



## **The Bees and the Trees**

The Beekeeper Model of Commercial Open Source

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 part one. Part two and discussion forums can be found at <http://www.pentaho.org/beekeeper>  
Email: [beekeeper@pentaho.org](mailto:beekeeper@pentaho.org)

Distributed under the Free Art License

Free Art License 2007-2009 Pentaho Corporation. Redistribution permitted. All trademarks are the property of their respective owners.

For the latest information, please visit our web site at [www.pentaho.org](http://www.pentaho.org)

## Introduction

In 2004 I co-founded Pentaho to provide Business Intelligence (BI) under a commercial open source software business model. After spending a few years immersed in commercial open source software I created the Beekeeper model as a way to share my thoughts about the workings of this business model. Two years have gone by and I have expanded the model based on the encouraging feedback I got from many people.

There are various kinds of commercial open source business models and various companies using variations of those models. The first version of the Beekeeper model only addressed commercial open source companies using a 'single vendor' model where members of a single commercial organization writes most of the software. This second version includes models for companies that write little or none of the software and who provide services and/or support for software that they do not author. It also includes analogous models for proprietary software companies and pure/organic open source projects.

Companies using the single-vendor commercial open source models include MySQL, Ingres, Compiere, Open Bravo, Liferay, SugarCRM, Pentaho, Mule, OpenNMS, Alfresco, JBoss, Digium, and Zimbra. In this document I will refer to this collection of business models as the 'single-vendor' model to indicate that the direction and development of the software is funded by a single company.

Companies using the support or services commercial open source model include Optarus, Sourcesense, OpenLogic, and SpikeSource, Open For Business (OFBiz).

## Why Commercial Open Source Software?

Open source is a set of principles and a philosophy for software development that has been growing since the 1970s. 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 'Functional Gaps', they are all risk-related.

## Whole Product

It is interesting to consider these barriers in the context of Geoffrey Moore's 'whole product' concept from his 'Crossing the Chasm' work. In essence the 'whole product' is everything needed by the customer in order to have a compelling reason to buy to product. Apple's iPod provides a good example: the iPod music player is the core product but the 'whole product' includes accessories, the iTunes music store (backed by agreements with the music industry) and a lifestyle-oriented marketing campaign. It is the combination of all of these that has made the iPod the dominant product in the market (it was certainly not the first). In order to overcome Lane's barriers, the mainstream IT market needs open source software to be delivered as whole product: an out-of-the-box, easy-to-consume, packaged-and-delivered, risk-free solution. Commercial open source exists to do just that.

## Commercial Open Source Models

There are a number of different business models that can be described as 'commercial open source'. These models vary in pricing, structure, offerings, and start-up costs. The 451 Group has done a good job of describing the many different models that are currently being tried: [http://www.the451group.com/caos/caos\\_detail.php?icid=694](http://www.the451group.com/caos/caos_detail.php?icid=694)

## 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 as in 'Freedom' Not 'Zero Cost'

Open source is not free. In 1998 the term 'open source' was coined to because many people assume any use of the word 'free' to mean 'zero cost' and so assume 'free' software the same as shareware. In the context of open source the freeness relates to the freedoms that are provided by the licenses.

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 rationale 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.

## Re-Use and Modularity

Engineers in general are obsessed with efficiency and architectural elegance. Software engineers are no exception and this is evident in the open source movement in different ways:

- Re-use: One of the most inefficient and inelegant aspects of proprietary software development is that every software company has to develop (pay in time) or license (pay in money) everything that goes into the product. In the proprietary model there is little re-use of common or commodity components. A comparable analogy would be if every house builder manufactured their own screws, bolts, and cables. In open source software the incentive is to re-use rather than re-create, because it more efficient.
- Simplicity: In order to be 're-useful' you have to design simple software. You have to write it to do a specific thing very well and avoid the tendency to add things that are peripheral. A web server should be a web server, it should not also be an email server, and a file server, and a fax server. As a really bad example you can buy an external battery for an iPhone that is also a laser pointer and a LED flashlight.

## 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.

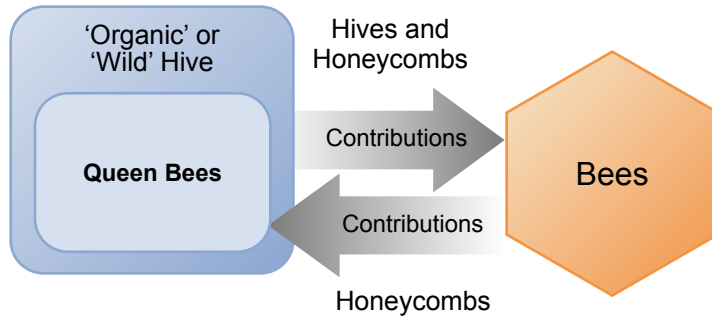
## The Models

There are four software development models described below. In each case there is an analogous model that helps to describe and compare the models in easy to understand terms. These model are:

- The Wild Hive Model for Open Source Projects
- The Maple Syrup Farm Model for Proprietary Software Companies
- The Beekeeper Model for Single-Vendor Commercial Open Source
- The Honey Gatherer Model for Services/Support commercial Open Source

## The Wild Hive Model for Open Source Projects

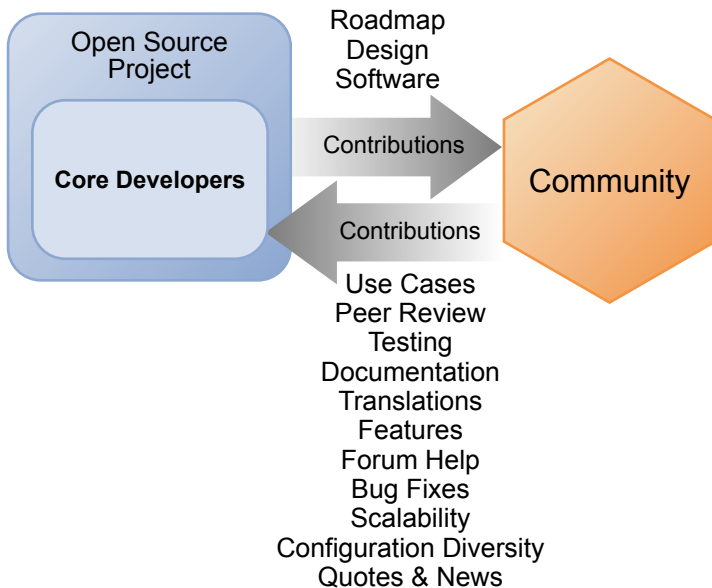
Consider a beehive in the wild:



James Dixon, Pentaho, 2007, Free Art License

The hive is started by a queen bee. The hive grows over time in terms of the number of bees, the size of the hive, and the amount of honeycomb in the hive. Each bee performs a function within the hive and collectively they all benefit, but (at least theoretically) any bee is free to fly away to another hive, as is the queen. Bee can also (in this model) participate in multiple hives at the same time, flying from one to another at will.

An open source project can be described in a similar way:



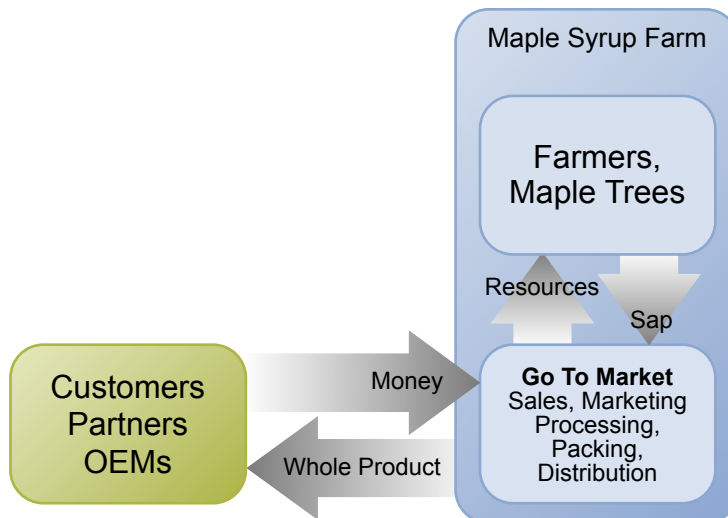
James Dixon, Pentaho, 2007, Free Art License

- The core developers are often the founders of the project. Typically they do much of the development and design and also set the project roadmap. They act like queen bees.
- The community participates in many roles and tasks involved in designing, implementing, and testing the software. The community benefit from the project and the project benefits from the community. The community are like bees. Community members frequently contribute to multiple open source projects at the same time, often in different roles.
- There is no 'Go To Market' process in an open source project. This is why the barriers to the adoption of open source listed above exist. Open source projects create software, they do not create 'whole product'.

- There is no specific marketing role in this model so open source projects gain mind-share and attract community members through technical articles, blogs, and word-of-mouth.
- In some instances the creation of the software is democratic in nature, but in most cases (at least with small projects) the project is run in a benevolent dictatorship fashion by the core contributor(s)
- No matter how few or many contributors there there is little attention paid to intellectual property, license adherence (fulfilling the license obligations of other open source projects that are used), or patent issues.

## The Maple Syrup Farm Model for Proprietary Software Companies

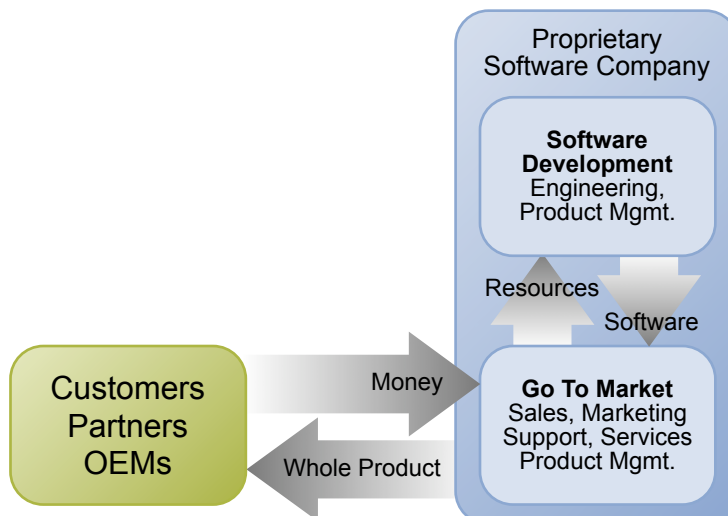
In a proprietary software company the raw material, software, is the same as with an open source project but the process by which it is created is very different so we need a different model. We use the Maple Syrup Farm model.



James Dixon, Pentaho, 2007, Free Art License

In this case the raw material is the sap from maple trees. The source of the sap (the trees) are fixed assets with no active involvement in the process. As noted above bees have free-will and they participate because they choose to. The maple trees have no free will and no choice. Moving a tree from one farm to another is a permanent and costly exercise. This is very different than the mobility of the bees in the Wild Hive model.

The Maple Syrup Farm model compares nicely to the proprietary software model:



James Dixon, Pentaho, 2007, Free Art License

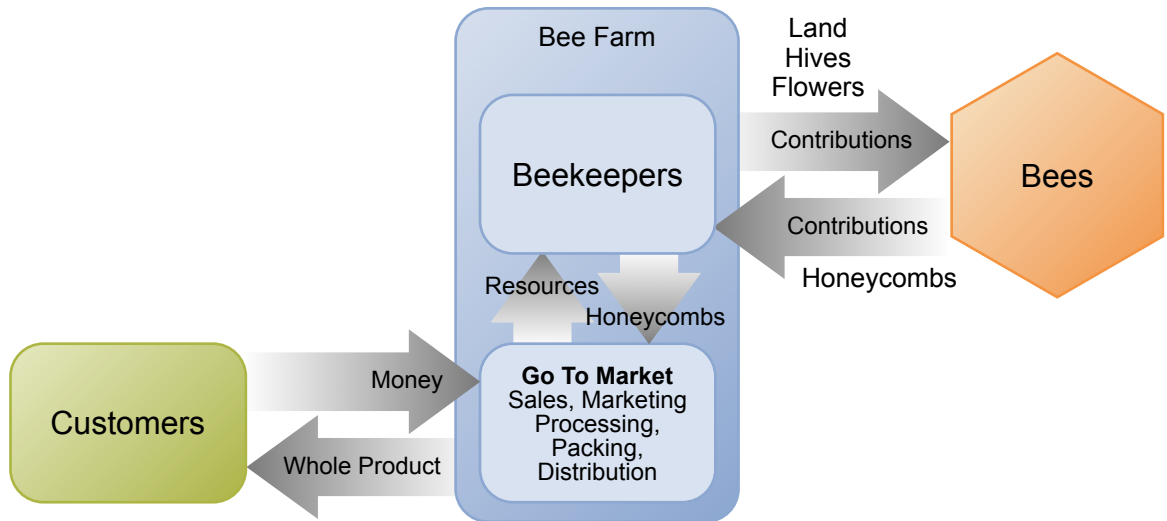
In the proprietary model:

- 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 delivering the 'whole product' to 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.
- Engineers (the trees) can only be involved at one company (farm). Engineers are explicitly restricted, through contracts, from working for any other company.
- Intellectual property and license issues are actively monitored by the organization.

The open source and proprietary models have the same raw materials and both involve software engineers but they are clearly very different from each other. The first has a community but produces only raw material (software), the second has no community but produces 'whole product' from the raw software. The Wild Hive and the Maple Syrup Farm models can be compared in a similar way: the raw material is a sugary, sticky substance in both cases but the processes and outputs are very different.

## Beekeeper Model for Single-Vendor Commercial Open Source

Now lets turn to the Beekeeper Model:

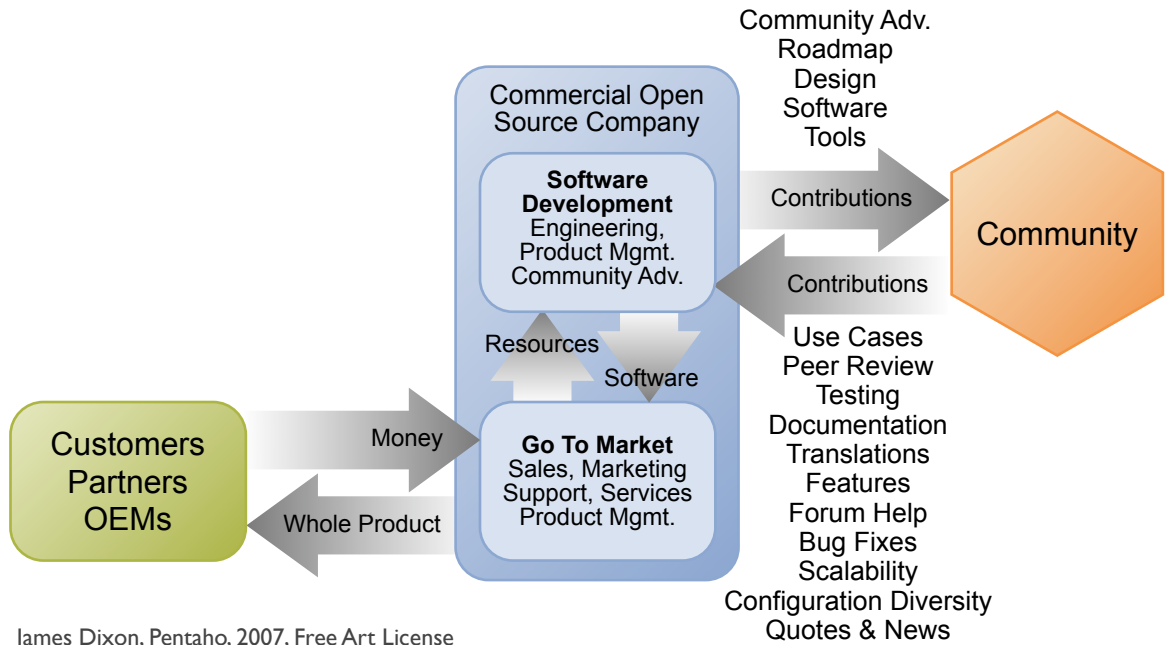


James Dixon, Pentaho, 2007, Free Art License

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 honeycombs are processed and the resulting products, honey and bees-wax, are sold to customers the money then used to grow the bee farm. 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.

The single-vendor commercial open source model looks very similar:



James Dixon, Pentaho, 2007, Free Art License

Single-Vendor Commercial Open Source software 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, and training etc contribute financially for those

services as paying customers. That money is used by the 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 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 company gains by growing and increasing its valuation as a result of keeping both sets of consumers content.

The Beekeeper and Maple Syrup Farm models work well because the end products (bottles of honey and maple syrup) are very similar and are distributed, packaged, and sold in similar ways. You do not need to know how honey or maple syrup is made in order to buy either of the end products. It is obvious that the big difference is the way the raw materials are created. In the beekeeper model the raw material is generated by a mutually beneficial partnership between the beekeeper and the bees.

In addition:

- The closest ties between the company and the community are through the engineering (includes development and quality assurance) and product management (PM) groups. These roles are 'honey-focused'. There are interactions between other departments of the company and the community but they do not happen on the daily and hourly basis as with engineering and product management.
- In this model the roles of Engineering and product management become much more closely aligned to each other because between them they are sharing the role of the project administrators of the open source model.
- There is no buffer zone between engineering and the consumers of the software. In fact the engineers have more routine contact with the community than anyone else in the organization. The roles of engineering and product management are significantly different compared to the proprietary model. In the single-vendor commercial open source model the engineers have continual, direct communication with the community (open source consumers). In the proprietary model direct contact between engineers and the consumers is not a major part of the model (an understatement). In fact contact between engineers and consumers is usually kept to a minimum.
- 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 is usually a community liaison role within the single-vendor commercial open source company whose job it is to focus on community growth and satisfaction, and to assist with incoming contributions. Open source projects typically do not have this role.
- 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 be involved (if only passively) in the creation of the software to avoid significant delays in creating the 'whole product'.
- Since the company creates most of the software they are in a position to monitor intellectual property, license, and patent issues. Many single-vendor companies have contribution agreements that include copyright assignment so that any IP-related issues can be cleanly resolved.

As you can see there are numerous similarities between the Beekeeper and single-vendor commercial open source models:

- 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, if they are unhappy with the project, can leave the project or even worse: fork it (see the Wikipedia entry on [Fork](#)). So the company must keep their community happy. The 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 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 OEMs (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 single-vendor commercial open source 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 products the quicker it will grow. Likewise the single-vendor commercial open source company must build a community that helps create the software before they can engage in the commercial world. The longer the company can focus on adoption before having to worry about revenue the better it will be.

In many cases the up-front investment necessary to create a usable seed (bee farm) is substantial and beyond the capacity of a volunteer or self-funded team (in a timely manner). This is the entry point for venture capitalists (VC), and from 2004 onwards VCs heavily funding commercial open source companies. It is more attractive to invest in a disruptive commercial open source company than to invest in a new closed-source proprietary software company that has to compete with open source.

- 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 customers' 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 company's customers do not directly help the open source community. It is only when the 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 contributions of the community are vital to the model but the individual contribution of most community members is small.

In my definition if a group of people associated with a single commercial organization develop or maintain a substantial part of the software (60%) and/or they provide the majority of resources (download site, wiki, other online tools etc) they are operating a Beekeeper model. Within this category, however, there are significant differences in how contributions and software development are managed. If you are looking at code contribution only:

- OpenNMS is very democratic and open about the software development process
- MySQL is notorious for being opaque and difficult to contribute code to

If you broaden the scope to include use cases, peer review, testing, documentation, translations, forum help, bug fixes, scalability studies, configuration diversity and quotes & news the contribution story is more open across the board.

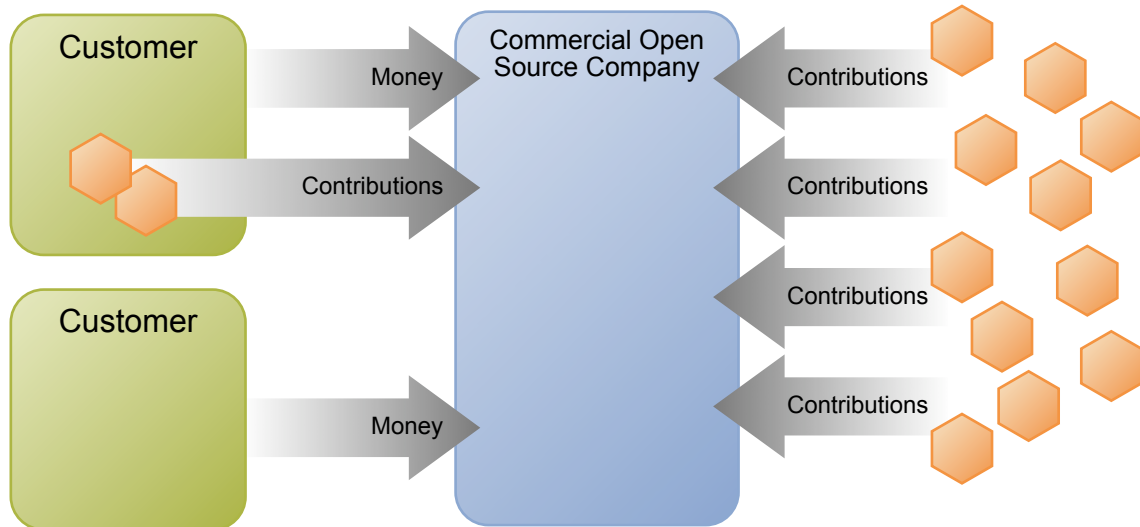
To provide some context: in my experience contributing to Apache Foundation projects (wild hive) has been like giving candy to fog: you stand there and hold out your offering and it falls to the floor in silence. In contrast contributing to Alfresco, JBoss, and MySQL has been easy and rewarding. I still use Apache software for many things but I have given up trying to contribute. Philosophically, the Apache Foundation projects are the most open of all of these, however, in practice, the effort required to contribute is beyond my endurance.

To my mind, you need to look at all the barriers to accepting contributions of each type. The barrier could be philosophical, organizational, procedural, legal, resource bound, or something else. From the perspective of a contributor, unless they are told, they might not be able to identify the barrier.

## Community and Customers

**Customers are corporations, the community are people. They have very different needs.**

In the Beekeeper Model 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 and for a customer to make honey. I call this the Consumer Dichotomy. It seems that this might not apply to the commercial open source 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 commercial open source company. The bees equates to the community of individual employees and enthusiasts.



James Dixon, Pentaho, 2007, Free Art License

In the diagram above notice that all the customers have a monetary relationship with the commercial open source company. Notice also that the community members provide contributions. Some of the community members are employees of customers, but many are not, and some customers have no community members on staff.

Consider:

- The 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 single-vendor commercial open source company. If an employee in a role that is a touch point between the customer and the 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 still is.

The community/customer discussion becomes complicated when community members and customers get mixed together. This happens frequently. But in terms of the model, the customer is still an organization, and community members are still individuals, its just that some of the community members work for some of the customers.

This 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 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.

Note that, at high level, the roles of the Sales, Marketing, Support, and Services departments are very similar between the single-vendor commercial open source 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. The sales and marketing materials should neither hide their open source model nor require understanding of it by the market. The business model of the commercial open source company is a competitive advantage that should be presented to the customers, but if they are not interested or do not understand, they should still be able to purchase offerings easily. Although the role of marketing is not fundamentally different in this model the strategies, tactics, and techniques used to market the 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.

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 commercial open source company needs to educate its community members on the services that are available and the long term advantages to everyone if those services are used. The 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.

When we compare the open source project model and the single-vendor commercial open source model we notice that the commercial model contains all of the people, processes, and items of the open source model. There are no elements missing. The difference is that the single-vendor commercial open source model includes a Go To Market program produces and a 'whole product'. The same differences exist between the Wild Hive and Beekeeper models.

When we compare the proprietary model and the single-vendor commercial open source model we see that the single-vendor commercial open source model contains all of the people, processes, and items of the proprietary model. Again there are no elements missing. The difference is that the single-vendor commercial open source model includes a community and all of the contributions that the community make. The same differences exist between the Maple Syrup Farm and Beekeeper models.

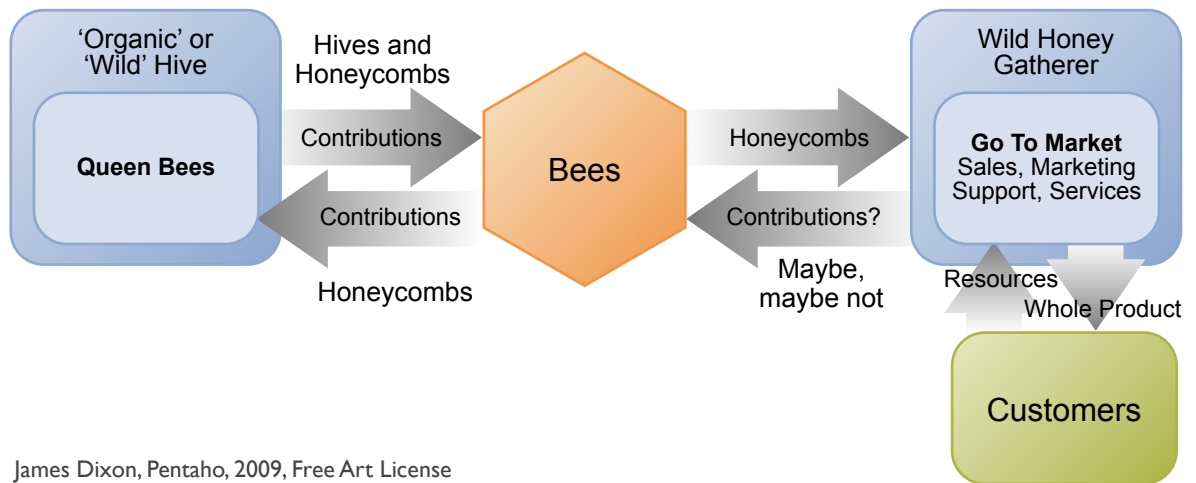
The single-vendor commercial open source is a perfect (in mathematical terms) combination of an open source project and a proprietary software company. It has all of the elements of the other models: nothing added and nothing removed.

## The Honey-Gatherer Model for Service/Support Commercial Open Source

The first version of this paper did not include a model to describe companies that use a services or support commercial open source business model. Matthew Aslett of the 451 Group wrote a blog about extending the Beekeeper Model to include these companies.

<http://blogs.the451group.com/opensource/2008/06/20/applying-the-bee-keeper-model-beyond-captive-open-source-projects/>

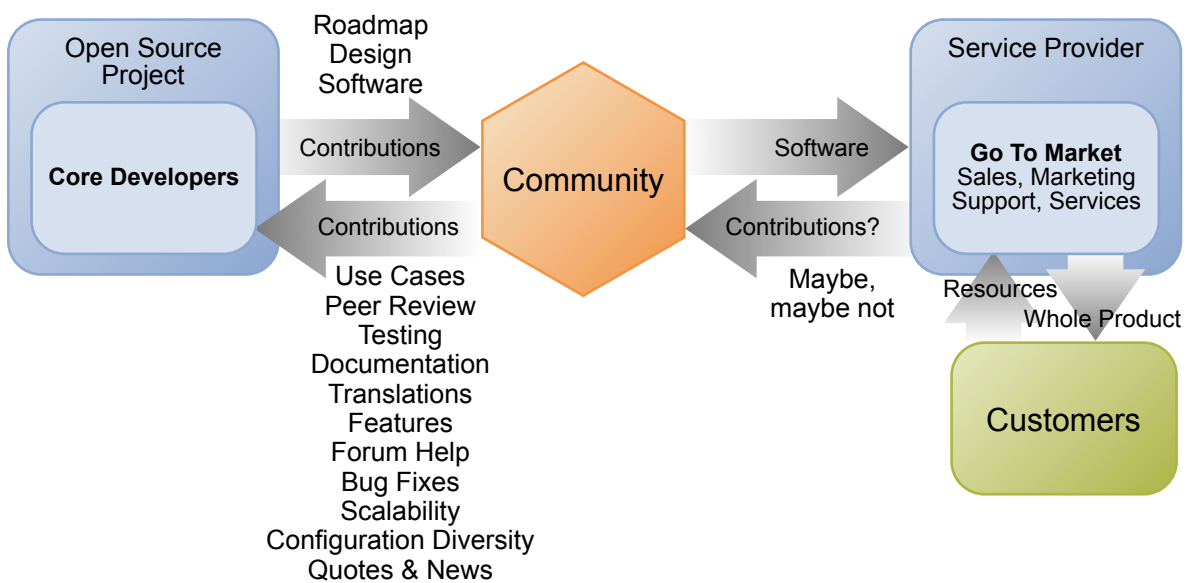
He suggested that these companies can be likened to honey gatherers that collect their raw material from wild or organic hives.



James Dixon, Pentaho, 2009, Free Art License

These honey gatherers produce the same products as the Bee Farm but their involvement with, and contribution to, the bee community is different. Notice that there is no bee farm in this model.

In diagram form the workings of a services / support commercial open source model is similar.



James Dixon, Pentaho, 2009, Free Art License

When compared with the Beekeeper Model it is obvious that the resources provided to the bees by the gatherer are different. In the Beekeeper Model the bee farm provides land, hives, and flowers etc. In the Honey Gatherer Model the gatherer goes into the 'wild' and finds existing hives from which to obtain raw materials. Likewise the single-vendor commercial open source business provides resources to the community that a services/support business does not. These resources include:

- Initial or 'seed' code. This can take millions of dollars to create.
- A sizable team of full-time engineers
- Product managers
- Graphic designers and usability experts
- Development tools such as continuous integration servers

It is clear that the single-vendor model is more costly to set up and operate than the services/support model. It is logical that companies using the Beekeeper Model need to generate more revenue to recoup these costs than a company using the Honey Gatherer Model. This explains the common practice of the Beekeeper companies to offer some kind of 'Enterprise Edition' that includes features not available to the community. These are high-end features that only larger organizations find valuable.

Since the 'gatherer' does not write little (or none) of the software they are not in a good position to effectively control intellectual property and patent-related issues.

In support of this observation, in his OpenGEO blog (<http://theopenplanningproject.org/2009/03/the-beekeeper/>) Paul Ramsey of The Open Planning Project points out that many potential customers do not perceive enough value in the whole product features alone to convince them to become customers.

Alfresco provides another data point. Prior to March 2007 Alfresco had enterprise features that were only available to customers. They then switched to a services/support model. As of March 2009 it looks, from John Newton's blog (<http://newton.typepad.com/content/2009/03/building-a-stronger-open-source-product.html>) like Alfresco is switching back to the open-core model used by many single-vendor commercial open source companies. Newton ends his post with this message: "We are making these changes in a way that is based on a set of principles that are fair and accountable. We believe in open source and making it freely available and providing choice of not just proprietary systems, but between enterprise and open source. We think the rest of the open source world is heading in a similar direction, because this is what makes open source stronger in the long run."

Some people assume that all commercial open source models are flawed because the company does not have direct control over the direction of, and development of, the software:

- This is not true with the single-vendor model. The only resources that cannot **directly** be influenced are the community who perform software-oriented roles at a scale and capacity that is orders of magnitude higher than any proprietary organization could do on its own. The core developers and product managers are under the direct influence of the company.
- The services/support model does suffer from this. The company might pay for full-time developers to work on some of the open source projects that it utilizes but it does not have the same level of influence that the single-vendor model provides.

## Summary

The deliberate simplification of the models and diagrams above shows something important. If you reduce the workings of an open source project, a proprietary software company, and a single-vendor commercial open source software company down to the fewest number of lines and bubbles there are fundamental differences between them.

By comparing the diagrams you see the single-vendor commercial open source model is a perfect combination of an open source project and a proprietary software company: there is nothing left out and there is nothing added.

**single-vendor commercial open source = open source software project + 'whole product' program**

The single-Vendor commercial open source model is a robust combination of an open source project and a 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 single-vendor commercial open source company benefits by increasing its valuation when it meets the needs of both customers and open source community.
- Out of the three open source models the single-vendor model is most effective in the areas of intellectual property and patents.

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 table:

	<b>Open Source</b>	<b>Proprietary</b>	<b>Commercial Open Source</b>
Transparency of design & implementation	<b>Higher</b>	<b>Lower</b>	<b>Higher</b>
Quality of software	<b>Higher</b>	<b>Lower</b>	<b>Higher</b>
Reliability of support and training	<b>Lower</b>	<b>Higher</b>	<b>Higher</b>
Reliability of roadmap	<b>Lower</b>	<b>Higher</b>	<b>Higher</b>
Ownership of solution	<b>Higher</b>	<b>Lower</b>	<b>Higher</b>
Availability of professional services	<b>Lower</b>	<b>Higher</b>	<b>Higher</b>
Availability of references/case studies	<b>Lower</b>	<b>Higher</b>	<b>Higher</b>
Ability to 'try before you buy'	<b>Higher</b>	<b>Lower</b>	<b>Higher</b>
Cost of license or subscription	<b>None</b>	<b>High</b>	<b>Lower</b>
Ability to customize software	<b>Higher</b>	<b>Lower</b>	<b>Higher</b>

As demonstrated by the Beekeeper diagrams and this table the Single-Vendor Commercial Open Source model, when implemented well, is an 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 / commercial open source is due to the profound and fundamental differences that exist between the proprietary software development model and the open source / commercial open source models.

Commercial open source companies are a lot of fun to work at. I don't know of anyone working in a commercial open source 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 single-vendor commercial open source 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 part two of this model at: <http://www.pentaho.org/beekeeper>

email: [beekeeper@pentaho.org](mailto:beekeeper@pentaho.org)

### External References to the Beekeeper Model

The Beekeeper

Paul Ramsey, The Open Planning Project

<http://theopenplanningproject.org/2009/03/the-beekeeper/>

Open Source Business Models: the Beekeeper model

Roberto Galoppini

<http://robertogaloppini.net/2007/05/23/open-source-business-models-the-beekeeper-model/>

Applying the Bee Keeper model beyond captive open source projects

Matthew Aslett, 451 Group

<http://blogs.the451group.com/opensource/2008/06/20/applying-the-bee-keeper-model-beyond-captive-open-source-projects/>

Overheard: The beekeeper model and open source software

Margaret Rouse, IT Knowledge Exchange

<http://itknowledgeexchange.techtarget.com/overheard/overheard-the-beekeeper-model-and-open-source-software/>

The Bee Keeper analogy as an explanation of professional FOSS

Martin Michlmayr, foss bazaar

<https://fossbazaar.org/content/bee-keeper-analogy-explanation-professional-foss>

Bee Keepers, the Chasm & Open Source

Michael Grove, CollabWorks

[http://blog.collabworks.com/openit\\_blogs/2007/09/bee-keepers-the.html](http://blog.collabworks.com/openit_blogs/2007/09/bee-keepers-the.html)

Making Money With Open Source, Part 1: Turning Users Into Buyers

Dominic Sartorio and Bernard Golden, Linux Insider

<http://www.linuxinsider.com/story/61083.html>

Commercial Open Source Support

Recombinant – Healthcare Data Warehouse

[http://www.recomdata.com/www/services\\_commercial.html](http://www.recomdata.com/www/services_commercial.html)

Take Care of Your Bees

Michael Grove, Open IT Works

[http://opensolutionsalliance.org/osa/osaalert\(nov07\)-grove.html?command=Default&siteId=2&tabId=32&pageId=100](http://opensolutionsalliance.org/osa/osaalert(nov07)-grove.html?command=Default&siteId=2&tabId=32&pageId=100)

Penguins, Bees, Cathedrals and Wikis

Dick Selwood, Embedded Technology Journal

[http://www.embeddedtechjournal.com/articles\\_2009/20090224\\_opensource.htm](http://www.embeddedtechjournal.com/articles_2009/20090224_opensource.htm)

Open Source is Not a Business Model

451 Group

[http://www.the451group.com/caos/caos\\_detail.php?icid=694](http://www.the451group.com/caos/caos_detail.php?icid=694)

Martin, Open Source or Not?

<http://opensourceornot.blogspot.com/2007/05/dancing-with-professional-open-source.html>

FOSS GIS in Local Government

Nico Elma, GIS Global Image Ltd

<http://www.globalimage.co.za/Documents/GIS%20Global%20Image%20presentation%20FOSS%20GIS%20in%20Local%20Government%20FOSS4G%202008.pdf>

RESILIENCE OF PROFESSIONAL OPEN SOURCE ECOSYSTEMS

Wynn (University of Dayton), et al

<http://is2.lse.ac.uk/asp/aspecis/20080123.pdf>

Government Open Source Software Resource Center  
<http://www.gossrc.org/articles>

Open Source Week in Review  
Mark Hinkle, Socialized Software  
<http://socializedsoftware.com/2008/06/22/links-open-source-week-in-review/>

Commercial Open Source  
Martin P, Inflection Technologies  
<http://inflection-technologies.com/PACSFerret/?p=12>

Open Source Software Engineering  
G. Sivakumar, ITT Bombay  
[http://oss-conference.in/userfiles/file/GShivakumar\\_ITTBombay\\_CII\\_FOSS.pdf](http://oss-conference.in/userfiles/file/GShivakumar_ITTBombay_CII_FOSS.pdf)

Open Source  
The IT Bureau  
[http://www.theitbureau.co.uk/index.php?option=com\\_content&task=view&id=268&Itemid=68](http://www.theitbureau.co.uk/index.php?option=com_content&task=view&id=268&Itemid=68)

The apiarist's dilemma  
Matthew Aslett, 451 Group  
<http://blogs.the451group.com/opensource/2008/07/28/the-apiarists-dilemma/>

The Economics of Open Source  
Chris Katz et al, Department of Computer Science, Tufts University, Massachusetts, USA  
<http://www.cs.tufts.edu/comp/180/teams/t3/documents/economics.ppt>

Open Source Methodology: Software Engineering Principles  
Dr Pretra Malik, School of Mathematics, Statistics and Computer Science, Victoria University of Wellington, New Zealand  
[http://www.mcs.vuw.ac.nz/courses/COMP301/2008T1/lectures/COMP301\\_FOSSM\\_2008.pdf](http://www.mcs.vuw.ac.nz/courses/COMP301/2008T1/lectures/COMP301_FOSSM_2008.pdf)

A Review of New Framework for Low Cost ERP System using Open Source Software  
Baharum ( Universiti Teknologi Malaysia. ) et al  
[http://repository.gunadarma.ac.id:8000/iiWAS2007\\_527-533\\_1159.pdf](http://repository.gunadarma.ac.id:8000/iiWAS2007_527-533_1159.pdf)

Open Source Business Strategy: Feedback on the Beekeeper Model Revisited  
Roberto Galoppini  
<http://robertogaloppini.net/2009/04/11/open-source-business-strategy-feedback-on-the-beekeeper-model-revisited>

The Sharecropper Model for Commercial Open Source  
Tarus Balog, OpenNMS blog  
<http://blogs.opennms.org/?p=764>

Open Source Race to Zero May Destroy Software Industry  
Gene Quinn, IPWatchdog  
<http://www.ipwatchdog.com/2009/04/02/open-source-race-to-zero-may-destroy-software-industry/>