pentaho™
open source business intelligence™

The Beekeeper

A Description of Commercial Open Source Software Business Models

James Dixon
Chief Geek / CTO, Pentaho

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 an abridged version of The Beekeeper.
The full version and discussion forums can be found at http://www.pentaho.org/beekeeper
Email: beekeeper@pentaho.org

Introduction

There is much media coverage and water-cooler talk about open source software and commercial open source software companies but these companies and their business models are so new that most people do not have direct personal experience of them and consequently many people have numerous misconceptions about how they work. In almost all cases I think people underestimate not only the potential of the commercial open source software business model but also how much it differs from the proprietary/ commercial software business model and how much it affects the roles and functions within the organization.

In 2004 I co-founded Pentaho to provide Business Intelligence (BI) under a commercial open source software business model. After spending the last few years immersed in commercial open source software I am sharing my thoughts about the workings of this business model.

There are various kinds of commercial open source software models and various companies. I will describe the different models briefly but as this document is based on my experience at Pentaho it naturally applies more to those companies that operate with a similar model (where the company is the main source code contributor) such as that of MySQL, Alfresco, JBoss, Digium, Zimbra etc.

Why Commercial Open Source Software?

Open source is a model of software development that has been growing since the 1970's. It is a very effective model at producing high quality software. As the number of open source projects grew, along with the scope of what could be done with it, it attracted the attention of IT organizations, systems integrators, software vendors and other commercial consumers. These organizations identified a number of barriers that make it hard, and in many cases impossible, for them to adopt open source. The primary barriers are:

1. Lack of formal support and services.
2. Velocity of change.
3. Lack of roadmap.
4. Functional gaps.
5. License types.
6. Lack of endorsements by independent software vendors (ISVs).

These barriers are real, rational, and with the exception of 'Feature Gaps', they are all risk-related.

The Principles of Open Source

In order to understand commercial open source software companies there certain facts about open source that need to be understood.

Free means Freedom Not 'Free Stuff'

Open source is not free. In 1998 the term 'open source' was coined to replace the term 'free software' because many people assumed 'free' to mean 'zero cost' whereas it was always intended to mean 'freedom'. If you consider the barriers to adoption of open source listed above it becomes clear that the notion of 'free stuff' is false. It is true that an organization could use open source software and support itself by hiring technical staff with the necessary skills to:

- Evaluate and select the most suitable open source software or software distribution.
- Integrate and embed the open source software into internal systems.
- Fix any critical defects that are found.
- Decide which patches and releases to migrate to and ensure migrations between versions are free from problems.
- Participate in the communities to ensure the direction that the software is taking suits the organization.
- Train any users or new technical staff.

Organizations have the freedom to do all these things but they should not consider fulfilling these needs to be of zero cost.

If you accept that open source software is not a zero cost solution you must then accept that these costs can occur in the form of time (internal man hours) or money (given to some external organization) or both.

In addition to the costs above there are also risks to be managed. In fact if you look at the commonly listed barriers to the adoption of open source software you will find that most of them are related to some kind of risk and not to any kind of cost. It is difficult for most organizations to manage all of these risks themselves: the number of people and the range of skills required to do so is prohibitive. For small organizations or low-risk projects these risks can be tolerated but for larger projects risk management is a significant issue.

This leads to the basic rational of commercial open source software: to provide organizations with an alternative to 'going it alone' with open source.

Principle of 'Openness'

Accepting feedback through a web page and providing mechanisms for people to report defects is one thing. Allowing everyone else to see that feedback and all those defects is another, much more uncomfortable, step. Allowing everyone to see the source code so they can review it and try it is also an act of faith. Providing a public forum where people can openly criticize and contribute to the design and implementation of the software is another act of faith. These are difficult behaviors for a proprietary organization to accept but open source projects rely on them. If the project administrators do not provide mechanisms for people to communicate openly amongst themselves there can be no community.

The availability of the design documents and source code enables the community to provide peer reviews and to use the software for their purposes and provide feedback on the quality.

Principle of 'Transparency'

Transparency is the ability of the community to see what's going on. This involves:

- A published road map so they know where the administrators plan to take the project.
- A public defect tracking system so they can report and review defects.
- Published design documentation.
- Communication about schedules and hurdles.

Without transparency it is hard to grow a community.

Transparency and openness are not the same thing. A glass door is transparent but whether it is open or not is a different matter. Transparency is the ability to witness the inner workings, openness is the letting outsiders 'in' so they can participate.

Principle of 'Early and Often'

This is the philosophy of making information available in its earliest drafts and updating it often. This includes, but is not limited to, the source code of the software. Zipped archives of the source code are available for every open source project. Many projects go further and have a public repository where the current code is always available. As the developers change the source code and check it in, it is available to anyone who wants to review it or use it.

The principles of open source compound and combine together. Each one is a leap of faith.

In order to be successful all the principles must be observed: merely letting someone view source code, or giving someone a free evaluation license does not have the same disruptive power of the open source model. The tendency of the open source model to resolve design defects early in the software development cycle only occurs if all three principles are applied.

Expectation of 'Community'

Every generation grows up with a new set of expectations. For example my parents had the expectation that they needed to buy encyclopedias if they wanted access to reference knowledge at home. I grew up with the expectation that I could get Encarta(tm) on a CD and later that I could search the Internet. My son is growing up in the Web 2.0 era where he can participate by submitting Wikipedia entries.

When it comes to open source the web site, source code, roadmap, defect tracking system, and forums are the 'project' and the community participates in the project. The fact that the source code and roadmap are available is a result of 'openness'. The fact that the defect tracking system and forums are available is a result of transparency. The fact that the design and initial code is available is a result of 'early and often'. It is these principles and the results of using them that ultimately attract and retain the community.

The community is a byproduct of the project. The project is a byproduct of the open source principles.

Participants in open source have an expectation of a community and the development model and commercial open source software company relies on it.

The Beekeeper

Commercial open source software (COSS) companies exist as an exchange system between two sets of consumers: an open source community (motivated by mutual contribution) and a mainstream market (motivated by economic rewards). Organizations in need of support, services, training etc contribute financially for those services as paying customers. That money is used by the COSS company to pay for full-time resources (engineers, product managers etc.) whose efforts (the majority, if not all of it) end up as open source software, freely available to an open source community. The open source community contributes to the software by helping improve the design, functionality, quality, translations, and documentation of the software. The improved software attracts more customers and the cycle continues, hopefully perpetually. In this model all three parties gain:

- The community gains open source software they can use for their own purposes. This software has more functionality and more resources than a 'pure' open source project could provide. In this way the community profits directly from the COSS company and its customers.
- The customers gain higher quality software at a better price. The customers profit from the open source community's ability to produce high quality software.
- The COSS company gains by growing and increasing its valuation as a result of keeping both sets of consumers content.

My analogy to this is 'The Beekeeper'.

The Beekeeper creates an environment that is attractive for bees: accommodation and a natural, food-rich habitat. The bees do what they do naturally and make honeycombs. The Beekeeper sells the honey and bees-wax to his customers and uses the money to grow his bee farm.

The analogy goes a little deeper.

- Bees can fly and so have the opportunity to leave the bee farm if they decide to. So the Beekeeper must tend to his bees. The Beekeeper has very little control over his bees and has no ability to direct them to do his bidding. Likewise the community can desert, or even worse fork, an open source project if they so desire (see the Wikipedia entry on Fork). So the COSS company must keep their community happy. The COSS company cannot rely on the community to follow any directive or schedule it might have. Bees can sting. Community members can publicly object or criticize the COSS company on its own or other web sites.
- The growth of the bee farm depends on how much honey and wax the Beekeeper can sell to direct customers (passers by), channel partners (the grocery store), and OEM's (the candle maker). How much he can sell depends partly on his business skills and partly on how much honey and wax he has. How much honey and wax he has depends on how many bees he has. How many bees he has depends on how much honey and wax he let the bees keep for themselves. In order to achieve maximum growth the Beekeeper must grow both his bee population and his customer base at the same time. Likewise a COSS company must perform a balancing act and grow the community and customer base together.
- The bees and honey came first. There is no chicken-and-egg dilemma here. The Beekeeper invested time and effort in his beehives before he had anything to sell to his first customer. The longer he spends building his bee population before he starts selling the quicker it will grow. Likewise the COSS company must build a community that helps create the software before they can engage in the commercial world. The longer the COSS company can focus on adoption before having to worry about revenue the better it will be.
- Each bee hive has a queen. In order to start a new hive the Beekeeper needs to attract a queen and enough of her bees to make the hive viable. Likewise open source projects often have a single founder or administrator that is the main leader of the project. Open source projects can be 'acquired' or 'merged' if the project leader and prime contributors are convinced that the move is beneficial to the project.
- The customers don't want to deal with the bees. A single bee or even a swarm of bees cannot directly meet the needs of any of the Beekeeper's customers. It is the work of the Beekeeper that turns the efforts of the bees into products that the customers desire. The honey in the jar is the

same honey that was in the hive, but the customers will only pay for it in the jar. Likewise the commercial customers don't want to deal with open source. They want 'whole product'.

- The customer's cash is no good to the bees. A crisp bank note or a pile of coins is of no use to a bee but the Beekeeper can use that money to buy hives, bee feed, or bee medication (those varroa and tracheal mites can be problem). Likewise the COSS company's customers do not directly help the open source community. Its only when the COSS company uses that money to pay for engineering resources to improve the open source software that the community benefits.
- Each individual bee makes a small contribution to the system. It takes a large number of bees for a successful outcome. Likewise the contribution of the COSS company's community are vital to the model but the individual contribution of most community members is small.
- Customers are not bees, bees are not customers, and you cannot convert one to another. It is impossible for a bee to pay for a product, for a customer to make honey. I call this the Consumer Dichotomy. It seems that this might not apply to the COSS model and so is limitation of the analogy but it is not. The customer in the Beekeeper analogy equates to the organization that is paying for the 'whole product' from the COSS company. The bees equates to the community of individual employees and enthusiasts. Consider:
- The COSS company often knows a community member for a long time before finding out who they work for. Sometimes they never find out. Community members often remain part of the community as they move from employer to employer.
- For business software the sales contract is between an organization (customer) and the COSS company. If an employee in a role that is a touch point between the customer and the COSS company leaves their employer, a new individual is nominated to fill the role, and the old employee is no longer a part of the relationship but the organization is.

Customers are corporations, the community are people. They have very different needs that need to be met.

This inherent distinction between customers and community is important when it comes to attempting to turn members of the community into customers. The sales and marketing groups within a COSS company need to be aware that, in most cases, it is not possible to convert a community member into a customer. However is it possible to convert the employer of a community member into a customer. Slamming community members with a marketing pitch is unlikely to achieve this.

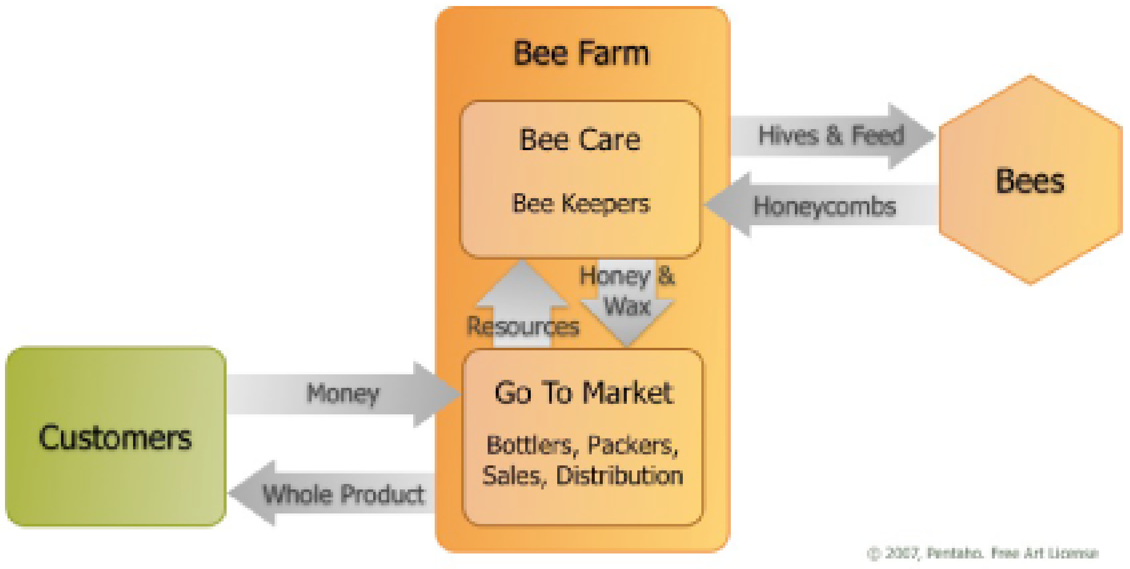
Community members have the potential to persuade their employer to become a customer. The members themselves typically have no budget or no control over how the budget is spent. A COSS company needs to educate its members on the services that are available and the long term advantages to everyone if those services are used. The COSS company needs to find a way of presenting this and enabling members to present it to their employer without de-valuing the capabilities of the community member.

The Beekeeper Diagrams

The series of diagrams that follow explain the model above by comparing the Beekeeper, open source, proprietary, and COSS models at a high level.

The Beekeeper

This diagram shows the high-level workings of a bee farm.



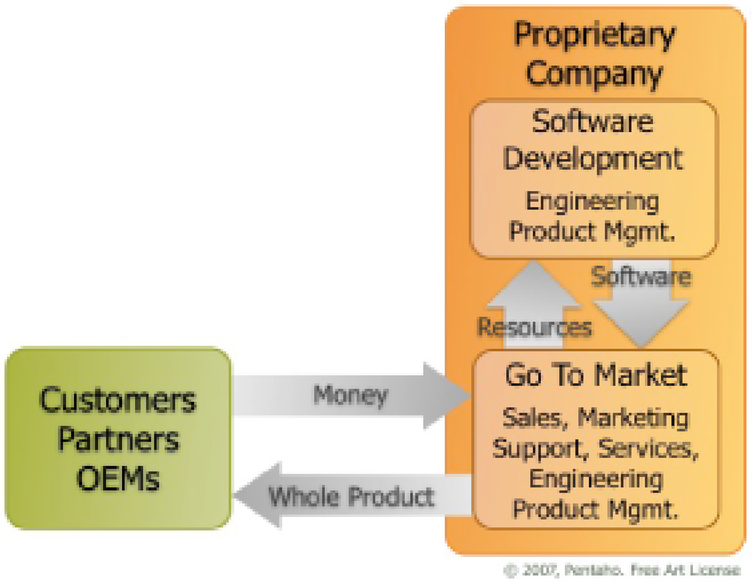
As described above the Beekeeper provides for the bees.

Notice that there are multiple roles that need to be filled within this model and that some of those roles are focused on the bees, whereas some are focused on getting the honey and wax into the hands of customers.

Notice also that there is no interaction between the customers and the bees.

Proprietary Software Model

This diagram shows the high-level workings of a proprietary software development company.

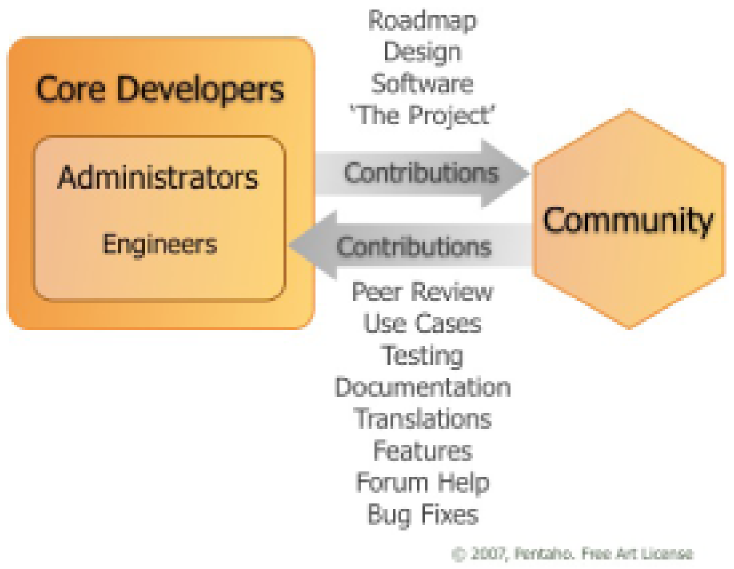


This diagram is a very simplified representation of the real-world workings within a proprietary software development organization. There are obviously many communication paths, touch-points, and cross-functional deliverables that occur that are not shown on this diagram. However it is useful if we view the model this way and compare it with the diagrams of the open source and commercial open source software models.

- Engineering has two roles in this model: first to create software, and secondly to participate in the 'Go To Market' program.
- Product Management 'owns' the product roadmap and has the responsibility of creating it by collating the requirements of Sales, Marketing and customers. They also act as a buffer between Engineering and these groups. This is done for two reasons: to keep engineers focused on writing software, and to control the flow of information from engineers to customers. Product Managers also describe how the features are to be turned into 'whole product'.
- Note that the roles of the Sales, Marketing, Support, and Services departments are focused on the customers.
- It is the 'Go To Market' program that creates the 'whole' product that mainstream customers require. Engineering does not run the 'Go To Market' they are (usually reluctant) participants.
- The customer is not very involved in the process of creating the software.

Open Source Software Model

Looking at an open source model in the same way:

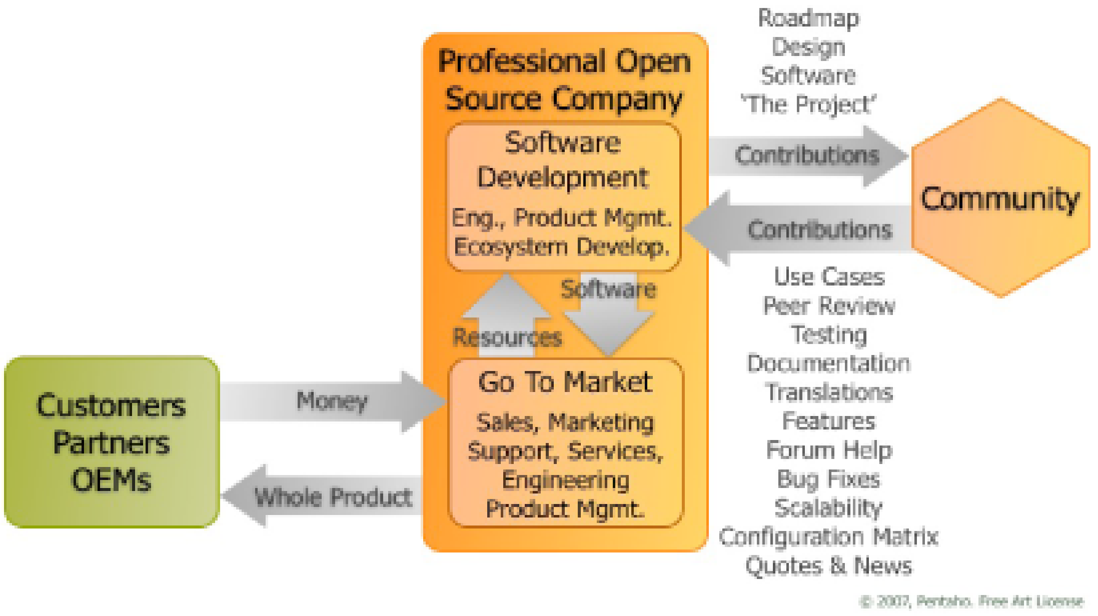


- The project administrators are typically the engineers doing much of the development and design and also setting the project roadmap.
- The road map is owned by the project administrators. In this respect they are partially fulfilling the role of Product Managers. The project administrators do not perform the other, 'whole product' focused, roles of Product Managers.
- The community participates in many roles and tasks involved in designing, implementing, and testing the software.
- There is no 'Go To Market' process in this model. This is why the barriers to the adoption of open source listed above exist.
- There is no marketing in this model so open source projects gain mind-share through technical articles, blogs, and word-of-mouth.

A few people have a negative reaction to the open source concept because they think it is in some way socialist or communist. This is not the case: open source is effective because the motivations of all participants are naturally aligned (rather than artificially aligned with financial reward as a 'pseudo-motivation').

Commercial Open Source Software Model

In the form of a diagram COSS model looks like this:



Notice that the COSS model includes all the features of the proprietary model and the open source model. Notice also that the COSS company model mirrors the Beekeeper model above.

The important points from this diagram are:

- The closest ties between the COSS company and the community are through the Engineering (includes development and quality assurance) and Product Management (PM) groups. This is a natural extension of the open source model diagram (see above). These roles are 'honey-focused'. There are interactions between other departments of the COSS company and the community but they do not happen on the daily and hourly basis that exists within engineering and PM. As we shall see later the interaction between the engineering and PM groups and the community is a good deal more dynamic and frequent than is implied in this diagram.
- In the COSS model the roles of Engineering and PM become much closer aligned to each other because between them they are sharing the role of the project administrators of the open source model. PM no longer exists in a buffer zone between Engineering and the consumers. In fact the engineers usually have more routine contact with the community than the product managers do.
- Notice that the roles of Engineering and Product Management are significantly different compared to the proprietary model. In the COSS model the engineers have continual, direct communication with the community (open source consumers). In the proprietary model direct contact between Engineering and the consumers is not a major part of the model (an understatement).
- The Sales, Marketing, Support, and Services departments are market-focused. Their roles are oriented towards the customers and are involved in an ongoing program of creating a 'whole product' around the software and getting it to market.

- There often exists a community liaison role within the COSS company whose job it is to focus on community growth and satisfaction.
- Note that, at high level, the roles of the Sales, Marketing, Support, and Services departments are very similar between the COSS and proprietary models. This is because they are focused on the 'whole product' aspect of the model. A prospective customer should not have to learn about open source in order to become a customer of a COSS company. For example I do not need to know how honey or maple syrup is made in order to buy either of them. The COSS company sales and marketing materials should neither hide their open source model nor require understanding of it by the market. Although the role of marketing is not fundamentally different in this model the strategies, tactics, and techniques used to market commercial open source software are different primarily because:
    - A mass marketing approach is used instead of a high-touch, expensive enterprise approach
    - A prospective customer's transition from the marketing domain to the sales team occurs significantly later in their experience.
- Engineering and Product Management are involved in both aspects of the model. Their roles in the 'Go To Market' program are not significantly different from the corresponding roles within a proprietary organization.
- The Go To Market program is highly inefficient if it has to react to the contents of the software after it is done. That is to say the participants in the Go To Market program need to involved (if only passively) in the creation of the software to avoid significant delays in creating the 'whole product'.

The deliberate simplification of the diagrams above shows something important. If you reduce the workings of a proprietary software company and an commercial open source software company down to the fewest number of lines and bubbles it shows fundamental differences between the two.

By comparing the three diagrams you can see that the COSS model is a perfect combination of an open source project and a proprietary software vendor. I use 'perfect' in the mathematical sense: there is nothing left out and there is nothing in addition to the two other models.

Some people assume that the COSS model is flawed because the COSS company does not have direct control over its resources. This is not true with the COSS model used by Pentaho, MySQL etc. The only resource we cannot ***directly*** influence are our community who perform software-oriented roles for us at a scale and capacity that is orders of magnitude better than any proprietary organization could do on its own (keep reading). Community members do not participate in the 'Go To Market' program and so are not involved in the creation of the 'whole product' that customers require.

commercial open source software = open source software project + 'whole product' program

Summary

The COSS model is a robust combination of an open source project and a commercial open source software company backing that project, and a complete 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 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.

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.

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 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.

More Information

There is a discussion forum and a longer, significantly more in-depth, version of this document at:

http://www.pentaho.org/beekeeper

email: beekeeper@pentaho.org