The Beekeeper Part II

A Description of Commercial Open Source Software Business Models

James Dixon
Chief Geek / CTO, Pentaho tm

The Beekeeper is a brilliant analogy for the architecture of participation that is at the root of the success of open source. This paper elegantly explains how it can be possible that everybody wins when many contribute rather than pay, and some pay rather than contribute. - Marten Mickos, CEO, MySQL

This is part 2 of The Beekeeper.
The short version and discussion formums can be found at http://www.pentaho.org/beekeeper
Email: beekeeper@pentaho.org

April 2007

Introduction

This is part two of the Beekeeper. Part one contains the basic model and I assume that you have read it. This part contains supporting evidence and observations.

Additional Background Information

Pentaho is the third start-up that I have co-founded, the other two both being BI start-ups founded with the same group of co-founders: Richard Daley, Doug Moran, Marc Batchelor, and Adrian Marshall. The first start-up was called Appsource with a product called Wired for OLAP that is now Hyperion Analyzer (shortly to become Oracle Analyzer). The second start-up was called Keyola and its products are now the foundation of the BI suite of Lawson Software. For both these start-up we used, as our 'bible', a book called 'Crossing the Chasm' by Geoffrey Moore. Chasm Theory describes how technology start-ups can overcome the hurdles to mainstream adoption via a specific marketing and partnering strategy. Our business strategy for both start-ups was based on Moore's Chasm Theory. Many other successful technology start-ups have followed the same path.

When it came to starting Pentaho we left Moore's trusty book on the shelf. We were stepping into the new territory of professional open source where the relevance or applicability of Chasm Theory was not proven.

At that time, reaching around for reading material about open source, two things became apparent. Firstly there were no texts about professional open source models. Secondly, when it came to regular open source Eric Raymond's 'The Cathedral and the Bazaar' (CatB) is a highly recommended book. I have read Raymond's book several times and it deserves its recommendations. In it, in a series of essays, Raymond explains the roots and history of open source and looks at open source and its participants from an anthropological and motivational perspective. In one of the essays, 'The Magic Cauldron', Raymond summarizes various possible open source business models. He does not go into much detail on how those models work in practice, probably because Raymond, very commendably, sticks largely to his own experience and did not have any in-depth experience with professional open source models at the time. My intention here is not to plagiarize Raymond's works in any way but to provide some details from my own personal experience of professional open source. I hope these pages are taken in that spirit.

The ultimate question I want to answer here is: Do all the principles, effects, and disruptive nature of open source apply to professional open source or does its commercial bias fatally pollute its own foundation?

I will describe how, to my mind, professional open source works in practice by looking at the differences I have experienced between working in proprietary software companies and working in a professional open source company. I will contrast and compare proprietary, open source and POSSs from many perspectives and point out interesting comparisons as they come up.

My intention is that you do not need to read either 'Crossing the Chasm' or 'The Cathedral and the Bazaar' in order to understand this paper although they are both great books that are well worth reading. If you do read them both you will come to appreciate how hard they are to compare to each other. They describe very different worlds.

Crossing the Chasm

Moore contends that the early adoption of a new technology involves two groups of people: technology enthusiasts (techies) who review and 'bless' the soundness of the technology and visionaries who see the

potential of the technology and want to use it. The visionaries trust the opinions of the techies and are willing to take the risks of using the technology on a larger scale. So far so good. The next phase in the adoption involves the 'early majority' or 'pragmatists' of the market. This group of people are much more conservative than the visionaries. They want a 'whole product' that is proven and endorsed by people they trust, is packaged in an easy-to-consume and maintain form, that comes with guarantees and warranties, and has specialists that they can engage to provide services, support, and training if needed. The 'Chasm' described by Moore exists between these two populations of adopters as the 'early majority' often view the visionaries as maverick risk-takers who are not to be trusted. Moore suggests marketing and partnering strategies that can be used by a technology organization to get across this 'Chasm'.

The history of MP3 music players includes an example of 'Chasm Theory' in action. In 1998 the Recording Industry of America Association (RIAA) took Diamond Multimedia to court in an attempt to block the sale of their Rio MP3 player. The Court of the Ninth Circuit eventually rejected RIAA's case. After their defeat RIAA made the statement that it really didn't matter because it was too hard to download music and transfer it to the Rio for it to gain mass market adoption. They were predicting that the Rio could not cross the Chasm.

At that time Moore was in agreement that MP3 players were pre-Chasm. In his description of the techies he said 'At the moment I am writing this sentence they are on the Internet at an MP3 site downloading songs to play on a Diamond Rio playback machine.'

in 2007, 9 years later, we can hardly use 'owning an MP3 player' to identify the techies amongst us. Even the Pope has an iPod.

In the end the MP3 player that crossed the Chasm was Apple Inc's iPod, not the Rio. Why? The iPod came in multiple styles and colors, with a desktop interface (iTunes) to make it easy to transfer songs, an on-line store for buying songs (iTunes Store), and a very effective mass-marketing media campaign. Also, very significantly, Apple had support from the music industry because the iPod includes encryption to protect their rights. The iPod was a 'whole product' designed for the mainstream market. Credit is due to Moore as he specifically identified Apple Inc's Steve Jobs as being a prime example of a visionary.

Chasm Theory, as defined by Moore, only applies to discontinuous innovation (one requiring a change in behavior by the consumer). Some people jump up and down and wave their arms in frustration when people refer to it in the context of smaller (not discontinuous) innovations. Some say that open source does not represent a discontinuous innovation. I say, from experience, that Chasm Theory can be applied very successfully to all kinds of innovation. I also will show that either way open source represents a discontinuous innovation because it does involve a (positive) change in behavior.

Open source has a large number of technology enthusiasts that bless it and there are many visionaries companies that are using it in production. But the barriers listed above are classic 'Chasm' ones. What Ray Lane was saying was that open source is not a 'whole product' and by Moore's definition open source software is a pre-Chasm technology.

In my career I have been involved with creating 'whole product' for 8 different software packages. In my experience this is not an easy process. There is a lot more to creating a product than the 0's and 1's of the software. Making it work is easy, making it easy is work – hard work. The open source model is great at producing software, but it does not produce 'product'. Thus the barriers to adoption of open source are real.

There are purists in the open source community who feel that open source exists in a realm that should not be open to commercial organizations. The opportunists and pragmatists see the removal of these barriers

and the potential of mainstream adoption of open source as a huge opportunity. This second group contains the founders of the POSS Companies.

Professional open source companies exists to create a 'whole product' around an open source project (software+community) and deliver it to a mainstream market. They must get across the Chasm on their own.

If you take a good student of business or economics, gave them only 'Crossing the Chasm' and an understanding of open source, they should be able to predict the existence of professional open source companies.

Operational Observations

The proprietary software development approach is what Eric Raymond likens to building a cathedral. It is undertaken by a closed sect of people who work in isolation until the edifice is complete. The open source approach Raymond likens more to a bazaar where there is much interaction between consumers and producers and dynamic competition for space and participation. Hence the title of his ground-breaking work 'The Cathedral and the Bazaar'.

Raymond approached his comparisons from an anthropological perspective. I will reach many of the same conclusions by making comparisons from operational, procedural, and practical perspectives.

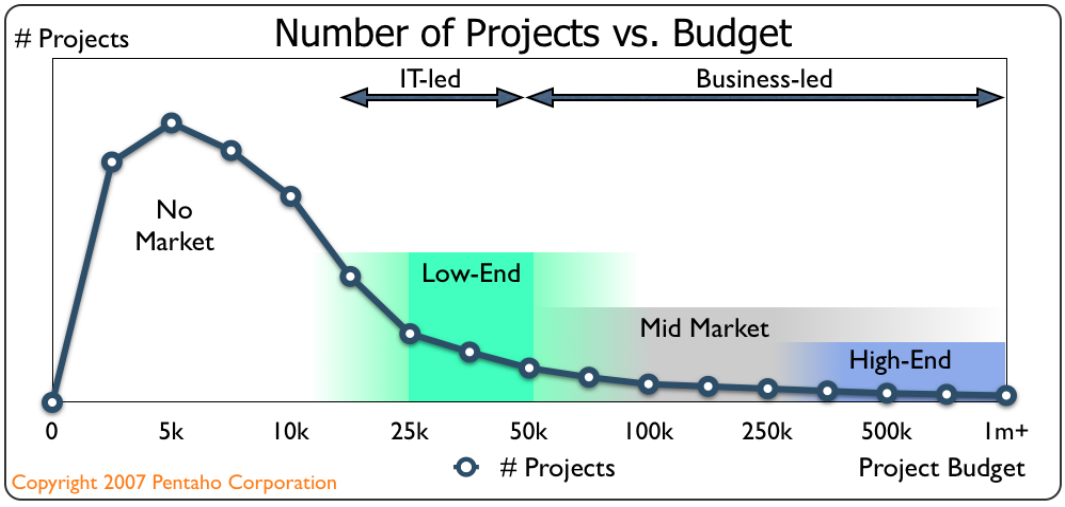
In the following sections I will discuss how the factors that affect COSS, proprietary, and open source models.

The Market

Now that we know a little more about the model used by these kind of COSS companies we can look at their potential impact on a market.

Within any given market domain there are usually multiple products available in different price ranges to meet different needs. The price asked for these products is typically not based on how much value the customer will get from the product. For example I don't get a discount on a toaster just because I'm not going to use it very often. But from my perspective I will probably determine the maximum I am prepared to pay for a toaster on that basis. The amount I am willing to pay is related to the value that I think I will get from the product, it is not related to the value that the vendor thinks I should expect.

If we apply this principle to a domain within the software industry and plot the number of projects that might be undertaken each year against the amount that the consumer is willing to spend (a factor of the perceived value) we get a chart like this:



I have added zones for the low-end, mid-market, and high-end proprietary products in a sample market. The numbers on the axes, nature of the skew, and the transition points between the zones varies from domain to domain and product to product but the overall shape of the chart will apply.

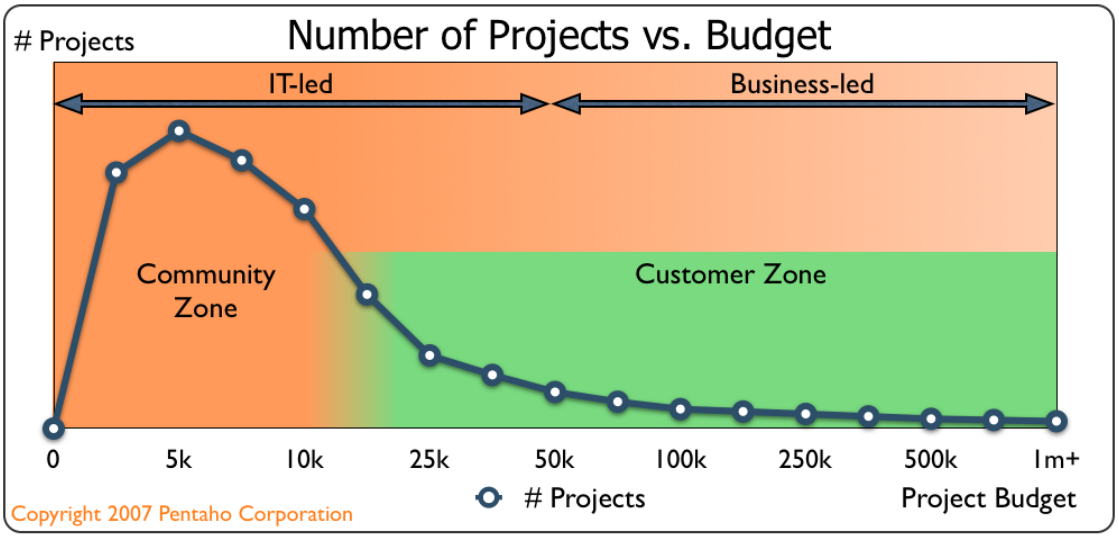
Note also that some of the projects are lead by business (CIO, sales, marketing) and some projects are lead by IT. Typically the projects lead by IT are in the low-end region of the chart (below $50k in this case).

Note that below a certain price there are no products available from any software vendor that meet the needs of the consumer. Below this price point the consumers have to make do with on of the following:

- No solution. The consumer has to do without the capability or has to stick with their existing solution.
- Suboptimal solution. The consumer has to use a solution that is based on a technology or product that is not ideally suited to the task. The solutions often have many manual (error-prone) steps in them.
- Custom solution: For example hand-coded by an employee. There are longer-term risks associated with this option.

There are other consumers who are blocked by reasons other than price: for example by lack of support for their language or text direction or by lack of local service providers.

For a COSS company in this domain the line on the chart is the same of course however we now add 'the community' to the chart.



You can see that there is still a minimum budget needed in order to become a customer. But there is no minimum needed to become part of the community. This is a very important aspect of the COSS model.

There should be no entry barrier to joining the community.

In any given time period the ratio of software downloads to new customers is typically in the range of 100:1 to 1000:1. These conversion numbers are difficult to calculate sensibly. Some of these downloads are by potential customers evaluating the product, some are by freeloaders looking to get something for nothing, and some are by community members. As stated above the community and the potential customers are very different from each other. The COSS company needs to execute well on converting the employers of

the potential customers and freeloaders into customers. Attempts to turn community members themselves into customers is not well received by the community.

The majority of the downloads are from the community and therefore estimating how well a COSS company is converting downloads to customers is problematic at best.

You can also see that the community zone extends all the way up the budget scale. This is also important.

As we have already stated customers are corporations and the community are individuals so the community is not a collection of organizations of any kind. We have also stated that there considerations other than budget (e.g. localization) which might preclude participation as a customer.

The community is not the collection of organizations too poor or too cheap to be customers.

A member of the community may or may not be participating as part of their employment. If it is part of their employment it is not necessarily part of an approved or well-budgeted program. This can be frustrating to the customer-focused departments of the COSS company.

Here is an example. At Pentaho we recently worked with a community member on some new functionality for Pentaho Data Integration. The community member works for a company with one of the highest market capitalizations in the world. They provided us with a new use case for our software and worked with us by providing feedback on our proposed design and testing various prototypes in their environment. From a community perspective this was a success: we gained a new use for our software and had it tested in a real world, large scale (60 CPU) environment. What was less successful was any attempt to get a quote, reference, or press release out of the situation (to help our market-focused groups). This was hard to do because these are corporate (customer) functions. In this case the individual participated as an active community member and added value to Pentaho. But what, if any, are the obligations of the organization that they work for? Despite the huge resources of the company I don't think they have any obligation at this point. This could be a skunk-works project, a proof-of-concept, or a side-line interest of the community member. The community member might have been doing this without the knowledge of his employer. However if a different organization takes this and put it into production, relying on its function within the business and getting value from it, should their obligation be different? By employing the community member and giving us a new use case they have contributed already. A different organization using that new feature maybe has a different obligation to participate in some other way but it is the responsibility of the COSS company to provide ways for those organizations to contribute.

Notice also from the chart that the slice of the action available to IT-led projects is significantly increased in this model: it now extends the entire width of the chart. There is a fortuitous coincidence here. The only group within an organization that is in any way equipped to handle raw open source is IT. They are the only group capable of taking open source components and performing a prototype with them. At the same time open source brings mid-market or high-end functionality into the low-end (commodity) market and opens new opportunities for IT. Consider these two conflicting facts:

- Recently Chris Koch, an editor at CIO Magazine, asked Bernard Golden (CEO Navica and blogger at CIO.com) “why do CIOs not care about open source?”
- eWeek Magazine named open source as one of the top 10 things that IT was thankful for in 2006.

I contend that this dichotomy is a classic 'whole product' one. IT is more comfortable with raw (pre-chasm) open source, whereas a CIO wants whole product to reduce risk. In addition some CIOs only think they do not have to care about open source and are often enabled to maintain this illusion by the organization they

work for. For example Jonathan Schwartz at Sun Microsoft blogs about a meeting with the CIO, CTO and CISO of a large commercial institution who completely unaware that their organization had downloaded MySQL 1300 times a year and that many of their internal systems work built with it. (http://blogs.sun.com/jonathan/entry/freedom_s_choice)

To look at it another way when a need arises within a business a 'build vs buy' decision needs to be made. It is up to the IT group to decide whether 'build' is even an option and in many cases the answer is 'no'. Open source brings much functionality that enables IT to answer 'yes' to the 'build vs buy' question more often. See below for more information about the build vs buy decision.

This creates an opportunity and a challenge for the COSS company. In the past many of these implementation projects where handled by a vendor's professional services or by consultants. The COSS company is bringing high-end capability, whether that is data integration, business intelligence, document management etc to a new audience. With the COSS model IT is enabled to take on these projects themselves, however they often have not tackled these kinds of projects in the past. The COSS company needs to provide methodologies and best practices in order for their consumers to be successful.

The Long Tail

This expansion of the market to include time-only and low-expenditure projects introduces a long tail into the demographics of the market. The POSS company is introducing an opportunity for projects to exist that were not economically viable with a proprietary solution. These are new opportunities and do not detract from the market of the proprietary vendors. In typical 'long tail' fashion the effect of this market expansion is to reduce the % of the total market that can be attributed to the largest projects.

Company and Management

When you look at the founders and management teams at a COSS company you often see a common situation:

- The founders, managers, and many of the staff at Pentaho (open source business intelligence) have worked at Hyperion, Business Objects, and Cognos (proprietary business intelligence) for many years.
- The founders of Alfresco (open source content management software) are ex-employees of Documentum (proprietary content management software).
- The founders of SugarCRM (open source customer relationship management software) are all ex-employees of E.piphany (proprietary customer relationship management software).

These people are not just 'interested' in the problem, solving the problem is their chosen career. They have worked at proprietary vendors in their domain for many years and are bringing both passion and experience to the COSS company.

Commercial open source companies are founded by passionate, experienced leaders of that domain.

Acquisitions and Mergers

TODO

Intellectual Property

TODO talk about auditing, GPL infection, licensable IP

Development

As has been mentioned earlier there are significant differences in the daily tasks and responsibilities of developers under a COSS model.

Forums

Developers need to monitor the community forums and help out where they need to. This does not always mean that developers should jump in and answer every forum post as quickly as possible. When the open source project is new and the community is still building this is necessary as there won't be any other community members capable of answering the posts. There are also posts that are so involved that it is unlikely that an external community member will be able or likely to answer it. But the community grows the COSS company's developers should resist the temptation to jump in immediately and allow newer community members the opportunity to help the poster. This is not always easy and has to be a conscious effort at times.

Where ever possible give the community the opportunity to participate.

The COSS company is, after all, giving people the opportunity to work for 'free'. This is an easy concept in principle but can be hard to implement without complete transparency.

The forums need direct developer involvement because the community includes developers and discussions about lines of code frequently occur.

Visibility

Developers are much more visible to the outside world in an open source project and COSS company than in a proprietary model. They are not hidden away in a campus somewhere and the most presentable of them dusted off once a year for the annual user conference. This visibility is refreshing and motivating.

Consumer Focus

Community

Developers spend considerably more time communicating with other members of the community than they do communicating with customers in a proprietary model. This interaction between engineers and consumers happens daily or hourly instead of once a quarter at a focus group or once a year at a user convention. Over time this changes the way that engineers approach design and implementation as they start to anticipate problems that the consumers may experience. Developers also start to anticipate how easy it will be to explain how to configure or use a feature and adjust their design to make it easier.

This is an important point. This is self-serving from the perspective of the developers because it reduces the community support that they need to provide. This incentive is not nearly as strong under the proprietary model. This incentive helps produce better software for the community. A by-product of this is that the customers, partners, and support and services teams all benefit as well.

Open source and COSS engineers are self-motivated by the community to produce better software.

The majority of participants in the community are technical people. They have an inherent distrust of glossy marketing material. However some of these people still represent potential customers. They are easy to spot on the forums when their posts start with statements like 'I need help. My boss has given me two days to evaluate this software'. They often use bold and uppercase text to highlight how urgently they would like the community to assist them. Other potential customers are harder to spot. For both cases developers need to know how to asses these situations and react to them.

Customers

Developers are also involved in the creation of the 'whole product' in the Go To Market program. For example:

- Providing materials for training course content and end-user documentation.
- Providing knowledge transfer to support and services organizations.
- Helping create demos and samples.

They are also involved in providing third-level (and sometimes second-level) support to customers.

These tasks match very closely to the proprietary model but most of them are never undertaken under the open source model. This is because, for most software engineers, writing software is fun, whereas the tasks needed for a full Go To Market program are much less interesting.

Job Description

You can consider a developer employed by as COSS company in one of two different ways:

- Traditional developer with forum duties: From this perspective a developer employed by the COSS company writes software, participates in Go To Market and helps out on the forums. The employer happens to have processes/infrastructure in place and an open source business model that puts some or all of the software into open source. In this viewpoint the developers are considered to be in a slightly modified role.
- Committer with benefits: From this perspective developers are viewed as full-time open source committers that the COSS company happens to employ. In this viewpoint the developers are considered to be in a significantly modified role.

Adoption of the second viewpoint has many implications. In both cases the developer is employed by the COSS company and the COSS company sets the long-term roadmap for the software and the short-term priorities of the engineer. The differences come to light when you look at the infrastructure, processes, and tools that are put in place to support the developers.

If you look at it from the first perspective you have the luxury of putting in place infrastructure that is internal to the organization, processes that include hard-coded paths between roles, and knowledge that is not publicly accessible. In short you have the option to choose how transparent and open you really are.

If you look at it from the second perspective you come up with infrastructure and processes that are more transparent. For example:

- Source code control system: From the first perspective it is acceptable to have a system that is behind a firewall and that periodically publishes the source code to the outside world. From the second perspective it is better to have a public server that is live for all participants.
- Feature and defect management: From the first perspective it is beneficial for product manager and development managers to keep engineers focused by only making current work items visible

and keeping the rest (lower priority, stretch goals etc.) off the table. From the second perspective it is better to have all the work items visible and clearly categorized and prioritized so that any committer with additional time and energy can help advance the software.

- Community liaison: From the first perspective the developers are not considered part of the community and so the community liaison role will be solely focused on the external community (and possibly be detrimental to the relationship between the developers and the community). From the second perspective the community liaison views the employed committers as part of the community and so will include them as part of their community activities.
- Participating in other projects: Sometimes, in order to complete a feature or fix a defect, a developer needs to contribute a change or fix to another open source project. From the first perspective this is an item of work that is necessary but not easy to categorize. From the second perspective this item of work is 100% in line with the job description.

This difference becomes obvious when partners or customers want to contribute or sponsor committers to the project. How do they know what would help improve the software the most? How does the COSS company handle those contributions? Whether or not the COSS company needs to put in place new processes or procedures or infrastructure depends on which philosophy has been followed.

Dual Focus

Developers are one of the few groups within the COSS company that are dual focused. Developers cannot be focused solely on the community because the creation of whole product is complex, resource intensive, and very ineffective without development involvement. This can lead to conflict if the market-focused groups within the COSS company do not appreciate the extent to which the role of development is different under the COSS model.

Development Traction

Raymond makes the statement 'Good programmers know what to write. Great ones know what to rewrite (and reuse)'. This is interesting and applicable on several levels. The architecture of the Pentaho platform is not a rewrite of the BI products we did in our other start-ups. Its primary design goals are based on the lessons we learned over 15 years of implementing our own, and other people's, BI products for customers. We started out with many years of good and bad experiences and used the professional open source model to add the ability to build upon a base of open source components. These are both significant factors.

This talks to a point Alfred Brooks makes in 'The Mythical Man Month': 'Plan to throw one away; you will anyhow'. The founders of a COSS company typically have 'thrown one away' (the previous proprietary software they created) before they even start, sometimes more than one.

Another factor is that being an open source company we cannot embed licensed components into our BI platform. Since we make it available for free we cannot incur any third-party costs for each download. When we need a new component, whether it is an embedded database, rules engine, scheduler, work-flow engine we **must** either write it ourselves or use an existing open source component. There are over 100,000 open source projects on SourceForge.net alone and then there is the Apache Foundation, and Freshmeat, and ObjectWeb etc. We have a lot of choice when it comes to components that we can use. My experience is that when we need an infrastructure or middle-ware component (that is not specific to our domain) there are, without fail, viable open source offerings. When we need a component that is BI-specific we usually need to write it ourselves.

We end up creating (and owning the copyright to) only the functionality that is unique to our domain, and for everything else we make use of existing open source components. This is a very efficient development model. Eric Raymond talks about this principle at length in his essay 'Homesteading the Noosphere' in 'Cathedral and the Bazaar'. The outcome is that we own the copyright to the core functionality of our platform and we do not need to own the copyright to an infrastructure utility such as a scheduler. This gives us enormous traction when it comes to development of functionality.

Proprietary software vendors are usually under immense pressure to get a product to market quickly. In order to reduce time-to-market they have two choices for functionality that is not core to the product: use open source components which are financially free but that can be politically costly, or use commercial components that come with a financial cost. Either way the don't own the intellectual property. The embedding of open source components by proprietary vendors leads to an interesting positioning problem for them (see the Marketing section below).

Integration Effort

Another factor in traction is the 'mergers and acquisitions' factor. When a proprietary company needs to merge or integrate products together it is either the result of a merger or acquisition or as a result of licensing another proprietary product to fill a significant functionality gap. In most cases the two products involved are 'whole products'. This makes the integration significantly harder. Integrating the features is not hard if there is only minor functionality overlap between the two products and this is typically the case. However there is often significant overlap when it comes to authentication, authorization, metadata, repositories, installation, administration etc. These features are part of the 'whole product' aspect of the software.

This makes sense if you look at the things you have to add to a typical piece of software to make it into a product. If you start with two pieces of software, A and B, that have both been turned into 'whole product' by two different vendors, you now have two **products**, Y and Z, that have different features but that have both been packaged with a similar (but not always compatible) set of technologies designed to make the products easier to deploy and maintain. It is the integration and the migration of these two products into a single one that is much harder than the integration of the core software. That is to say it is typically much easier to integrate software A and B together than to integrate products Y and Z together. What can be even harder is the migration of the install-base from the individual products to the combined one.

In the COSS model things are different because we are embedding and integrating with open source components. We are typically dealing with components that were designed to be embedded. They were designed with interfaces that support the applicable standards. In open source standards are not supported for marketing or check-in-the-box reasons. They are supported because they define boundaries in the technical landscape that enable open source developers to focus on the core functionality of their project. There is no point wasting time trying to create a schema for storing metadata models when the Common Warehouse Metamodel exists. Not only would it be a waste of time recreating the work already accomplished it would be detrimental because the effect of it becoming popular would be a dilution and fragmentation of the metadata community.

Recruitment

One advantage for the COSS companies is that they can hire from their own community. There are several advantages to this:

- The COSS company can monitor the performance and quality of community members over time before making any kind of offer. The COSS company can determine that a developer is capable of understanding the software and can contribute to it within the guidelines required.
- The COSS company has a significant pool of interested developers at all times.
- Once hired the new employee needs little in the way of training as they have been working with the software for some time.

This recruitment advantage applies to many departments within the COSS company, not just development. This advantage also extends to the customers of the COSS company as well: for example because of MySQL's open source model there are probably many times more people familiar with the MySQL database than are familiar with Oracle's database.

COSS engineers needs to be comfortable with having their work peer reviewed (sometimes quite critically) by anyone that wants to. They also need to be comfortable with the commercialization of open source via a professional open source model. Not all engineers fit both of these profiles.

Community Web Site

The COSS company needs to provide a web site for the community to use. This web site is sometimes call the 'Dev Zone', 'Community' site, '.org' site, or the 'forge'.

This web site is where the principles of open source are most clearly visible.

- Availability of design, source code, binaries and documentation (openness and 'early and often')
- Forums for interaction between community members (community).
- Communication from the administrators (transparency).
- Public roadmap that is based on community feedback (openness).
- Feature and defect tracking (transparency).
- Ways of participating in the project (openness).

In order to provide these capabilities these web sites use forum software, a download site, a wiki for documentation, and a feature and defect tracking tool.

These features make up the 'project'.

Quality Assurance

This is the department where the largest difference exists between the COSS and proprietary models.

During my years of proprietary development I used to see claims about the quality of open source software but did not necessarily see why this should be the case. After spending a few years in open source I have come to believe in this completely and have some theories justify it.

When people talk about software quality they typically think primarily of bugs that are defects in the implementation of a feature. There are two other kinds of defects in software: requirements defects and design defects. Defects of these other types are often more damaging than implementation defects.

Proprietary software vendors use several methods to improve quality: regression testing, manual testing, beta programs, usability studies and acceptance testing. This work is managed and/or performed by internal Quality Assurance (QA) teams.

I will show that an open source model is more effective in achieving higher quality of requirements, design, and implementation than can be achieved with these methods.

First lets consider the fundamental differences in the overall processes followed by proprietary and open source models:L

- In proprietary software the vendor attempts to attain their desired quality level before the software reaches the consumer (customer) by using various resources and the techniques listed above. This is done in a period that occurs mainly after the software has been written and before it is released or made 'Generally Available' (GA). Once the software is 'GA' the number of people that have access to it is vastly increased and so the majority of the feedback from real users of the software occurs after GA.
- In an open source model the 'early and often' and openness principles are used to put not only the software but also the requirements and the design in the hands of the consumer (community) as early as possible and to update it often. The feedback occurs early and often as well and the improvements are built into the process early on.

In my description of the proprietary process above I used the phrase 'desired quality level'. This might seem odd, and so it should. After all the desired quality level, from a consumers perspective, ought to be 'perfect'. Unfortunately for all concerned this desired quality level turns out to be 'less than perfect'.

I remember the day I first learned this lesson. Our first start-up was our first foray into packaged, off-the-shelf software. Prior to that we had much more experience with service-focused and custom software than with shrink-wrapped software. We executed to a strategy we based on Moore's Chasm Theory and were acquired by an established vendor as per our plan. During the due diligence performed by the acquiring company they reviewed our tools and procedures and it resulted in an interaction with their engineering director that I have never forgotten.

'We examined the data in your defect tracking system and want to know where your bugs are'.

'We fixed them.' I replied with considerable confusion.

'Stop doing that,' she said, 'it takes too long, it's too expensive, and you'll never be competitive.'

How embarrassing! We were working under the delusion that bugs were bad and had been fixing them like total amateurs. It was not until I recently, after spending considerable time in an open source model, that I had much further insight into this. This was not an isolated, on-off, experience. This is the standard for proprietary software development.

The Cost of Quality - Proprietary

Lets examine the costs associated with achieving a given quality level with a proprietary development model. If we examine each cost and the factors affecting it some interesting conclusions can be drawn.

Before we get into the costs we need to understand the workings of the Quality Assurance team in a typical (at least in my experience) proprietary software company.

QA teams are part of a process that results in the release of software. In this process product managers interpret the requirements of prospective customers, actual customers, and sales and marketing

departments, and document it for the software developers. The developers interpret the requirements as described by product managers and implement them. QA engineers then ask 'how do I test this?' and create artificial data to test the documented use cases of the feature. The best they can do is to ensure that the quality of the implementation is sufficient to use the feature as it was described by the product manager. They are not the ultimate consumer of the feature and so it is practically impossible for them to verify the quality of the requirements and the quality of the design. Their inability to effectively test the quality of the requirements and the design has several implications.

- Requirement opportunity miss. It is frequently the case that a feature is design to solve a specific requirement and it turns out to be random how suitable that feature is for solving unforeseen requirements or use cases that vary slightly from the documented cases. I can recall many instances of meetings with product managers and account reps and sales reps to discuss a customer's request that a newly released feature be modified so that it truly solves their need. This is typically logged as an enhancement request although in reality is usually a defect of the original feature requirements.
- Blocked use case. Bugs that are not found prior to release are frequently associated with unforeseen use cases. Some of these bugs are obscure and harder to find but many of them are fairly glaring once the path to reproducing them is known. The result is that the customer is unable to use the feature to solve their specific problem because their use case was not anticipated and consequently not tested during the QA process.
- Patching. The result of the issues above is that patches to the feature are required almost as soon as it has been released. This is very commonplace. Creating and releasing patches is an expensive and resource intensive exercise that lowers the productivity of product managers, developers and QA teams.

Don't get me wrong QA engineers are hard-working professionals, however the position they are put in by the software vendors is not enviable. Internal QA teams fight a losing battle in which there is a high price for failure and little glory for success. As a consequence it is often hard to motivate internal QA teams. The most common way for proprietary organizations to motivate people to test their software is to offer them financial incentives (salary, benefits, bonuses etc.). The organization is attempting to align the motivations of their staff with the motivations of the organization using money. As we will see below this is not nearly as effective as having real, lasting, direct alignment of motivation.

The costs involved in software quality can be broken down into different categories.

Cost of Finding Defects

The factors involved in finding defects are:

- Finding defects is harder than finding them (in most cases).
- Attempting to find all the defects without involving the consumer is very hard.
- The fewer defects there are the harder they are to find.

The consumer and their usage of the software is an important factor in this cost. In order to find defects 'pre-consumer' a software vendor must attempt to predict the uses cases that all their customers have and then create the data and environment required to test those many use cases. Philosophically this is redundant work as the consumers already have the use cases, the environment, and the data. The vendor has to bear this redundant effort because they are attempting to protect the consumer from this process.

Proprietary software vendors use beta programs to mitigate these issues. Beta programs typically involve giving a limited number of customers access to the new software before it is released. Beta programs are not always highly effective and are often omitted because of this. Public beta programs are much more extensive than closed ones and can get better results but are not common because they require more time, more management, and are often undesirable from a licensing perspective.

The net result is that the cost of finding defects before the consumer receives the software goes up as the number of defects decreases because it takes longer and longer to find them.

Cost of Fixing Defects

Most software engineers will attest to these facts:

- Implementation defects are often harder to find than to fix (see above).
- Design and requirements bugs are harder to fix the longer they exist.
- Overlapping defects that interact with each other are hard to diagnose and fix.
- A defect is much easier to fix if a reliable reproduction path exists.
- The fewer defects there are the easier they are to fix.

The cost of fixing a defect does not usually depend on whether it was found by a developer, QA engineer, or a consumer, as long as it can be reproduced in a controlled environment.

The overall result is that the cost of fixing defects decreases as the number of defects goes down.

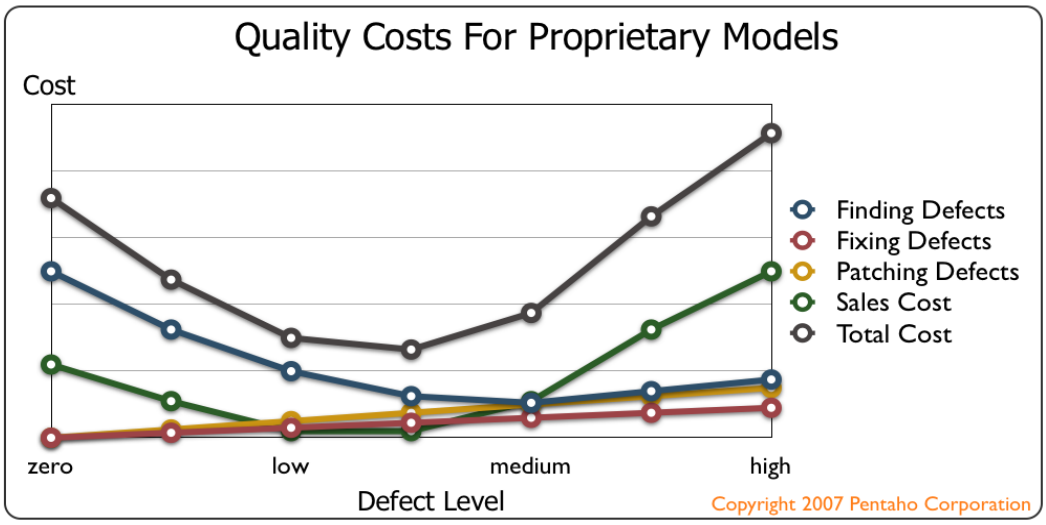
Cost of Patching Software

If defects are found and fixed they then need to be provided to the consumers. Creating, testing, and releasing patches is not cheap. Obviously if a defect is found before the software is released a patch is not needed. This cost is a factor of the ability to find defects before release and is most heavily influenced by the methods used to find defects. In a proprietary model the overall result is that the cost of releasing patches decreases as the number of defects goes down.

Sales Cost

If it costs a lot of time and money to find and fix defects and provide high quality software this expense needs to be re-couped from the customers. An increase in the price of the software will result in lost sales – another cost. Budgets are constrained these days. Ask yourself this question: If there was a competitive product to Microsoft Office that had no additional features, but had no bugs, and cost 5 times more would you buy it? The answer is usually 'no, I'll put up with the bugs' so the consumer is to blame for this one. Consumers of software frequently choose low price over high quality.
On the other end of the quality scale if software is of so low quality that either a demo does not work, or implementations are difficult, or many patches need to be issues then sales will be lost, legal issues may arise and customers won't renew their maintenance agreements. This cost becomes dramatic as quality decreases below an acceptable level.

If we plot all of these costs for different quality levels we see an interesting result.



The vertical axis shows relative costs.

The top line represents the total cost of achieving a given quality level under a model which attempts to find and fix the defects 'pre-consumer'. The lowest point of this line is the optimal (cheapest) quality level under this model. It is clearly apparent that this optimal quality level is not at the zero defect point.

The increasing sales cost as you get closer to a zero defect level is caused exclusively by the increased cost in finding the defects (not in fixing them). If the cost of finding defects did not increase as quality improves the sales cost would not increase.

Many people might have a reaction that the cost of perfect quality cannot really be the barrier that I present here. For those people I offer the following additional evidence.

Triage

When a customer reports a defect to a software vendor the case is discussed in a meeting that typically involves various product managers, engineers, and account managers. In this meeting a decision is made about when, and if, the defect will be fixed. The factors that are used to determine this include the nature and severity of the defect, which customer reported the issue (and how important that customer is to the vendor), whether a work-around exists for the defect, and whether other customers are also likely to encounter the defect. Often the defect tracking system stores severity and priority separately or stores a 'customer severity' or 'customer priority' to allow the vendor to set a severity or priority of their own. When placed in context of the chart above the implicit intent of this meeting is to decide whether the fixing of this defect moves the current quality level closer to, or further away from, the optimal level.

Orbiter Flight Computer

As further proof lets examine the costs of a piece of software that has proven itself to be of very high quality: the software running on the flight computer of the Space Shuttle orbiter. There have been two tragic accidents in the long history of the Space Shuttle program and other lesser incidents but none of them has been caused by a failure of this software. This is a prime example of software that must be of the highest quality before it is used by the consumer (the crew).

There have been 21 releases of this software and less than 20 defects have eluded the QA process and been present during a mission. None of these defects has caused operational problems.

To my knowledge this software is approximately 500k lines of code. I am assuming that of those 500k lines of code maybe 10% of those are modified on an annual basis, presumably as other operational systems on the orbiter are upgraded. So now we are dealing with 50k lines of modified code a year. It takes 280 engineers and an annual budget of $350 million to assure the quality of these code changes. If the same level of quality had been applied to Windows XP it would have cost Microsoft over $200 billion to develop it. Microsoft has a $7 billion per year budget for R&D so it would have taken them over 25 years to develop Windows XP. This is clearly not a reasonable option for a commercial software vendor.

In 2002 the United States Department of Commerce's National Institute of Standards and Technology estimated that software defects cost the US economy $60 billion a year (http://www.nist.gov/public_affairs/releases/n02-10.htm). This sounds like a lot of money until you consider that the cost of getting Microsoft Windows (tm) alone to zero defects would cost the US economy significantly more than this. In this study they also estimated that earlier and more effective identification and removal of software defects would save the United States over $20 billion a year.

The Consumer Experience

For consumers the process of reporting defects and applying patches is largely negative. They have paid money for the software and so expect it (unreasonably as it turns out) to be free of defects. They are forced to report the defect and wait to hear when, or if, it will be fixed. They don't like having to wait and they don't like having to apply the patch (which might introduce other problems since it contains other changes not relevant to their problem).

Summary

I'm not trying to discredit the quality of proprietary software other than to say that the challenges of achieving very high quality levels are a natural consequence of their attempt to achieve that quality level prior to the consumer being involved.

The Cost of Quality – Open Source

Now lets consider these same costs under a pure open source model. As mentioned above in the open source model the requirements, design, and software are offered to a community of consumers (software developers, IT personnel, etc) in a usually transparent way. The design is made public when the first draft is ready and critique is welcomed. The software is made available as it is written.

Cost of Finding Defects

The 'burden' of finding defects falls to the community. After all, they have the use cases and the data. Since they are getting the software for free they tend to expect that it might have some defects in it. I have 'burden' in quotes because it is not really a burden at all. In the majority of cases they are going about their business as they normally would. If they encounter a defect of design or implementation it is in their interest to help the developers of the open source project fix it.

The community is self-motivated to participate in testing open source software.

Since the software is available as it is written it is possible for defects to be fixed and made available to the community very quickly.

It is a critical difference between these models that the consumer is testing the software in the process of trying to solve a real world problem. The consumer has a real use case that might not be quite what the designer intended, is running the software on a platform and configuration that is probably different from the designer, and has real data. The testers of the software are consumers going about their business. It is a consequence of their natural actions that leads to defects being reported. This is in stark contrast to the motivation and situation of an internal QA team.

With a reasonably successful open source project there will be thousands of consumers in the community. In a proprietary environment you will find a ratio of developers to testers thats ranges from 1:1 in very enlightened organizations to 5:1 in those less focused on quality. In an open source model the ratio maybe 1:1000 or considerably higher. You can think of it as 'massively parallel testing'. It is true that these testers are not managed and directed to test the features that the developers think should be tested on any particular day. It is also true that many of those testers end up testing the same things that other testers do. It is an inefficient and redundant model if you look at it in these terms. But you need to factor in that these testers are not testing for the sake of testing. They are going about their job as they otherwise would and only if they hit a roadblock do they even become conscious of their role. Every community member with a unique use case and unique data (almost all of them) has a potential contribution to make.

An open source project with a small community will probably not have enough members to ensure that the software gets enough use cases and testing to attain a high quality level.

The members of an open source community will often provide peer review of a feature design before it has begun development. I have been both the donor and recipient in exchanges like this and it has been productive for both sides every time. By publishing a design spec to the community the designer often will get different types of feedback. Questions that start like this 'That feature sounds great, how would I use it to do....' are really clarifications of requirements that may introduce the designer to a new use case and to a defect in the original understanding of the requirements. Questions like this 'I like the API, however I notice that I have to provide a File object in many cases which is tough since my input data is in a database' highlight design defects. I have been involved with multiple projects where the quality of the design as it went into development was significantly superior than in its early drafts because feedback like this was incorporated.

By making the design available to the community prior to development design defects are identified much earlier than with a proprietary model.

What is the real cost of finding defects this way? The total cost (in terms of time) accumulated across all the participants is potentially very high. The average cost to each participant is very low and they have no way of collecting payment anyhow. The cost to the developer for this effort is very low. The real cost to the developers amounts to the cost of reviewing defects that are reported and collaborating with community members to get reproduction paths. If the quality is low there is likely to be a lot of duplicated effort on the part of the developers as they try to sort through all of the redundant incoming feedback. This cost therefore increases with lower quality.

There is often no schedule for the process of finding and fixing defects. The software is ready when its ready and not before. This can lengthen the software development process considerably. This process is a better reflection of reality. You cannot predict how long it is going to take to find and fix the bugs in a piece of software. In order to do that you would need to know what all the bugs are. In the proprietary model the main testing and fixing process has a set duration.

Cost of Fixing Defects

If you agree that many bugs are harder to find than to fix you will understand how the community helps to reduce this burden. Members of the community may also be sufficiently technical to be able to fix the bug themselves and contribute the fix to the open source project reducing the cost still further.

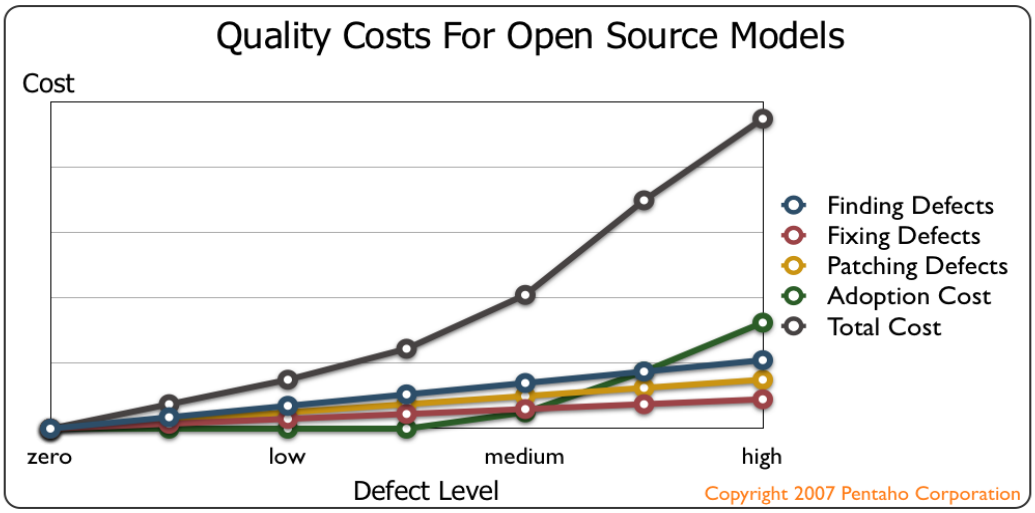
The advantages of this model in the early identification of requirements and design defects and the usage of real use-cases in the testing cannot be over-stressed.

Cost of Patching Software

This is where the 'early and often' principle starts to make a major difference. Since software and designs are made available as they are produced, identification of defects by the community can start immediately. If the developers heed that feedback and adjust their design or implementation a second feedback iteration can begin. This is part of the normal open source process and are often regular and planned and not done in a hurried 'reaction' mode. Open source projects however still do patches to fix defects and so there is a cost here.

Adoption Cost

There is no 'sales cost' to speak of in the open source model however there is a cost associated with a very low quality level as community members can simply abandon the project and adopt another one if the quality level is unacceptable. As the community diminishes the developers will have to bear the testing burden themselves.



You can see from the chart that the optimal quality level under this model is theoretically zero defects. I say 'theoretically' because in reality, in general usage, if there is a defect that no-one encounters or does not cause anyone enough annoyance to cause them to report it the defect will probably remain unfixed. If we constrain ourselves to only considering defects that the consumers care about then the 'zero' defect target is attainable.

Here are some examples to back up these statements. In March 2006 Coverity, sponsored by the Department of Homeland Security, used their static code analysis tools to perform an analysis of a collection

of the most widely used open source projects. Static analysis examines source code for suspect design patterns and known causes of memory leaks and security breaches etc. The results were made available on the web and within 7 days 900 defects had been fixed by the communities of the projects analyzed. Within two weeks the Samba (Linux-Windows network connectivity) community had fixed all 218 issues originally identified by the analysis: from a static code analysis perspective Samba (and several other project analyzed) achieved a zero-defect level. The rate at which these defects were fixed averaged less than 12 minutes per bug. To date a number of projects have reach 0 statically-identified defects, including Samba, OpenLDAP and Python. Over 5,600 bugs have been fixed across all projects. The fact that the average time to fix defects was 12 minutes supports two things:

- That defects are easier to fix than to find
- That communities enable massively parallel participation. I suspect that very few of the 5,600 defects were fixed in under 12 minutes however the number of defects being fixed in parallel by different community members brings the average time down dramatically.

This model without a doubt solves the problem of earlier and more effective defect resolution identified by the National Institute of Standards and Technology in their 2002 report.

The Consumer Experience

For consumers this overall process is much more positive than in the proprietary model. They have gotten something for nothing and so do not expect it to be perfect. It is in their best interest to report and defects they find and are typically happy to work with the developers or administrators to reproduce the defect and test the fix for it. There is also a gain in recognition and contribution of the community member when they find and report a bug. Paradoxically the expectation of lower quality is unfounded as the zero-defect level is realistically attainable under this model.

The Cost of Quality – Commercial Open Source Software

Now lets consider these same costs under a professional open source model. As you would expect this is a hybrid of the two models above but it is not 'fixed' at a specific point between them. The behavior of the COSS company determines how their model compares with the proprietary and open source models.

I have shown above that it is the principles of transparency and 'early and often' that give significant advantages to the open source model. It is natural therefore that largest factors in the software's quality are commitments by the COSS company to the transparency and 'early and often' principles.

Cost of Finding Defects

To be any kind of COSS company at all they must have their source code (and usually compiled binaries) available for the community to download. This enables the community to use the software for their purposes and they will find bugs as a natural course of trying to use in in their particular situation.

The community needs to be able to communicate about any defects encountered so the COSS company needs to provide public forums and tools for reporting and tracking defects. Unless the COSS company does this implementation defects cannot be reported by the community and the benefits of the model are vastly reduced. This requires a commitment to transparency.

Making the source code available gives more technical members of the community the ability to comment on design defects but it us really the availability of roadmap and design documents as early as possible that enables this. This requires a commitment to the 'early' part of 'early and often'.

The COSS company needs to refresh the source code and binaries on a regular basis. If this is not done the speed of the iterative feedback loop is greatly reduced. This requires a commitment to the 'often' part of 'early and often'. Enabling this to happen requires using different tools and/or policies that are used by a proprietary company.

If the COSS company is committed to the transparency and 'early and often' principles of open source they will be able to have all the advantages that come from an open source model.

If the COSS company has any functionality that they only offer as part of their whole product to customers this software will be proprietary and will not have any of these advantages.

As mentioned above only those members of the community that are 'in phase' with the COSS company at that time are likely to contribute. A large enough community ensures that there will always be enough members in-phase at any one point. Making a longer window for release candidates increases the % of the community who will be in-phase.

Cost of Fixing Defects

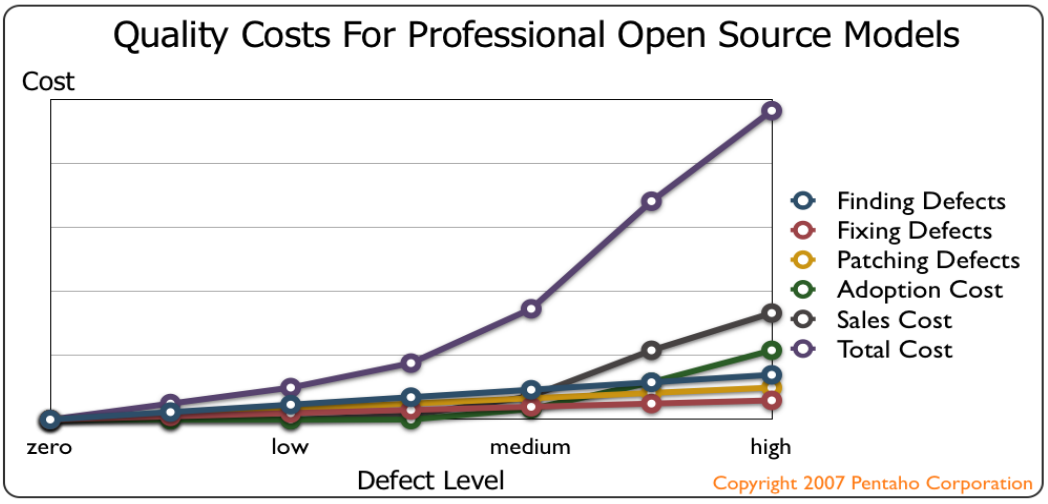
This is the same as for the open source model except that it can be considerably quicker as the COSS company has full-time engineers whereas an open source project may have a collection of part-time engineers.

Cost of Patching Software

COSS companies do patches to fix defects and so there is a cost here.

Adoption and Sales Costs

Both these costs apply to COSS companies. If the open source model is used effectively for finding bugs then the sales costs at the low defect end of the scale does not exist and the only costs are associated with high defect levels.



The optimal chart for a COSS company follows the pattern of the open source model, not that of the proprietary model. This assumes a real commitment to the open source principles. If the COSS company

does not have a sound commitment or does not implement the necessary infrastructure the chart above will follow the proprietary model.

In the COSS model the open source nature of the development process means that defects are found and fixed earlier but the existence of a schedule does limit the effectiveness of the community because that schedule might not align with their natural behavior during that period.

The Version Effect

There are significant differences in the meanings of version numbers between open source and proprietary software.

e.g. 5.0.2	• Majority of implementation defects detected and fixed.	
3rd Patch to Major Release e.g. 5.0.3	Harder to find defects detected and fixed.	

As an example have a look at this generalized time-line of the development of three features in a piece of software under a proprietary and an open source model.

Time	Proprietary Model	Open Source and COSS Models

January	'Feature 1' designed.	'Feature 1' designed.
February	'Feature 2' designed.	'Feature 2' designed. Defects in 'Feature 1' design identified by community and fixed.
March	'Feature 3' designed.	'Feature 3' designed. Defects in 'Feature 2' design identified by community and fixed.
Apr	'Feature 1' developed	'Feature 1' developed. Defects in 'Feature 3' design identified by community and fixed.
May	'Feature 2' developed. Defects in 'Feature 1' identified by internal team and fixed.	'Feature 2' developed. Defects in 'Feature 1' identified by community and fixed as 'Feature 1.1'.
June	'Feature 3' developed. Defects in 'Feature 2' identified by internal team and fixed.	'Feature 3' developed. Defects in 'Feature 1.1&2' identified by community and fixed as Features 1.2 and 2.1.
July	Defects in 'Feature 3' identified by internal team and fixed.	Defects in 'Feature 2.1&3' identified by community and fixed as 2.2 and 3.1
August	Small beta program of consumers	Defects in 'Feature 3.1' identified by community and fixed as 3.2
September	Software is released.	
October	Requirements and design defects identified by customers	
November	Patch for Features 1.1, 2.1, and 3.1	
December	Defects in 'Feature 1.1, 2.1, 3.1' identified by customers and fixed.	
January	Patch for Features 1.2, 2.2, and 3.2	

It is obvious from these (grossly simplified) time-lines that the open source model achieves any given quality level faster than the proprietary model. Note specifically that the design defects are spotted and fixed much earlier in the open source model. These defects typically are more expensive to fix the longer they are undetected.

Customers of proprietary software vendors are justified in being wary of major version releases from those organizations. However these assumptions of instability do not necessarily apply to COSS releases. Educating customers about this essential difference will be a challenge.

The Consumer Experience

If managed correctly the COSS company can make the testing and debugging process a positive 'open source' experience instead of a negative 'proprietary' one.

Quality Assurance Summary

A COSS development model has the potential to have many of the advantages of the open source model. The factors that affect how well the COSS company can enable this are:

- Commitment to the principles of open source
- Good infrastructure implementation
- Impact of the 'Go To Market' program on the software development process

In the early days of a COSS company no dedicated QA resources are needed. As the community grows a QA manger is needed to put the infrastructure, processes, and communication in place so the community is enabled to do QA themselves and that the QA contributions of the community are maximized. The QA manager needs to ensure, as far as is possible, that the testing and fixing aspect of the open source model is utilized by the COSS company.

The QA manager also needs to participate in the Go To Market program in the traditional QA manager role. The QA manager is therefore dual-focused like development and product management.

Product Management

Most open source projects are started by a single software developer working on a problem they are particularly interested in. Raymond summarizes this as 'Every good work of software starts by scratching a developers personal itch'. Richard Stallman, with reference to projects such as Linux, points out that there can be other motivations such as a well-defined problem space. In the case of COSS company there is typically a long-term roadmap that is defined by product managers who are experienced in the specific domain of the COSS company.

Product management interacts with both the community and the customers so are dual-focused.

- They need to reply on both traditional and new methods of requirements gathering.
- They need to actively involve the community in requirements and priority setting. Some feature and defect tracking tools (e.g. JIRA) support voting to allow automated collection of community feedback. A tool that provides voting enables community sentiment on decisions to be gathered in a scalable manner.
- They need to communicate the product roadmap externally as well as internally, possibly at different levels of detail.
- They fulfill one of the roles of a project administrator in the open source model.
- They do not act as a buffer between engineering and the consumers. In the COSS product manager has less contact on a day-to-day basis with the consumers (community) than engineering does.
- The software that the consumers download must meet the needs of the community and the needs of potential customers.

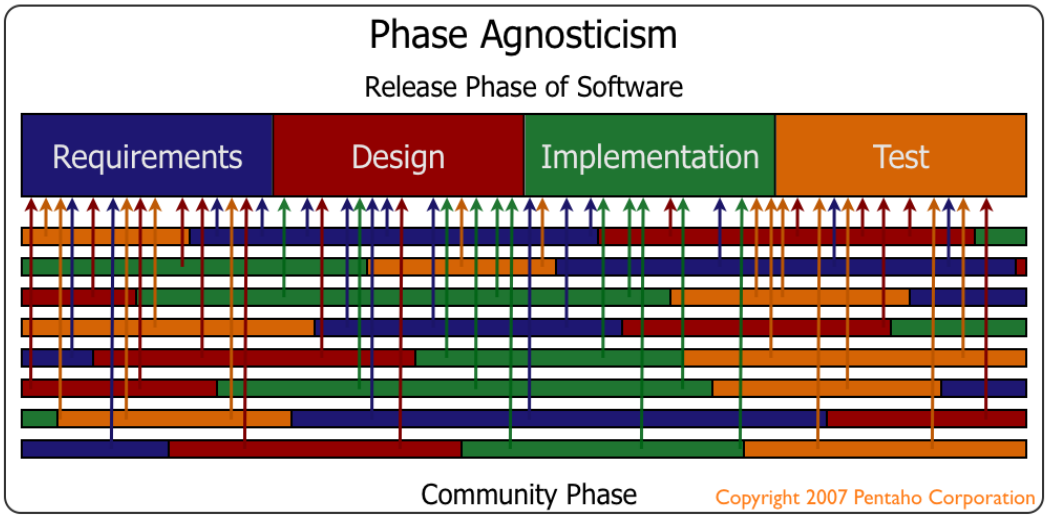
Community Phase

The open source model works well because the community members are behaving naturally in terms of what they are doing and when. For instance a community member might download a new version of the software and in the process test that it works in their environment and that it integrates with their other systems. In the process of doing this they might find the need for a new feature and write a forum post about it. Once they have done this they move on to another activities related to their project. They might not download and test another version of the software for a few months or more. It is pure chance whether

or not the community member is requesting enhancements during a period when Product Management is in a requirements gathering mood.

The COSS company therefore needs to be able to handle out-of-phase feedback. In the proprietary model the release cycle has various well defined phases during which specific activities, like requirements gathering, happen.

For example lets image that someone has just posted on a forum that they would be interested in seeing a new interface added to our software. But the current release cycle for the software is not in the requirements gathering stage. If the product manager waits until the next cycle before looking on the forum for enhancement suggestions it may be several months before the poster gets a reply asking for more clarification of their use case. It is likely that the poster will have moved on to other projects, maybe partly because they did not get a response to their post.



In diagram form you could display it like this. The big boxes across the top represent the phases of the software release cycle for the COSS project over time, starting with the requirements phase and ending with testing and patching. The smaller boxes below represent the phases of a sample of single, active community members at the same point in time.

- All phases all the time: Not matter what phase of the release cycle the software is in there is feedback coming from the community for all the phases. Note also that it is important to know which version of the software they community member is running because they may be out of phase and also one or more versions behind.
- Clumpy: The feedback is randomly distributed not evenly distributed.
- Pummel-fest: The volume of feedback is large: much larger than typical for a proprietary model.

Product management needs to monitor the forums continually looking for enhancement and feature requests. When these are posted PM needs to respond and clarify the requirement if necessary. From the perspective of the community member it may seem as if they are both in-phase, when in reality they are not. As far as possible the COSS company needs to be agnostic to the phase of the community members.

In-phase and out-of-phase feedback should be treated equally.

The existence of a schedule complicates matters. The open source model works because when it comes to testing the community is only performing their natural duties. The existence of a schedule causes problems because it means that there is a period of time, a window, defined by the COSS company during which a certain type of feedback is desired. Only those community members who are naturally 'in phase' with the COSS company at that time will be able to contribute 'in real time'.

The community cannot be made to work to your schedule.

This issue of phase changes as the community gets larger. A larger community means that a larger number of people will be in-phase at any one time, it also means that the total volume of out-of-phase, but still valuable, feedback increases.

Release Manager

It should be clear from the model diagrams and the sections above that there are two main processes involved: Software Development and Go To Market. In the COSS model the software development process needs to follow an open source model as much as possible and the Go To Market program needs to produce the same outputs as the traditional proprietary GTM.

Notice that the GTM program does not have to be the same as it is in a proprietary company, it just has to produce the same outputs: marketing materials, sales materials, training materials, press releases, analyst updates, customer references, partner materials, knowledge transfer to support and services etc.

In this area COSS companies are currently in era of process innovation because these dual community/ market processes are not standard yet.

The release manager is responsible for tracking and coordinating these GTM activities. The release manager will also be involved in the process of creating stable binaries and release candidates for the community and so they are another roles that has a dual focus.

Sales

Some people have proposed that open source will lead of the eradication of sales positions in the software industry. This is highly unlikely. What is true is that open source, and particularly professional open source, is disruptive to the traditional sales cycle.

RIP RFP

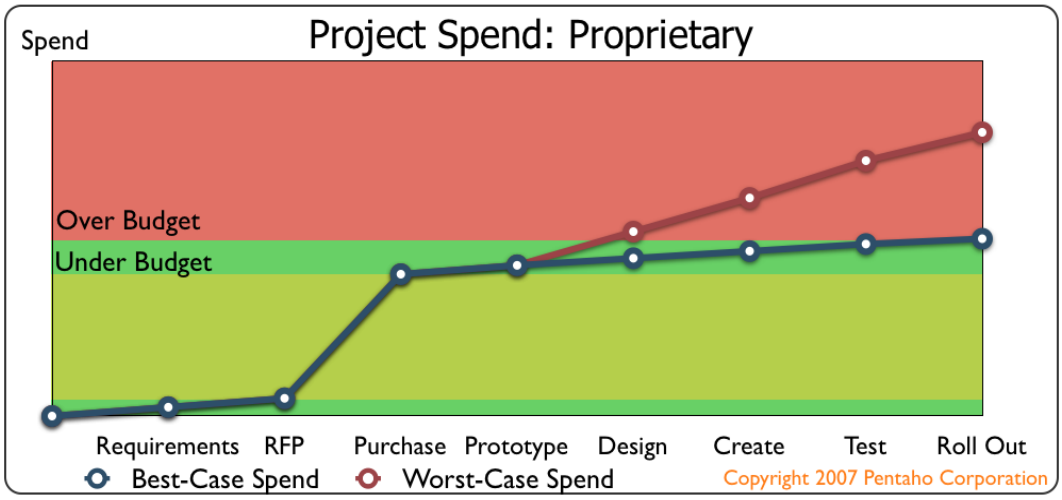
Open source and COSS companies can have a large impact on the sales cycle in certain software domains. This is particularly true any domain where a large pre-sales effort is made by the software vendor for example in enterprise software. In many cases this effort is required by a proprietary vendor in order to convince the consumer that the software is suitable for their purposes. An example of this situation is the Request For Proposal (RFP).

An RFP is part of the evaluation process at the start of a project. In the RFP the consumer describes their project and its requirements and sends this to vendors that they feel might be able to satisfy the requirements. The vendors respond to the RFP with documents, demos, pre-sales resources, or participation in a competitive comparison against the other vendors. As part of this process the customer may also get a third-party opinion from an analyst or consulting organization.

The RFP is an opportunity for the vendor to explain how its products and services could meet the needs of the consumer. The RFP is a standard mechanism used to solve a need during the initial phase of a project, but it is not a ideal when you look at the (simplified) overall process:

1. Identify a business need and possible solutions.
2. Estimate the expected return on investment (ROI).
3. Establish a budget for the project (based on the expected ROI).
4. Issue an RFP.
5. Select a vendor and purchase a license for the software.
6. Create a prototype to verify the solution and the potential ROI.
7. Implement the solution.
8. Roll it out to the end users.

The reason it is not ideal is the relative order of the prototype and the license purchase. Charting the % of the project budget as it is spent through this process you get a chart like this.



The yellow band on the chart represents the proportion of the budget spent on software licenses.

Notice that the RFP and the purchase come before the prototype. The prototype is a very important step in the process. In our market the BI gurus recommend that you take 5-10% of the project budget and perform a prototype. If you have to purchase the software in order to do a prototype it gets expensive very quickly. If you have to spend $50k on a prototype your expected ROI must exceed at least $500k in order to justify the cost of the prototype.

Notice that once the software has been paid for there is not much remaining budget for design, implementation/customization, change management, training, roll-out etc. At this point budget overruns and failed projects can occur and indeed these are commonplace within our industry.

Some vendors will provide 'free' pre-sales assistance to complete a prototype prior to the customer paying for the software. These pre-sales activities are not really free. Pre-sales expenses are part of the sales and

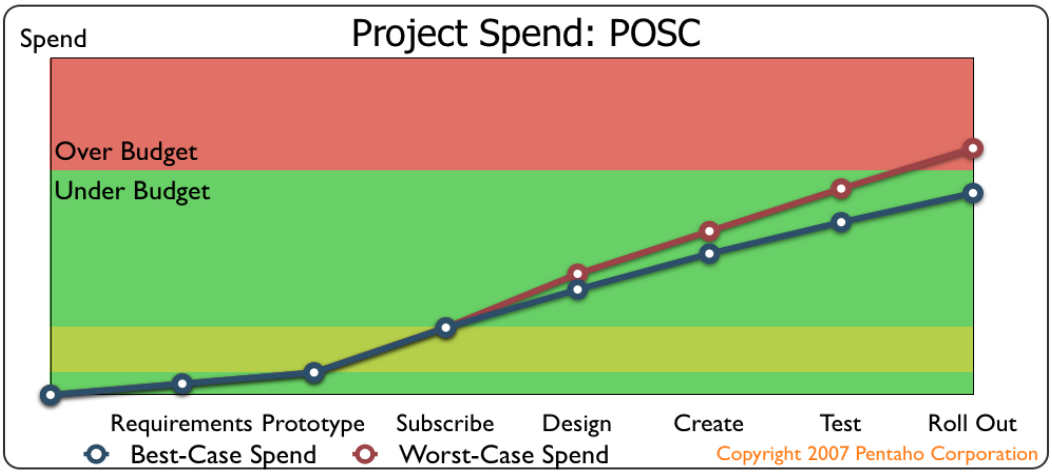
marketing expenses for the vendor and those costs are recouped by selling new software licenses. This means that whether or not you make use of their pre-sales team, when you eventually buy software licenses you are paying for it anyway. The only time the customer gets any real value from these pre-sale activities is when the prototype proves that the project it not viable or will not provide the expected ROI.

The existence of a pre-sales team on-site during the evaluation period should be handled carefully. I have witnessed situations where the development team, working directly with the pre-sales engineer, has modified the product to overcome defects in the software and shipped the modified version to the on-site team. In this circumstance the prospective customer is no longer evaluating a version of the vendor's software that is available or supported and is not aware of that fact.

The RFP is a distracting and awkward process at an inconvenient time. When there is a large price to pay prior to the completion of the prototype it increases the fear of failure, increases the minimum ROI needed, and is a deterrent to getting started in the first place. Under this model there is no ability for someone to say 'Do a quick prototype, show it to the internal customers, and see if they like it'.

This process is different with an open source or COSS solution:

1. Identify a business need and possible solutions.
2. Estimate the expected return on investment (ROI).
3. Establish a budget for the project (based on the expected ROI).
4. Download open source software and create a prototype to verify the solution and the potential ROI.
5. Select COSS company services or subscriptions.
6. Implement the solution.
7. Roll it out to the end users.



The yellow band on the chart represents the proportion of the budget spent on subscriptions or licenses.

Under this model the prototype can come much earlier in the process. After this prototype and before completing the project the consumer signs-up for the subscription or licenses offered by the COSS company.

The remainder of the budget is spent on design, implementation/customization, change management, training, roll-out etc.

This chart assumes that the COSS solution is not harder to implement than the proprietary one. Some people will argue that this is an invalid assumption however this is not a significant factor. The most important point here is that the customer can do their own prototype.

- They can prove to their satisfaction that the software does what they need it to do.
- Not only do they not have to trust a vendor or analyst, they don't even have to call one (see Pipeline below)

Also notice that compared with the original budget we are not suggesting that an open source solution is necessarily significantly less expensive than a proprietary solution. However the budget is being spent in a way that increases its chances of success as a much higher % of the budget is spent on implementation, training, change management, roll-out etc.

In order to attempt to mitigate the problems experienced by customers vendors sometimes offer evaluation licenses that are time-restricted or limited in functionality. For enterprise software suites vendors typically do not offer effective evaluation licenses for all products within the suite.

Another factor that influences budget spend is that a large % of the IT budget for a lot or organizations is already 'pre-spent' in the form of annual maintenance contracts. A COSS solution (at typically 15%-20% of the license cost of a proprietary solution) sometimes comes in under the threshold when a proprietary solution cannot.

With open source and COSS consumers can prototype without an RFP. This is a (positive) change in behavior.

The New Build vs Buy Option

When it comes to solving a business need with a new computer system many people are used to the build vs buy dilemma. Both of these options have advantages and disadvantages. The emergence of a COSS company into a market introduces a new option that was not available before.

Issue	Build	Buy	COSS
Up front cost	Low	High	Low
'Out of the box' features	None	Lots	Some or Lots
Development cost	High	Low	Low
Time to completion	Long	Short	Short
Availability of third party help	Limited	Yes	Yes
Easy prototyping	No	Sometimes	Yes
Complete ownership of result	Yes	No	Yes

For many organizations the 'build' option, whilst it does not look very attractive on this chart, is appealing partly because of the lower up front cost but mainly because the ownership of the result does not introduce reliance on an external company. The COSS option introduces many or all of the 'Buy' advantages without affecting the ownership.

The open source offerings presented by a COSS company represent a game-changer in some markets. For example in the data transformation / ETL space the IT industry, the vendors, and the analysts have been used to the fact there have always been two options: hand-coded data transformations (build) and expensive proprietary ETL tools (buy). Open source and COSS solutions (like Pentaho's Data Integration) bring an entirely new option.

COSS companies introduce a new option into the 'build vs buy' decision.

This is a change in thinking and behavior for consumers and analysts.

Pipeline

As mentioned above the consumer who downloads COSS to try it out does not need to engage anyone outside their organization in order to do so. Imaging if you could buy a car by test driving it as often as you wanted for as long as you liked without ever having to talk to a sales rep. This affects the nature of the sales pipeline.

As a COSS company we get calls from people who say things like:

- We have just completed a prototype with Pentaho and would like a support agreement now.
- I'd like to give you money so I know you'll help me in the future if I need it.

and even:

- I don't need any of your services right now but I have some money left in my budget and I'd like to start a relationship with Pentaho.

Up until these calls we have no contact with the potential customer. The availability of free downloads is great for the customer but it means that the COSS company has less visibility into its sales pipeline. The situation for the proprietary vendors is much worse however: there was no RFP, so they never found out about the project at all. This issue also affects analysts (see below).

Summary

The position of sales people is not likely to go away as a result of open source software, or COSS companies although it changes the environment in some markets. In the enterprise software market it might (hopefully) hasten the end of the obscene commission payments that sales reps sometimes get.

Partnering

From the charts above we saw that under the proprietary model up to 80% of a project budget might be spent on software licenses leaving only 20% left for services. If a services company has to bring in a software vendor to help close the deal the software vendor sometimes tries to squeeze their partner out of the account so they can pitch their services for the remaining budget.

Under the COSS model around 20% of the budget is spent on a subscription for COSS leaving 80% for services and implementation. This is

- Better for the partner as a larger % of the budget is allocated for work that they can provide services for.
- Better for customer as a larger % of the budget goes towards important tasks: design, implementation/customization, training, change management, roll-out etc.

A partner of both a proprietary software vendor and a COSS company would want to propose the COSS solution as often as possible. However a critical feature gap or scalability issue would mean having to pitch the proprietary solution instead of the COSS one. Therefore is in the best interest of the service partners to help the COSS mature as quickly as possible.

Partners see the benefit to contributing resources and become contributors to the COSS company.

When many partners do this, they all benefit. This is a change in behavior for partners.

Marketing

Within the Bee Keeper model marketing is focused primarily on the customer side although the presence of the community and the commitment to open source principles by the COSS company affects marketing in many ways. Marketing is as important in the COSS model as it is in the proprietary model. Marketing needs, like all groups in the COSS company, to be aware that the community is not a pool of sales leads. Just as in the proprietary model Marketing is focused on creating a market that the organization can create customers from.

Many of my observations here are based on conversations with and on the observations of Lance Walter, the VP of Marketing at Pentaho.

One of the results of the open source movement is that previously high-priced item are becoming middle-ware (e.g. application servers), and middle-ware is becoming infrastructure. This affects marketing in two ways:

- If the COSS company is successful it is likely to be not only a witness to but a participant in the transition of its own market from luxury into commodity. Despite Microsoft's repeated successes in this arena analysts and media often have a hard time understanding how a business such as a COSS company can gain from this process. (Lance Walter, Pentaho)
- Whilst the aims and general principles of marketing are the same for a COSS company as they are for a proprietary company a different approach is needed. Ian Howells (VP of Marketing at Alfresco) pointed out to me that COSS marketing in the enterprise software segment needs to use mass marketing methods and that the COSS sales cycle is low-touch and so the hand-off point between marketing and sales as the prospective customer interacts with the COSS company is much later than with a proprietary model.

Hype

Open source gets a lot of attention at the moment and this is of great benefit to a COSS marketing department.

Positioning

We discussed how COSS companies have to either create their own IP where it is core to their product or select and embed an open source component as they add functionality to their software. Now consider the awkward position of a proprietary vendor in the same position. For example in the BI market our proprietary competitors do not want to give credence to our BI platform that is built on a base of open source components. In fact they publicly state that 'Business Intelligence is too strategic to trust to open source'. This position is clearly difficult for them to maintain if they are embedding open source components into their own products (which they are). In addition to this Eric Raymond contends that high strategic value is a

factor that inevitably leads to the existence of an open source alternative (to reduce vendor lock-in of a strategic resource) so the statement itself is self-defeating. At the same time these vendors are attempting to position themselves as 'open source friendly' usually by claiming support for Linux, MySQL, and Firefox.

This is partly an interesting issue of intellectual property. The software industry can be considered from two different perspectives: as a service industry or as a manufacturing industry. Many people assume that software vendors are in a manufacturing industry. Many software vendors also believe this. Many software vendors have gone out of business still believing this. Others believe that software, particularly enterprise software, is a service industry, for example SAP makes $4 in services for every $1 it makes selling software.

The financial analysts who follow the BI vendors seem to believe that these vendors are in a manufacturing industry. They are fixated on 'new license revenue'. If the vendor's services revenue starts increasing the analysts are up in arms and demand to know what the vendor is going to do to raise license revenue and lower their service revenue. However if you look at the vendor's financial statements you can see that they get about 40% of their revenue from sales of new licenses. You also see that about 40% of their expenses go on sales and marketing. This means two things: firstly they don't make much money on selling new licenses and secondly when a customer buys a license for software what they are actually buying is future sales and marketing activities (that is of little or no value to the customer).

One of the side effects of taking a manufacturing view of the software industry is that you get unnecessarily excited about getting, keeping, and protecting intellectual property. Most software companies don't even own the intellectual property they think they have. For example consider the example of the Solaris operating system. Sun Microsystems decided to put their Solaris Unix operating system into open source. No biggie, you might think, its theirs anyway. However when they looked into it they discovered many components and libraries in Solaris that they had licensed from third parties. It took Sun Microsystems seven years and $120 million to clear the legal and financial obstacles necessary for them to acquire all of the intellectual property in Solaris so that they could put it into open source. Kudos to them for seeing it through.

Analysts

These are interesting times for analyst companies. It used to be that they were the sole holders of critical information. Proprietary vendors and analysts hold regular meetings so that the vendor can inform the analyst, under a non-disclosure agreement, of their product roadmap and strategies. The analyst give advice to the vendors and summarize or sell this information to potential customers in the market for new software.

We have shown above that when it comes to COSS a potential customer can prove to their satisfaction it meets their needs and they do not necessarily need the opinion of an analyst.

Analysts should not think that any lack of inquiries about open source means that no-one is interested. This would be a case of Travel Agent Syndrome: 'no-one is calling me to book airline flights, I guess that means no-one flies these days'.

Analysts need to cover the open source and COSS in their market or their analysis will become progressively and obviously incomplete.

Another factor for the analysts to consider is that they need to re-evaluate their metrics. For a long time after the emergence of Linux into the server room analysts following the operating systems market measured market share in terms of license revenue. They knew from talking to people that Linux was being

used more and more yet the results of their surveys did not reflect that. However once they started looking at the market in terms of CPUs they saw a different picture.

Here is a simple example. I took a survey a few months ago sent to me by an firm of analysts trying to asses the impact of the Red Hat acquisition of JBoss on adoption of the JBoss products. We were using the JBoss Application Server in our server offerings for both our community members and customers, that we were seeding the market for JBoss in this respect, and that we would continue to do so after the acquisition by Red Hat. However the question asked of me was 'Will I (a) continue to ***buy*** JBoss, (b) buy WebSphere, ... or (e) other'. The accurate answer to this question is (e) because I was not buying JBoss to start with. In this survey the customer / money bias of the questions meant that not only was our community / adoption relationship with JBoss not conveyed it made it seem that we were planning to decrease our relationship with JBoss. It seemed as if the questions had been created by someone who was not familiar with the COSS model and the many possible relationships that could exist.

Some analyst firms are rising to this challenge by hiring a specialist to track open source. This is a great start but does go far enough. Within a particular segment of the software market a COSS company might emerge to threaten the entrenched proprietary vendors. The success of the new-comer will depend on the exact model used by that COSS company, how well that model exploits the weaknesses of the proprietary vendors, and how the features in the COSS software match up to the proprietary software. An analyst who is familiar with open source but does not have in-depth, market-specific knowledge will not be able address this.

Unfortunately there are a lot of people who still do not understand the fundamental principles of open source. For example Bill Hostmann, Research VP at Gartner, recently said (during a Gartner Bi Summit keynote) of the term 'open source BI' - 'It's starting to lose its meaning, with some 'open-source' vendors demanding licensing fees'. Apparently Bill is one of those who think that open source is supposed to be free. Until basic misconceptions like this are removed it is hard to people to understand the COSS business models.

The existence of COSS companies require analysts to change their thinking and behavior.

No More Grand Ta-Da's?

The marketing department of a proprietary vendor is responsible for making product announcements. In a COSS company the openness and transparency needed in dealing with their community means that much of that information is already 'out there' by the time the software is complete. You might think that this means there is little point in marketing making a product announcement when it is 'GA'. However if we refer to the Bee Keeper model we see that:

- There are different audiences in the model: community and customers. These audiences use different information sources.
- Potential community members need to be informed what features are available in the open source software.
- Potential customers need to be informed what features are 'whole product'.
- Marketing has a vital role in increasing mind-share for the COSS company in both community and customer worlds.

This difference occurs because features that are available in the open source software are not necessarily available for production use as 'whole product' until the Go To Market program is complete.

For the perspective of a customer organization until the software is 'whole product' the open source features may as well not exist. Therefore the transition of the software from open source to G.A. is a very notable event that is worthy of significant marketing.

The difference between the COSS and proprietary models in this regard is that the existence of the features that will be announced as GA is not a secret.

Check-In-The-Box-Features

Another advantage to the COSS model is that it discourages check-in-the-box features. These are features that are added to software to combat a competitive threat from another product. These features are frequently added to a release at short notice often as the result of a press release from a competitor about their new version. If the development team is under time pressure the feature is often added under the following orders: implement enough of this feature that we can legally claim to have it but don't worry if it is not ready for use in production.

Needless to say check-in-the-box features are not very popular.

- They are effectively useless to customers because they cannot be used in practice.
- It is not made clear in marketing materials the feature is 'check-in-the-box' and not really intended for usage. This would defeat the purpose of adding it to the release.
- They are usually designed and implemented very quickly and often contain design defects.
- They often result in scramble patch to make the feature real when a customer buys the product based on the feature's availability.
- These features provide only a short-term benefit to sales. Longer term they degrade customer satisfaction.
- In later releases the feature has to be completed. This often involves re-writing it and having to provide migration paths for any customers who did start to use the feature.

It is very difficult to release a check-in-the-box feature under a COSS model. When a feature is added the community can immediately try it and review the source code. If there are design defects, missing capabilities or implementation defects the community will report them on the public forums. It is apparent to anyone interested that the feature is not ready. In addition any potential customer can download the software and try the feature to see if it suits their purpose. Either of these groups is able to publicly dispute marketing claims on the forums if they choose. Under the COSS model transparency means that check-in-the-box features are a detriment to all parties.

The Emperor's New Code

The commitment to open source principles by the COSS company means that reality-based marketing is the only rational marketing strategy available.

The presence of free downloads and transparency (public forums and issue tracking) makes check-in-the-box and exaggerated claims counter-productive.

We have been publicly challenged several times on the Pentaho forums to 'back-up' claims that we have made, for example on our download counts (http://forums.pentaho.org/showthread.php?t=27515). This is part of the COSS model.

Challenges for COSS Companies

These are challenges that COSS companies face.

Trusting the Model

This is a big challenge. An open source model requires a leap of faith. Open source projects do not compete with each other commercially and so there is no situation that makes the openness, transparency, or 'early and often' principles anything but the 'right' thing. COSS companies are competing with commercial companies that will react against the COSS company. This make a commitment to transparency an (unnatural) act of faith.

Community

The community is the driving force of this model. Everything flows from the open source code and the community around it. Adapting to and enabling the community does not come naturally.

Metrics

A metric often tracked by COSS companies (and their financial backers) is how many downloads per month are converted into customers. The download metric is tracked for two reasons: to monitor community adoption and to monitor the conversion of downloads into customers.

There are some problems with both of these.

Adoption

The download total includes the downloads of all files by all people.

You can increase downloads by breaking the source, binaries or documentation into multiple files. This can increase the download total but does not help adoption.

Putting documentation into a wiki will allow the community to participate and collaborate on documentation but will decrease downloads but provide better service to your community.

Downloads can be used as a short-term qualitative measure within a single open source or COSS project provided that:

- Additional files are not made available for download and no files are removed from the download site.
- Alternative sources of information (web site content, FAQ's, wikis etc) are not significantly altered during the comparison period.
- The makeup of the community does not alter over the comparison period.

For these reasons comparing download counts between different open source or COSS projects is often meaningless.

Download counts themselves are an unreliable measure and using them to measure adoption is even more unreliable. By 'adoption' we mean the adoption of the software by a community of people. Knowing how many files were downloaded does not tell us how many people were doing the downloading. In addition just because someone downloads something does not mean that they like it. Even if they like it it does not mean

they are able to use it. The download metric also does not tell how many people are recurring downloaders from month to month.

Regular month-on-month growth of downloads is a probable indicator that adoption is growing but it is not a quantitative or reliable measure.

Conversion Ratio

The conversion ratio is a measure how well the COSS company is able to attact customers. This measure is a ratio of software downloads to new customers for a given time period. Ratios work best when they express the relative size of two quantities of the same type.

As shown above downloads are an unreliable measure. In addition downloads are files, customers are organizations: these are very difficult things to compare directly. A more useful measure would be 'what percentage of potential customers are being converted into actual customers?'. But from the Bee Keeper model we know that customers (and potential ones) are organizations, not people. People, not organizations download files. In addition download metric includes downloads from:

- Established community members.
- Employees of current customers.
- Analysts, media, students, competitors etc.
- Employees of potential customers.
- Employees of freeloaders (also potential customers).
- Downloaders (could be either one of the above or none of the above).

In addition more than one employee from a potential customer could be downloading files during the period.

Without being able to identify the number of downloads per person and which people are employees of a potential customer a meaningful metric cannot be derived.

There are ways that this information can be determined. Analysis of a person's web-site activity prior to them downloading files can help distinguish which group the downloader probably belongs to. If this can be determined accurately it does not matter which, if any, files the person actually downloaded. The COSS company can also offer content (e.g. white papers), trials, evaluations, recorded demos etc that enable potential customers to be identified with far more accuracy and so enables a more reliable metric to be used.

Raw download counts are useful only as a short term qualitative measure.

Naming

Pentaho, MySQL, Alfresco, JBoss, Zimbra all have same name for the COSS company and for the open source project as well as an integrated (branding and menuing) web sites.

Compare this with the relationship between Digium and Asterisk. Asterisk is an open source Voice over IP (VoIP) project that is administered by Digium, a for-profit enterprise. The Asterisk home page (http://www.asterisk.org) does not mention Digium's involvement other than a link to the Digium web site and a copyright notice in the footer. Even the 'About' page does not describe the full relationship. If you read the 'Company Profile' page on the Digium web site (http://www.digium.com) it makes it quite clear that Digium

wrote the seed code for Asterisk, created the open source project, are the primary developers of it, and that they monetizing it. I was a speaker on a panel with Mark Spencer of Digium at the February 2006 Open Source Business Conference (OSBC) in San Francisco when this issue came up. Mark said that the difference in naming between the two entities was deliberate and was done to give the community a stronger sense of its own identity. Whilst this is a valid aim the extent to which the relationship is hidden does not score very highly on the 'openness' scale. I am not criticizing Digium for this decision only recommending (and predicting) that as more COSS company's emerge they will choose the 'name the same' option.

Be clear, and open: 'Name the Same'. Use Name your company and your open source project the same.

This issue arises only if the COSS company is the originator of the open source project. If the COSS company is providing services for an open source code-base that they did not originator they should not name themselves to make it look as if they are.

Poker Players

The transparency and 'early and often' tendencies do not come naturally to people who have spent considerable time in the competitive world of proprietary software. It is very important for the management team to completely buy into the COSS model and the principles of open source otherwise conflict is inevitable. It is also important for everyone to understand that whilst their role might not seem that different other departments in the company are operating very differently.

The Misguided Altruist

There are potential community members who are considering contributing to an open sorce project but are wary of the COSS company's profit motive. I have seen numerous forum posts and discussion threads debating the philosophy of contributing to a professional open source project. For some purists this is a black-and-white issue and for others it is at least contentious. These posts often bring up the following conflicting points:

1) I'd prefer to contribute a 'pure' open source project because I don't want anyone to profit from my contribution.

2) But the professional open source projects have a lot stuff that is useful and valuable. Sometimes a lot more than the 'pure' open source alternatives.

The first statement turns out to be misguided. Lots of people profit from open source and from Apache in particular. Service companies and systems integrators build entire businesses charging for implementation services for, and support of, open source. Many software companies embed open source servers and components into their products to reduce their costs and increase their profits. Some of these companies contribute to open source by paying the salaries for full or part-time developers (committers) to work on improving the open source projects they use. Some of these companies contribute financially to the open source organizations. However the majority of these companies contribute nothing: they take the work of the open source project and all its contributors and they use it for their own financial gain. They are the free-riders. If you contribute to open source projects a whole hoard of people are going to profit - you just don't know who they are. There is nothing you can do about this, this is what karma is for.

Now lets look at the second statement. Professional open source software typically does have a lot of functionality when compared with pure open source software in the same arena. This is because the COSS functionality has to compete with proprietary software in a competitive market and so it has many full-time

engineers feverishly working on it to reduce any functionality gaps. These engineers are paid for by the customers of the COSS company. As a community member you are profiting directly from, and in many cases very significantly from, the COSS company's customer's inability to deal with open source and their need for 'whole product'.

The question comes down to how much value does the open source software bring to you? If a COSS company provides a stripped-down, hobbled, or bait-and-switch offering that doesn't provide you with much value, don't contribute to it. Why contribute to any project that does not provide value to you? However if open source software or COSS provides you with something that is of value to you give your time and contributions to that projects. If we all do that for forces of natural selection and evolution will take care of the rest.

I have read several comments posted on-line from people who claimed they would never contribute to a COSS project and gave their reasons why. We have taken those reasons and used them to refine our positioning and communication with our communities. In the end people who never intended to contribute, did. Thanks very much.

Managing Contributions

There are many ways that the open source, partner, and customer communities can contribute. Including:

- Feature contribution
- Software defect detection
- Defect fixes
- Design review
- Unique use cases
- Platform validation
- Documentation improvements, wiki edits
- Scalability and performance testing
- Translations
- Press releases, quotes, success stories

The COSS company needs the processes and infrastructure in place to enable these contributions to be accepted efficiently and effectively.

Resistance to the Model

There is lots of education that needs to occur before the mainstream market understands COSS business models. However in the mean time the proprietary vendors are not really able to take advantage of this window to instill fear, uncertainty, and doubt (FUD) into the market. Microsoft tried this and discovered that attempts to market against open source resulted in a higher level of interest in open source. This, and other interesting findings, were leaked from Microsoft in a series of emails now called the Halloween Emails. These emails, with commentary by Eric Raymond are available online: http://www.catb.org/~esr/halloween/

It is interesting to consider that Microsoft identified the threat from Linux 10 years ago and has not, in the intervening time, come up with an effective marketing campaign against it.

Summary

The commercial open source software model is a robust combination of an open source project and a complete sales, services, and Go-To-Market program. Not only does the model benefit from the advantages of each there are additional benefits for all participants:

- The open source community benefits from directly from the full-time engineering staff that exist because of the fee-paying customers.
- The customers benefit from the increased quality of the software, quality of design, and increased traction enabled by the open source community.
- The COSS company benefits by increasing its valuation when it meets the needs of both customers and open source community.

The model is powerful because the customers, partners, engineers, and open source communities are all self-motivated to behave in ways that are beneficial to themselves and, as a side effect, beneficial to all the others.

In terms of Moore's 'Chasm' a COSS company does not enable an open source project to cross 'The Chasm'. Rather the open source project must gain adoption in the open source realm and the COSS company must cross the Chasm in the commercial realm. The community adoption must occur first. In order to cross the Chasm the COSS company must remove the recognized barriers to the adoption of open source and build a 'whole product' for the mainstream market. Whilst the features in the software might not represent a discontinuous innovation I have shown that the COSS model represents a discontinuous innovation because it requires or enables a change in behavior by consumers, analysts, and participants alike.

To summarize the main points of the Beekeeper model in a comparison table:

	Open Source	Proprietary	COSS
Rate of innovation	Higher	Lower	Higher
Visibility into product design / implementation	Higher	Lower	Higher
Quality of software	Higher	Lower	Higher
Reliability of support	Lower	Higher	Higher
Reliability of roadmap	Lower	Higher	Higher
Ownership of solution	Higher	Lower	Higher
Availability of professional services	Lower	Higher	Higher
Availability of references and case studies	Lower	Higher	Higher
Ability to prototype and 'try before you buy'	Higher	Lower	Higher
Cost of license or subscription	None	High	Lower
Ability to customize software	Higher	Lower	Higher

As demonstrated by the Beekeeper diagrams and this table the COSS model, when implemented well, is a ideal combination of the methodologies, principles, and roles from open source and proprietary software development models. By combining these models carefully the advantages of each can also be combined to produce a result that is powerful and compelling.

To go back to one of the questions in the opening: Do all the principles, effects, and disruptive nature of open source apply to professional open source or does its commercial bias fatally pollute its own foundation? Fortunately for the COSS company if, and only if, managed correctly and with sufficient commitment to the underlying principles the benefits of an open source model can be leveraged by a professional open source company. The Bee Keeper analogy is a good representation of the COSS model and it can be used to model the high-level processes within the COSS company.

I have listed several challenges in the COSS model. Many of these challenges are related to making a firm and company wide commitment to the open source principles necessary for the model to succeed.

Open source has been described as the biggest paradigm shift in computing in the last 20 years. I hope that the information I have presented here shows the disruptive nature of open source / COSS models is due to the profound and fundamental differences that exist between the proprietary software development model and the open source / COSS models. COSS companies are a lot of fun to work at because of this. I don't know of anyone working in a COSS company today that wants to go back to a proprietary vendor. I know lots of people at proprietary vendors who would like to move out.

The COSS model is sufficiently open source to be disruptive, productive, and fun. Much evidence indicates that it is commercial enough to be sustainable.

Thanks

The Pentaho team: Especially Richard Daley, Doug Moran, Marc Batchelor, Lance Walter, Julian Hyde, Matt Casters, Anthony deShazor, Jonathon Seper and extra-specially Thomas Morgner for his careful consideration and significant feedback.

The Pentaho, Mondrian, JFreeReport and Kettle communities.

Friends and peers: at JBoss, Red Hat, MySQL, Alfresco.

Bernard Golden: author of 'Succeeding With Open Source', CIO Magazine blogger and CEO of Navica

Eric Raymond: author of 'The Cathedral and the Bazaar'

Geoffrey Moore: author of 'Crossing the Chasm'

This document prepared on a MacBook Pro using NeoOffice (OpenOffice port for OS X) and Keynote.

References

Six barriers to open source adoption, a Dan Farber summary of a Ray Lane keynote, ZDNet:

http://techupdate.zdnet.com/techupdate/stories/main/Six_barriers_to_open_source_adoption.html

Cathedral and the Bazaar, Eric Raymond:

http://www.oreilly.com/catalog/cathbazpaper/

http://www.catb.org/~esr/writings/cathedral-bazaar/

Crossing the Chasm, Geoffrey Moore:

http://en.wikipedia.org/wiki/Crossing_the_Chasm

http://www.amazon.com/Crossing-Chasm-Marketing-High-Tech-Mainstream/dp/0066620023

Coverity: http://scan.coverity.com/

US Department of Commerce's National Institute of Standards and Technology 2002 Report: http://www.nist.gov/public_affairs/releases/n02-10.htm

Navica Newsletter on Community: http://www.navicasoft.com/Newsletters/May%202006%20Newsletter.htm

OpenOffice: http://www.openoffice.org

Pentaho Corp: http://www.pentaho.org

Red Hat Inc: http://www.redhat.com

Time Magazine: Person of the Year 2006: http://www.time.com/time/magazine/article/0,9171,1569514,00.html

Wikipedia on Web 2.0: http://en.wikipedia.org/wiki/Web_2.0

NeoOffice: http://www.neooffice.org

The Hitchhiker's Guide to the Galaxy, Douglas Adams