Computer Business Orientation Concepts: Market Assessment

One of the best ways to make money is to identify other people's needs, often unspoken or barely coherent, and respond to them with tools which will meet, or even better, make those "needs" unnecessary. From a moral viewpoint, the trade accompanying this transaction should be made in such a way that both parties make informed decisions in allocating risks and benefits, so that each party is left with a "fair share"--as defined by the presumably well-informed participants--of all the costs and benefits, monetary and otherwise.

In the real world, things typically do not work out so directly. Costs and benefits are difficult to assess, and information is particularly difficult both to acquire and to interpret. Ironically enough, the markets for machines intended to automate information systems (such as business accounting, laboratory analyses, ad inf.) suffer acutely from problems of

Too much/too many:	Too little/too few:
Complexity	Clarity
Compatibility problems	Standards
Machine choices	Honesty/capability
Specialization	Generality.

Of course, a tension always exists between, for example, the excellent "weirdo" machine and the mediocre standard, such as the you-name-it & the IBM PC. Standards which are poorly chosen, yet univerally mandated (and there seems to be a correspondence between the two) can result in universal stagnation, as witness industrial society in Great Britain and agriculture in the Soviet Union.

It seems to me, though, that the real problems of the market can be explained primarily by extensive ignorance and ubiquitous greed--the ignorance of the customers, and greed (or even worse, corresponding ignorance) on the part of those who have to find some way of "making a living."

"Information Services"

Helping solve those problems of information, and thus helping consumers to find solutions for their information management problems, is an important key to success in the marketplace. Normally, in the process of developing an information system, something like the following sequence of analysis/implementation should take place:

system analysis	=>	system requirements
system cost analyses	=>	hardware options
(hardware acquisition?)		
¶system design(s)		
¶system coding		
¶system testing		
hardware acquisition		
end-user experience		
maintenance/revisions.		

(The real world is not the place for strictly sequential opeations: these phases are interdependent and tend to overlap considerably. System testing, for example, should influence design decisions, and proper documentation should be maintained throughout the process.)

We should note immediately that vendor fragmentation is usually extensive, so that no one is really responsible for the end results: someone produces hardware, somebody else software, somebody packages it and, responsibly or not, makes the sale. Particularly significant is the fact that **software producers/designers have little contact with end users.** When done well, their systems are more-or-less satisfactory for 90% of the applications for which they were designed--and if you belong to the 10%, you might as well go fly a kite, because nobody is going to be able to fix it, since the designers have a stake in making their software unreadable and theftproof, thereby making it also unmodifiable. (This is a direct consequence of our rather warped system of laws regarding property in ideas, which encourages research (for profit), yet discourages the dissemination of information.)

In addition, the analysis and design phases are slighted at the customer end--in the ordinary case, some salesman, software package in hand, assures the nervous customer that it is totally configurable and will meet every need, is expandable, transportable, and transmogrifiable, and simply buzzword the client into a state of confusion so profound that signing on the dotted line comes as a relief--why not, after being assured that all that fine print is just--a formality. Naturally, the salesman knows blip about software, just about as much as necesary to sell machines, but the consumer is even worse off.

In light of the temptations presented to unscrupulous vendors, it can be seen that customers need help rather badly. Good advice on machines, when an expansive market makes any judgment on the relative merits of the n-odd machines which fit your pocketbook problematic, can only help consumers to get the best buy for their money. In order to get a grasp on how much these services are worth, we might assume that the consumer would part with say, 5% of purchase price, or some $100 on average, to help insure satisfaction with the final choice.

Similarly, benefits from second opinions, not to mention genuine consultation, might come to thousands of dollars for business systems. When long-term costs (such as system hardware migration and software flexibility and maintainability) are considered, the benefits of careful consideration of automation options can be even more extensive.

We should keep in mind that the primary cost benefits (as opposed to largely intangible benefits such as improved information on one's customer base, better knowledge of job costing, etc.) derive from displacing human labor. The morality of dis-employing people in an economy such as ours, where there are few guarantees of meaningful social participation or even survival outside of providing labor services, is dubious, but given the competitive nature of the free market system it is almost unavoidable; also, the benefits of having human beings do drone work are themselves of a dubious character. Whatever the social merits of the case, in economic terms businesses have a great deal to gain by reducing employment needs related to routine tasks: employment costs for minimum-wage level employees run to around $10,000 nominally; if costs of supervision and overhead (remember, $100/sq.ft./month downtown NYC rentals translate into at least $2000/month/employee!!!) are included in employment costs, the savings generated by even half-assed automation plans can be staggering.

Of course, to do this one must genuinely lower manpower requirements, rather than merely shuffling worker categories so that now an expensive and altogether rather weird society of programmers has replaced the file clerks.

Ay, there's the rub, knowing how.

Why UNIX is only answer that makes sense, and why you should fork over $$$ to **me**.

Sales, sales, $$$, brouhaha, many factors to be considered, in the event of, user-friendly conformal interfaces, zzzzzzzz, my clothes from the Country Couple, sexy hardware options, zoooooommmmm hard disks, memory like an elephant,

Computer Mantra OMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

ALONG with me now, this computer knows where your wallet is,

DON'T be LEFT BEHIND!!!!

Software distribution

Given the irremediable defects in the concept of rights to intellectual property (shall we pay Plato's heirs for all uses of trinitarian psychologies, as the Galambosians would have it?), and the subversion of such entitlements by technological progress (soon "everyone" will have a "printing press"), how can we secure the just proceeds of innovation?

We seem unfortunately to be thrust into the arms of Blind Faith. Trust, an unreliable mechanism in the best of times, is all we have -- trust in the honesty of customers. ~~In order~~ To reduce reliance on trust, we have several options:

1) become the cheapest & most effective distributor/supporter of the product;
2) tie the product to "hardware" (including books) which is difficult to reproduce;
3) make the product 'flaky', so that our expertise is required to support it;
4) pursue follow-on market for modifications to software (aspect of #1)

1) & 2) offer effective means of counteracting "piracy" without compromising the quality of the product. Distribution via commlines will soon provide a more cost-effective ("format-bypassing") means of distribution than floppies/etc.; if [redacted] distributed code is documented adequately only [redacted] in book form, there is some guarantee of profit from book sales. (Not the best way to become rich...) This approach will be effective only so long as OCRs & flexible printers remain expensive; ideally, too, software should be self-supporting & self-documented, with all documentation available on-line.

In the case of limited-distribution high-value software, ~~software~~ hardware/software ties may be made more complex -- say by requirements for boards to carry out certain software functions or decrypt the code. This strategy is short-term, and also inspires customer resistance if perceived pricing is high enough to make duplication economically feasible.

We can also seek ~~mass~~ mass-distribution through OS vendor/hardware mfr channels. In many ways, this is the most rational means of distribution; contacts are minimized along w/ their overhead; hardware dependencies can be introduced (& may improve software performance). The package had better be very polished, if we wish to avoid appearing comical.