because our software saves that developer time.** It is true that the developer is using the software. But, (at least for infrastructure software) the value is really to the project. The software may eliminate an entire expensive and risky team of people from the project. However, a GUI display engineer may not even be aware she's using an underlying commercial product…and she's still getting value. The system value is more than the sum of the value to individuals. Much more. The value is to the project.
- **We charge runtime royalties, so we share the risk of your success.** The shared risk rationale for charging runtime royalties is that you only pay if you deploy the code for revenue. This metric can work; enterprise and desktop software is mostly sold by runtimes. Runtimes make sense for quickly-deployed systems, but often not for complex systems with long development cycles.

  For these projects, the bigger risk is estimating your shipments. Calculating number of units sold, number of CPUs per unit, etc., is hard when the project is young. That makes calculating your future cost hard. Worse, you may not deploy and the revenue is delayed, so vendors take even more risk. Thus, they have to charge a lot – too much – for runtimes. In practice, the vendor often ends up dependent on a few customers who pay more than they planned. For most software, the real risk reduction value is simple: you may get it wrong if you do it yourself. That's hard to put on a quote.
- **We bundle and charge per product, because you get value out of each product.** Bundling lets you select what to buy, based on your perceived value. This value argument for bundle pricing is better than the cost justification, but it's still weak. The product selection often isn't (or shouldn't) be optional. You need to get the software that does what you need, period. Bundles are great for helping tune the purchase, but in practice there's often little real choice.

So, while they are better than the cost arguments, the value rationales are also weak. Value metrics do tend to map reasonably well to how much money you have. Projects with more people have more money. More runtimes shipped implies more money received. Projects with larger budgets will see value and pay for bundles. So, according to the dirty little secret, values are better metrics than costs.

The real problem is that these metrics get used in strange combinations with misleading justifications, and require analyses and complex approval chains. That leads to confusion, and confusion leads to surprises and unfairness.

So What Makes Sense?

Well…ironically, it almost makes sense to charge based on how much money you have. More precisely, it makes sense to charge in proportion to how much you are investing in your project. Software is expensive. If you have a bigger investment, then you face more risk and more pain. You're thus likely getting more real value, and you should pay for that value. This mapping isn't perfect, but it's better than the invented justifications. Seeking this goal drives vendors to follow the money. If they do that, then they develop the best product to fit those segments of the market in the most pain that get the most value and therefore deliver the most return.

The real imperative is that the model should be simple, open, and fair. Users need clarity without surprise. There's little reason to charge against a confusing array of metrics. Using a single metric (e.g., developers or runtimes) that maps to the size of your project correlates well with value received. It also makes software purchase costs easily modeled.

Some other metrics may fairly augment this model, but only when the value is real. For instance, customers should be able to choose those few who will interface to support, and only those people should pay. That way, support interface people can become experts on the technology and lower everyone's costs. Bundles are useful, but the goal should be to fit the solution to the problem, not just to give the appearance of lower cost. And, if the vendor offers an open source or free version, the cost of discovering the flaws or shortcomings of that free version should be open, easily calculated, and known up front.

You, Mr. Customer, must take some responsibility for making this work. Remember why vendors justify – customers are not realistic about software costs. Many of the poor results and hidden surprises arise because customers get sticker shock when confronted with real costs up front. Customers can ease the process by understanding how software companies tick and seeking to be good partners in a healthy relationship. And, that means paying a fair price for value received.

The dirty little secret should not be a secret. Charging based on your investment maps prices to real value, and that's good business. If pricing is simple, open, and fair, it makes perfect sense.

rti
Your systems. Working as one

CORPORATE HEADQUARTERS
232 E. Java Drive
Sunnyvale, CA 94089

Tel: +1 (408) 990-7400
Fax: +1 (408) 990-7402
info@rti.com

www.rti.com