Beyond Buy vs. Build:

The Case for Enterprise Freeware

October 28, 1998

Marc Baber
Marc@BotWorks.com
The Bot Works, Inc.
http://www.botworks.com

Introduction

Traditionally, IT management has seen the Buy vs. Build question as an either-or decision: either you buy an existing software system or you develop a custom software system for your organization. In practice, however, nearly every software project has both a buy component and a build component. In addition to the two "pure" possibilities (100% bought and 100% built) there are two other interesting possibilities.

Suppose a company purchases a software system from a vendor. In the course of rolling out the system, the company's staff find that the software is so poorly designed, so poorly supported or so ill-suited to their organization's needs that they spend as much time documenting and reporting bugs, testing fixes and customizing the system as they would have spent building the system in-house from scratch. Or, as is perhaps more common, the company finds that in order to install and transition to the new system, they must retain a small army of consultants specializing in the chosen software system and that consulting expenses exceed the originally projected cost of in-house development. In such cases, we can consider that the software system was *both bought and built*. This is an extreme case, of course, but it illustrates the point that the labor costs (the "build" component) of installing a bought system cannot be ignored. Software companies succeed by delivering at least the promise of saved staff time or superior features, and their siren song has resulted in profound changes in the IT landscape. The true costs of bought systems are just beginning to be fully understood.

There is, logically, a fourth possibility: systems that are neither bought nor built. This concept is so inconceivable to many people that examples of it are rendered virtually invisible. But such systems exist. I am, of course, referring to freeware—software that is available to download from the internet, free of charge, source code included, under licensing agreements such as the GNU Public License (GPL). Because the Apache web server is free, it doesn't show up in traditional market share analyses of the web server software market and remains invisible to those who consider market dominance to be synonymous with product superiority. Yet, over fifty percent of all web servers worldwide are Apache web servers. As a practical example, an organization might choose to deploy a network fileserver using Linux with Window connectivity options and avoid the expense of commercial fileservers' multi-user licenses such as Novell or Microsoft NT Server.

In this paper, I'll discuss some of the hidden costs of commercial software and how many of these costs might be avoided or reduced through the use (and development) of freeware. I will also describe how the "bazaar" method of software development—a radical and refreshing new approach exemplified by the Linux O.S. development community-- might be applied to the development of Enterprise software for use in corporate and large organization IT divisions. While we can't yet calculate all the potential benefits, benefits that have been demonstrated thus far are astounding and call for far more widespread use of freeware.

The Siren Song of Commercial Software

The promise of commercial software is not so different from the promise of freeware. The common premise is that the costs of developing a software system that is useful to many organizations need only be incurred once because, unlike other products, there is almost no additional cost to produce copies of a software package after the first copy has been developed. If ten organizations need a General Ledger system, for example, and they all share the costs of developing one system, then each organization's portion of the cost is about 10%. The price per customer can therefore be far lower than the cost of developing the whole system. The more copies that can be sold, the lower the vendor's price can be.

Since the costs of developing commercial software are ultimately borne by a large number of customer organizations, an independent software vendor (ISV) can draw on the resources of a whole market segment. The vendor can tap those resources to hire more and better software architects and programmers than a single user organization could budget for. It follows, then, that the vendor should be able to produce software of higher quality and better functionality than any single organization could alone.

By relying on purchased packages, an organization may be able to reduce its IT staff or leverage its existing staff to cover more projects with essentially the same head count. It is typically assumed that the costs of deploying a purchased package are negligible compared to the purchase price.

Purchased software packages are also "ready today" as compared to internal development options, which are ready at some difficult-to-estimate point in the future. Thus, purchased software has a strong claim to reduced risk and faster delivery of a solution—factors that increase the value proposition of the software package over internal build options.

A more cynical attraction of purchasing software packages is that an IT department may be able to dodge accountability for the shortcomings of the purchased software. How many times have you heard the following "purchased excuses" for: (a) feature deficiencies and bugs: "There's nothing we can do—that's how the software was shipped to us", (b) poor support of the software: "All we can do is report the bugs to the vendor and hope they'll get fixed in the next release", (c) delayed schedules: "5.0 isn't due out until next year, but the vendor has a history of missing release dates, so…", and (d) general liability "well, we could sue the vendor, but it probably won't get us the software we want any sooner".

The attraction of such excuses can be difficult to resist within organizations where:

- project leaders become scape-goats for delayed software projects

- overly aggressive cost-controlling results in understaffing and underfunding of development groups
- the business climate requires managers to choose between underestimating development schedules and risking project termination in favor of perceived alternatives (which are often also underestimated)
- there is staff incompetence or obsolescence.

Compared to the 100% internally built software project, purchased packages can be better, faster, cheaper and less risky. It is no wonder internal development projects, once the norm, are now disappearing and becoming exceptions to the rule of purchased software.

Dashed on the Rocks—The Commercial Enterprise Software Hangover

As with any successful approach, the approach of using vendor-supplied software has been applied more and more widely in diverse environments under varying circumstances. Purchased software began with single-user single-platform systems and has progressed along the spectrum to complex heterogeneous networks. Packages that entered the market as stand-alone programs with non-existent interoperability have progressed along the interoperability dimension to the point where interoperability can make-or-break a software sale. In today's large-scale enterprises, vendor-supplied software is approaching its limits in environments that require global interoperability, worldwide connectivity, and maximum platform diversity in both the Intranet and the Internet.

The approach of using vendor-supplied packages naturally continues growing in scope until practitioners encounter circumstances where costs outweigh benefits. Just as computer hardware has both a purchase price and a "cost of ownership", so software has both a license fee and what I'll call a "cost of use". The following sections introduce some factors that tend to drive up the actual "costs of use" of software within enterprise-scale organizations.

Low Volume = High Cost of Use + Low Quality

Enterprise software—large scale financial and accounting packages, multinational accounting systems, billing systems, inventory, payroll, accounts receivable, accounts payable, and such—are some of the lowest volume, highest-priced software products in the industry. Because they are low volume, enterprise software packages tend to be lower quality because they have been tested by fewer previous customers. Also, because it is much cheaper to ship a new software release to a few large customers instead of millions of small customers, the cost of fixing bugs in the field is less of a deterrent to poor quality for enterprise software vendors.

Low volume negates the central reason for purchasing vendor software. As volume approaches the worst case limit of one, development and testing costs are shared by fewer organizations. At the limit of one customer, of course, you're back to the full costs of in-house custom development plus overhead inherent in out-sourcing software development to a vendor. At low volumes, the advantages of combining efforts over in-house building are lost to increased software complexity to meet more than one organizations' requirements and increased communication costs to get

requirements from their source (the organization's internal staff) to the vendor's development staff (often far away in an inconvenient time zone).

Complex, Unique Organizations require Complex, Unique Software

Most consumers of enterprise software, are large-scale organizations. Large organizations tend to be older, more mature, more bureaucratic, and less flexible than smaller businesses. They tend to reject the "one size fits all" approach of software vendors and demand rich user configuration options as well as heavy customization of software. The former dramatically increases the complexity of the software. The latter leads to version explosion and consequent support nightmares for vendors. Both result in higher costs to the software vendor, which are passed on to the customers.

Value Pricing Means Economies of Scale Go to the Vendor

The price that the vendor charges can be whatever the market will bear, as long as the price doesn't exceed the customer's alternatives (such as building the software from scratch). In enterprise software, for a number of packages, we see the price of buying a vendor-supplied software system creeping up closer and closer to the cost of building a custom system. Especially, with so-called "value-pricing", vendors' prices are now based not on equally shared development costs, not even on the customer organization's size, but are instead based on what vendors think the software is worth to the customer organization—namely, whatever they think they can get away with.

As long as this trend continues, and customers believe they have no alternative but to custom build in-house, then prices will approach the cost of in-house custom development.

Lack of Open Standards Leads to Over-dependence on Single-Source Vendors

Whereas open standards are often used in large-volume software packages, such as web browsers (HTML), compilers (C, C++, Java), operating systems (Posix), 3-D rendering (OpenGL), and even database systems (SQL, ODBC), such standards are found less frequently in low-volume enterprise-class software. If you've ever struggled with trying to use one vendor's network job scheduler with a disparate set of vendor packages with different ways of setting up job parameters and different ways of reporting execution results, you understand the problem. If you've ever tried to use database vendor A's GUI application generator forms with database vendor B's relational database, you've probably hit the wall, or you've had to graft in a middleware solution.

In spite of the customer appeal of open standards, there are powerful incentives for software vendors to try to lock-in customers with proprietary non-standard features. If vendors can't be first to define an API, they'll typically introduce enhancements to the standards to try to entice users to become dependent upon their versions of software. Once your organization is dependent on one vendor's solution, your organization is at their mercy for license renewals, for upgrade prices, for maintenance and support fees, for bug-fixing schedules and for new feature development schedules. And, if that vendor goes belly up or gets bought out by a competing vendor, your organization is in even more serious trouble.

Almost all of the arguments that support dual-sourcing (or multi-sourcing) of widgets in a manufacturer's supply chain can be applied to software sourcing in the enterprise, even though software has traditionally been more akin to factory machinery (capital) than raw materials (costs of goods sold). Today, with software life cycles shortening, and one-time software purchase prices being replaced by periodic licensing, support and upgrade fees, software resembles a capital expenditure less and less and has begun to resemble an expense or raw material. Single sourcing leaves the customer with zero negotiating power and lets the software vendor hold all the cards. Multi-sourcing of software components that adhere to open standards allows the customer to negotiate for price, quality, and schedules. Only through multi-sourcing can there be true competition for the customer's business.

Unfortunately, the conflict of interests between vendors and customers with regard to open standards appears to be firmly entrenched in the software industry today. An ISV's voluntary adherence to open standards tends to commoditize the software and drive down prices—a strong disincentive for standards compliance. In the computer hardware industry, on the other hand, open standards are the norm, prices are low, quality is high and profit margins are sub-micron thin. The differences between the hardware and software industries are black and white.

I believe there are several factors contributing to this difference. First, the hardware industry monopoly of the past (IBM) broke up during the PC revolution. While there are still near monopolies (like Intel) that try to set proprietary (closed) standards, most components like disk drives, CD-ROMs, memory, floppy drives, monitors, etc. adhere to open standards. The availability of open standards-complying components creates conditions where technically savvy hardware integrators (like DELL, Gateway, Micron, and other PC vendors) can use multi-sourcing to negotiate for price and quality. In this arena, standards compliance is no longer negotiable—it is simply demanded by the hardware integrators. In the software industry, we still have monopolies, we don't have enough open standards, we don't have sufficient choices to effectively multi-source and we don't yet have widespread reliance on software integrators between software vendors and users to enforce those open standards.

Although it is a foundation of the information economy, the computer hardware industry is still rooted in tangible products that have definite per-unit costs of production in the form of raw materials, labor and capitalized production equipment. It may be that the reason we don't see more competition (multi-sourcing) in software is something inherent in the nature of software production.

Low Per-Unit Costs Reinforce Software Monopolies, But Innovation, Quality and Service Suffer

Software vendors have an incentive to provide the best offering in their market, but no better. After several years of development by a large staff, the barriers for market entry by a new player become so huge that very, very few software companies can afford to enter the mature market with a competitive new product. If a single vendor achieves unassailable dominance in a software market sector, then it has no serious competition and it has no incentive to improve its product. A vendor that has achieved such dominance then has a choice of how to increase it profits. If it can't significantly grow the market by improving its product, it can still increase profit by reducing costs. It can reduce customer support staffs. It might re-deploy software developers to work on other projects. It could hire cheaper staff—typically by exploiting younger and/or

immigrant staff, even overseas staff. Software companies understand the negotiating power in commodity markets and try to apply that principle to hiring programmers. But, when costs are forced down, typically quality is diminished too.

Because the dominant vendor can slash costs to almost nothing (since copies of software cost next to nothing), the dominant vendor can easily drop their prices as low as is necessary (even free if they have to) to crush upstart competition. Would-be competitors know this and typically won't enter a market where a dominant vendor exists. Not only won't venture capitalists fund start-ups that aim to enter into a mature market—they typically won't even fund start-ups that target a market that *might be entered* by one of the larger software companies, such as Microsoft.

I have a proposed dissertation topic for an Economics or Game Theory graduate student somewhere. Perhaps it's already been done but I haven't run across it in the literature yet.

If you developed a mathematical model of competition in the software industry, and you started with N vendors of an identical software package to be sold to M new customers each year, with periodic upgrades for repeat business and various fees for support, new development funded from some ratio of revenues and new investment, and some random factors for market fluctuations… would one vendor always randomly end up dominating the market in some finite period of time? Even if all vendors pursued similar strategies and similar policies? Is the system stable or unstable, with regard to the ability to predict the winner? Is access to early capital decisive? My hypothesis is that perhaps the reason we don't see more multi-sourcing (and therefore competition) in the software industry is because the zero marginal cost of producing more copies of software leads to a winner-take-all outcome. I hope we'll see some serious study and interesting results for this topic in the near future.

If it Isn't Broke, Why Fix It?

"Do we really need WhizBang 8.0? We seem to be doing just fine with 7.3…"

Paradoxically, software vendors can't seem to leave well enough alone, either. In order to keep their customer base buying something, they have to come out with new releases and terminate support for older releases as they go. Beyond a certain point, customers are stuck on the upgrade treadmill—installing releases they don't really need with bloated feature sets that have little to do with they want or need. Mature releases tend to be compatible with the vendors' strategic partners' software (latest version only) and incompatible (by accident?) with their competitors' software. Certainly, they do fix some bugs in old features, but often not the ones you wanted fixed. And now, you need a bigger server or a faster desktop to run the upgrade.

Source Code Secrecy Hides All Manner of Evils

The real costs of "bloatware" are probably never seen by anyone. I am referring to the high costs of poorly maintained and poorly managed source-code. There is good design and bad design. Good design supports flexibility, easier maintenance, good separation of functions, and adherence to APIs (potentially open standards) where they exist. Just as darkness may be defined as the absence of light, bad design is the lack of those qualities that make up good design. Few software managers appreciate the difference, let alone reward good design.

Consequently, with each wave of junior programmers that passes through a software company, another layer of poorly designed and poorly documented source code is grafted onto the original program until the original design is corrupted and almost completely hidden.

Because software source code is proprietary, few ever see it—but the symptoms of poorly maintained source code are apparent: inability to get certain “hard bugs” fixed, bugs that only get partially fixed because redundant code needs to be repaired at multiple points, inconsistent interfaces, unsymmetrical features that don’t work the way your experience with related features in the same product would lead you to expect.

Another disadvantage of source code secrecy is that software users cannot use debuggers at their own sites to diagnose bugs. Debuggers (software that allows a software engineer to analyze a program running in a controlled manner) cannot be used unless one has the source code for the software being debugged. Ideally, to fix a bug, you need to bring the source, the executable, the debugger and the problem data all together in the same place. Often, the data in an enterprise-class application is spread out all over the network and the state of the customer’s network is impossible to reproduce at the software vendor’s development center. So, network application complexity and software source code secrecy combine to create an impasse—the data and the source code cannot be brought together to fix the problem. This is one of the most frustrating and intractable problems in IT shops today.

Strung Out on Vendor-Supplied Enterprise Software?

Has your organization’s dependence on vendor-supplied software crossed over the line to become an unhealthy addiction? Take our self-test questionnaire and find out for yourself.

1. Do you find yourself waiting on the support line for an available representative longer than you end up talking with a helpful human being once you finally get through?

2. Are you developing nervous ticks after prolonged periods of listening to the “music” selections and self-promoting tape loops in your vendor’s telephony system?

3. Do you have trouble getting enough funding to purchase next year’s support contracts for your installed purchased software base?

4. Has the upgrade treadmill left you dependent on software that’s an order of magnitude more complex than what you originally thought you wanted?

5. How many times have you been forced to change vendors for a particular kind of application because your first vendor was forced out of business by a larger competitor? Because your first vendor was bought out and dismantled by a larger competitor?

6. Do you find that your organization is spending more than the cost of the software’s license on consultants to get the package configured, debugged and running?

7. Are you typically unable to trace user problems to their root causes because you have no source code for the failing product?

8. Is your vendor generally unable to reproduce your problem at their site because they don't have your hardware configuration, or your data, or your database revision or your persistence?

9. How many of your colleagues have fallen as victims of seismic activity along the General Protection Fault?

10. Do you worry that you will be unable to resolve production-down crises because you are still using a version of a critical package that's no longer supported?

11. How many instances can you recall where you were unable to resolve a problem with a multi-vendor system because all the vendors blamed each other and none of them would own the problem?

12. Do months seem to drag by in slow motion when you're waiting for the next bug FIX?

13. How often have you been disappointed by bug fix releases that didn't fix the bug(s) you thought were obviously the most important?

14. Are you losing your programmers to companies that actually want them to write programs?

15. Do you lose track of time or large fractions of your software budget and find that you are unable to recall what value was gained for an expenditure of months of staff time or hundreds of thousands of dollars?

16. Do you sometimes buy Microsoft software without even considering if there might be a competing vendor that might offer a better, faster, cheaper solution?

Count the number of items above to which you responded "been there, done that" and see below for your organization's software abuse ranking.

0-4 You are still in denial.

5-10 Your organization could benefit from software abuse counselling.

10+ Seriously at risk. Seek professional care immediately.

Enterprise Freeware—Suits in the Bazaar?

The success of the Linux operating system is attributed to what Eric Raymond [Ray98] has called the "bazaar" model of software development. The model is characterized as decentralized and unpredictable (as if most software development was predictable). Software is released early and often. Source code is included in the published versions. The Internet is used to distribute each version and developers located all over the world often collaborate from a distance, typically without meeting each other face to face.

A bazaar-style project requires a special kind of leader. It's an open question whether the leader needs to have exceptional software design talent, but Raymond asserts that the leader of a

bazaar-style project must be able to recognize and integrate well-designed software. The leader should also be friendly and the sort of person who can persuade (using personality, cajoling or "charm") other skilled people to want to help him or her with the project. It is also assumed to be a given that all participants have a common interest in the completion of the project. Besides a desire to use the resulting software, freeware developers are, in the absence of monetary compensation, motivated by more ego-related drives such as respect of peers and design aesthetics—the Art of software development.

The stereotypical freeware developer is, dare I say, a rather different animal from the stereotypical corporate IT staffer. What am I talking about? For one thing, the freeware developer doesn't look like people in IBM ads. Freeware developers look more like people who work at IBM's research centers. They generally respect ideas of merit more than positions of authority and titles. They tend to set direction more than they take direction. Given the choice between design integrity and schedule or budget, they'll choose design integrity. They defy generalization… well, most of them anyway ☺. They cannot (and should not) be controlled. They are not in it for the money and are therefore incomprehensible, unpredictable and fear-inspiring to those who are in it for the money. In short, four out of five doctors agree that freeware developers, taken as a group, scare the hell out of IT managers.

Do I really suppose that IT managers will embrace a software development model that, at least superficially, represents a significant loss of control? From my perspective, the question is somewhat absurd. IT managers have already proven their willingness to hand over so much control to software vendors, that any shred of control left is paltry. But there is a big difference. IT managers are *comfortable* enough that they understand the motivations of software vendors enough to believe there will be adequate consistency in the vendors' evolving products and that their organization won't be stranded with unsupported software. Most IT managers are not yet comfortable with the idea that their freeware software will be reliably developed and maintained by people they are not paying. At any time, an unpaid software developer can say, "I don't need this" and quit the project and there's not a damn thing the manager can do about it. That's the scary part.

The reassuring part is that new people can join the project at any time and the more organizations that need a given software package, the more resources will get committed to its development. Even if all the developers quit, you'd still have the source code. You could still hire your own programmers to complete the work.

As I envision the enterprise freeware model, in time, most packages will be simply downloaded and used with minor modification and configuration. But for now, while things are just getting started, enterprise software will probably be developed by small groups of paid professional developers within a number of medium to large corporations and governmental organizations who would cooperate and collaborate over the internet using shared data models and designs. There is a big difference here. The Linux development community included many unpaid individuals from academic and hobbyist backgrounds because these groups were representative of the future users with a common interest in freeware Unix—hackers in the proudest sense of the word. The developers representing groups of users of enterprise freeware will be those who are currently responsible for delivering solutions to their users—IT department staff and consultants.

These distributed teams would include both individuals and small groups working together at the same physical location. Using development systems on their company's LANs, they would develop applications compatible with the shared data models and distribute the code and

documentation to the other groups via the web. Some groups would set up test and development database servers that could be accessible to co-developers world-wide. Most user interfaces would be developed in Java using GUI development tools to enable client portability on Windows and Linux desktops. Client GUIs would generally connect to databases using JDBC or similar open standards for database connectivity. Server applications could be developed in any of several languages depending upon requirements for speed, ease of maintenance or internet connectivity. In particular, C/C++, Java, and Perl stand out as likely choices. Groups in different countries would make modifications for local financial, tax or accounting differences as well as translate documentation. Data models and API specifications would become the open standards of the enterprise freeware community.

Within the enterprise environment, I expect a pragmatic rather than purist (some might say "religious") approach to freeware will prevail. For example, a developer might use a commercial GUI builder to write a freeware Java client that would then connect to a freeware data model stored in a proprietary relational database running on a Linux server. A proprietary web browser might be the front-end user interface for a CGI-form-based set of freeware SQL queries that could run against some future freeware OLAP data warehouse. When commercial software vendors provide cost-effective components based on open standards—use them, by all means. But whenever freeware components appear to provide a lower total cost of use than the commercial choices, go for it! The duty of enterprise freeware developers is, as always, to deliver high-quality, reliable, maintainable and supportable solutions to their user communities for the lowest total cost of use.

In order to hammer out data models and APIs that work for a larger number of organizations, developers might gather in small conferences for a few days to lay the conceptual foundations of designs or to vote on proposed standards. These conferences would be occasions to talk shop, compare notes, or meet users from other organizations lobbying for new functionality.

In time, it may be that gurus of international repute in particular application areas will emerge within your own company's IT department. If your company has a strategic interest in the ongoing development of a certain enterprise freeware application, it might make sense to fund the salary, or "chair" if you will of a natural leader within that application's development community. Such an employee might spend a lot of time travelling, coordinating and educating his or her application's development community and might never work on your company's proprietary work products at all. But such a developer-leader might well be worth their weight in gold to your organization because of their ability to harness a far-flung development community to develop freeware that always meets your organization's evolving requirements.

In summary, I would not expect IT department freeware developers to adapt to the freeware culture of today. There will be give and take. IT departments will bring project planning discipline and the skills of paid professionals working in real-world business and government organizations into the freeware community. And the freeware community just might bring more fun into the careers of IT department workers everywhere, possibly increasing job satisfaction and employee retention. Or, to put it another way, perhaps you might stop losing your best employees to your software vendors.

Enterprise Freeware—How Good Can it Get?

In the previous “Dashed on the Rocks…” section, I presented a litany of problems associated with vendor-supplied enterprise software. In this section, I revisit those problems to see which might be solved by enterprise freeware.

Low Volume = High Cost of Use + Low Quality

In general, the total cost of use for a low volume (small user community) software system will always be higher because the costs of development must be amortized (or borne) by fewer users. Of course, the purchase price for freeware is zero, which will probably always be lower than the price of commercial software. But, if your organization must rely on truly unusual software, you’ll probably encounter higher costs of use whether you choose commercial software or freeware.

With freeware, the cost of use is highest for the initial implementers and the early adopters. The quality increases and the cost of use decreases over time in proportion to the *cumulative* number of previous users. In the commercial software industry, by contrast, prices tend to be high initially (until competition appears), falling during the market shake-out period, and then steady or slowly increasing after market dominance is achieved. With freeware the cost-of-use approaches trends toward zero as a package reaches maturity.

With freeware, however, all of the project parameters: cost, quality and schedule will be entirely up to you, so you will not be forced to live with the trade-offs chosen by your vendor. Also, you may have the option of making relatively inexpensive modifications to an existing generic freeware package.

Freeware Reduces Complexity and Boosts Efficiency

The problem of large organizations demanding more flexible, complex and configurable software has a truly elegant solution in freeware. Most organizations set these options once and don’t change their configurations. Today, because source code is generally not made available to customers, the mechanisms that make the software user-configurable must exist in the compiled binary code delivered and the state of the most recent configuration choices must be stored as data in either a file or a database. Every copy of the software must contain the binary code to support all the configurable options, whether the options are used or not.

Because source code is provided with freeware, it is possible to use program language syntax to encode configuration choices. The software would contain many conditional compilation blocks, known as “ifdef sections” to C and C++ programmers, corresponding to the various configuration options. A header file full of C macro definitions with copious documentation embedded would be very simple for even a novice programmer to modify (or a menu-driven perl script might generate a custom header file).

The benefits of this approach are numerous. First, the software becomes much simpler. Gone are the executable modules and GUI screens used for entering configuration selections. Gone are the files and/or tables used to store those configuration selections. Gone is all the “dead code” for options you’ll never use. If you have ever been bewildered by the clutter of value-validation tables in an entity-relationship diagram, or if you’ve ever worried about users accidentally or maliciously adding new, unapproved values to validation tables, you might find

that eliminating those small validation tables and replacing them with C-macros or enum-declarations will simplify your data models, speed up your applications and eliminate some security concerns at the same time.

Freeware and Value Pricing

The obvious answer to how freeware addresses the problem of value pricing is that freeware is free and it costs nothing. Considering total costs of use, it is fairly easy to see that all economies of scale in freeware projects accrue to the benefit of the users. What may be more interesting to consider is the potential effect of the freeware option on the value pricing practices of commercial software vendors. Today, if your only viable alternative to a single-source software package is to develop your own custom system, then the vendor knows that their product's value to you is only a smidgeon less than the cost of single handedly developing the whole system yourself, plus the value of instant gratification. They've got you. Try this experiment the next time you're negotiating software prices: tell the sales rep that you are considering participating in a freeware development project with four or five other organizations similar to yours and you project that your share of the development costs will be only 20-35% of their asking price. Suddenly, you have another alternative. Even if the idea of enterprise freeware turns out to be a total washout and not a single line of enterprise freeware code is ever written, you may be able to save a lot of money for your organization by simply considering the freeware alternative and convincing sales reps that you're serious about it.

The potential effect of freeware on open standards is also dramatically different from commercial software. By completely eliminating purchase price, the motivation to create a monopoly disappears and the motivations of the user-developer community ("prosumers" [Nai]) come to the fore. Whereas software vendors want to impress customers with the length of their features lists, users want to simplify the software to minimize the development work. Whereas software vendors want to create their own standards and try to get you to buy more and more of their product family, by getting you locked into a proprietary API, users want to re-use as much existing software as possible by conforming to existing APIs.

It is not necessarily a given, however, that freeware developers will always conform to existing standards. Different opinions about user requirements and technical merits of different approaches will exist, and sometimes implementations will diverge. But, because the source is distributed, each point of view will have an equal chance of being implemented and the user community can choose the implementation they like best—the "market" still decides. The possibility of multiple divergent implementations is the control that tends to make freeware approximate the benefits of multi-sourcing even though it appears to be a single-source "market". Good leaders can help minimize effort wasted on divergent efforts by encouraging open discussions of the merits of different approaches and by trying to build consensus within the user community before multiple implementations are coded.

While freeware with its non-existent prices and egalitarian access to source code represents the polar opposite of monopoly software, it is possible that the motivations of user-developers are not as compelling as greed. The lure of profits may be more motivating to some software developers than the convenience of personal use. This may mean that freeware might tend to suffer from lack of competitive drive the same way monopolies eventually suffer from lack of competitive drive—freeware might, typically, be less innovative, of lower quality, and the service component

might have to be bought separately from consultants. Nonetheless, all of these factors will still be in the control of the user. Any large organization can add innovations, improve quality or buy consulting support when working with freeware. In general, freeware will tend to be good enough to get the job done without any extraneous bells and whistles.

The absence of customer cash flow eliminates the motivation for needless updates. Mature software is allowed to stay in its complete form. Bloatware is avoided.

Because source code is peer reviewed in the freeware development model, developers (like performers with an audience) will tend to do their best work. Raymond [Ray98] mentions how "the free-software community's internal market in reputation exerts subtle pressure on people". The pressure of peer review may have quite a powerful and positive effect on the quality of source code.

Excepting the possibility that the freeware model doesn't generate enough competitive drive, it appears that there are good reasons to believe freeware can solve all of the problems of commercial software that I've mentioned.

If Enterprise Freeware is so Great, Why isn't it Already the Norm?

The Internet is the enabling technology for widely distributed software development. Linux was developed by people who wanted more and better Internet access and weren't getting it from PCs. At that time, expensive Unix workstations were the only commercial systems that handled Internet access well and then they cost tens of thousands of dollars.

There are many anecdotes about companies that having successfully developed software systems in-house, thought they might spin off a software sideline to market their software to other similar organizations. Most such stories end with the project getting shut down either because increased revenue from sales didn't cover the increased maintenance and distribution costs or the company felt it was losing focus on its primary business. It takes free distribution (the Internet) and zero maintenance commitment (free source code) to reduce the costs of taking software public to acceptable levels for most organizations.

Getting There from Here—A General Roadmap

Within your organization, start pilot projects for Linux servers to gradually replace your current mail servers, file servers, web servers (both Intranet and Internet), name servers, and NFS servers.

Oracle, Informix and Sybase have now all announced versions of their database products for Linux. Look into moving some of your database load to Linux servers.

Use the web to get in touch with other organizations that are getting involved in enterprise freeware.

Re-evaluate your current in-house projects and consider if they could be leveraged by converting internal development projects to freeware development projects. Publish your internally developed software on the web under the GNU Public License.

Get in touch with your IT counterparts in other organizations and learn how they plan to address certain application areas. See if there are opportunities to work together. Use web pages, mailing lists and news groups to organize projects.

As the software industry goes through shakeouts, look for commercial software that has been abandoned by second-tier software vendors and see if it can be donated to the freeware movement.

Conclusion

The truth of the saying that "the web changes everything" is becoming clearer in the context of enterprise-level IT. At least, it's possible that the Internet has created an opening for a completely new model of software development that was impossible before (the "bazaar" development model). Secondly, the Internet may have so altered the financial climate for software development that, for many applications, the inefficiencies introduced by the for-profit software vendor business model outweigh the efficiencies.

In other words, the Internet has created conditions such that the total cost of use for freeware is now lower than the total cost of use for vendor-supplied software. We may be witnessing the beginning of an era where freeware is generally better, faster, cheaper and less risky than vendor supplied software.

References

1. [Ray98] Raymond, Eric S., "The Cathedral and The Bazaar", 1998/02/04, http://www.redhat.com/redhat/cathedral-bazaar/cathedral-bazaar.html
2. [Nai], Naisbitt, John, Megatrends.

(c:/marc/eweb/Enterprise Freeware.doc)
Notes:
Check Naisbitt source, pub date, publisher.