component ecosystems – the context for cbd

richard veryard

Filename: **compeco.pdf**
Version: September 17th, 1999

introduction

motivation - conflicting notions and perspectives about components

Component-Based Development is commonly described in terms of a set of notions (interface, service, encapsulation, reuse, plug-n-play). But these notions do not have a single interpretation from all perspectives.

Take **reuse**, for example. Some people think reuse is terribly important, and other people don't care a fig for reuse, but do care about critical mass or quality. Champions of reuse try to engage wider support for reuse initiatives by arguing that reuse effectively means higher software quality and consistency, lower software costs, faster delivery and/or greater connectivity. But when you link reuse with these other notions, you alter the notion of reuse itself. This fact only becomes evident when you try to agree how to measure and manage reuse. A software engineer whose main motivation for reuse is to increase the productivity of software development doesn't want to measure and manage reuse in the same way as a software engineer whose primary concern is software quality or maintainability.

Even the notion of **component** itself means something different, according to whether you are talking to Java programmers or respository managers or potential purchasers. How many components are there?

When faced with differences in terminology or thinking, many engineers immediately assume that the only solution is to agree a standard terminology. In other words, the software industry must have only one notion of component or reuse. Although this seems reasonable in theory, practical experience indicates that the process of consensus-building and standardization is usually fraught with conflict, delay, compromise and confusion.

In this document, we take a different approach. We recognize that there are many stakeholders with different perspectives on components. We observe that there are several competing notions of component, reuse and other key terms, and we assume that all of these notions are valid in some context. Our goal is to understand these notions, and find ways of building useful bridges between them, not to decide which of them is the "best" or "most valid".

We analyse these differences by defining multiple **ecosystems**, each following a different logic. In this document, we define four ecosystems: Service Use, Service Supply, Device Use and Device Supply.

Now we can start to be more precise about different perspectives on, say, reuse. In each ecosystem, there are different reasons why reuse might be directly or indirectly important to stakeholders in that ecosystem. See Figure 1.

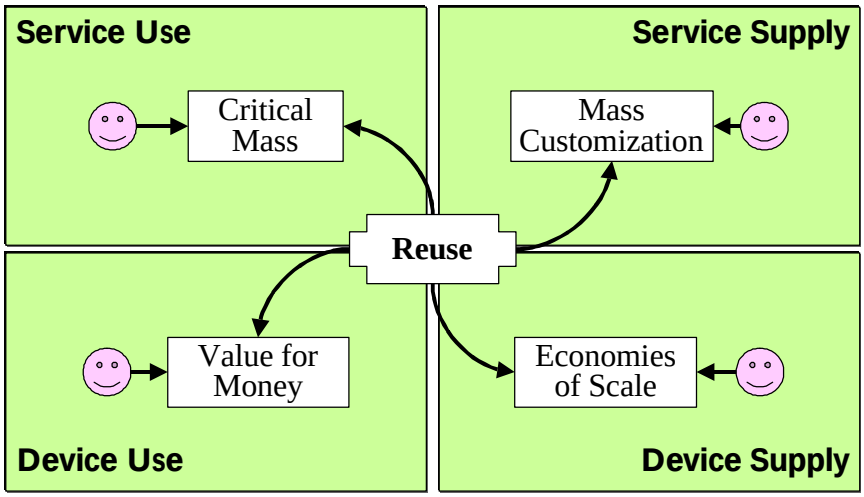

Figure 1: Four perspectives on reuse.

What is the connection (if there is one) between reuse in the Device-Supply ecosystem and critical mass in the Service-Use ecosystem? There are lots of ungrounded claims that more reuse leads to greater quality, but where's the proof, and what would actually count as a valid proof?

In short, how can a software engineer persuade a business manager to invest in "reuse", when they don't share a notion of what reuse is, and what value it might have?

joined-up thinking about components and related questions

Should the software engineer adopt a business management notion of reuse, or should the business manager understand the technical notion of reuse? Neither of these - instead, we need to find ways of connecting these two notions of reuse together.

When faced with conflicting terminology, most people try to smooth out the conflicts and agree a single homogeneous set of terms. This approach has at least four potential dangers.

1. The agreed terminology becomes so bland, and so abstract, that it becomes practically meaningless.

2. The agreed terminology becomes so complicated, that it becomes practically unusable.

3. The agreed terminology leaves out some perspectives or stakeholders.

4. The process of agreement is too slow, and is overtaken by events. (For example, unilateral action by a major vendor, or the arrival of the next technological wave.)

Rather than smooth out the differences, our approach is to understand them explicitly, and to build bridges and connections between them. This is not just an intellectual exercise but an important commercial one.

We believe that the dominant players in the component marketplace will be those that can understand and engage with multiple perspectives, and can straddle multiple ecosystems.

hybrids and multiple contexts

There was a guy who achieved some notoriety in the patent profession - my father was a patent agent - by taking out patents in strange hybrids. For example, he got a patent in a device that was a combined nuclear fall-out detector and catflap. (Given that patent law is designed to prevent silly patents being granted, this required an excellent knowledge of patent law, as well as extraordinary skill at drafting.)

Component-based development (CBD) is a similar hybrid, in the sense that it yokes together disparate concepts and mixes metaphors. Furthermore, many of the so-called gurus seem unaware of this. Endless arguments about what exactly a component is, or how you measure reuse, cannot be resolved without recognizing that there are multiple contexts.

To help make these contexts explicit, we have developed the model of four ecosystems described in this document. This should provide a decent basis for saying what CBD actually is – or even defining some other, more coherent notions. It also helps us to build conceptual bridges between the different perspectives, and find practical ways of collaborating across multiple ecosystems.

four ecosystem model

separation

To analyse the context for software component development, we separate the whole into separate ecosystems.

The first separation we draw is between the demand-use side and the supply side. This separation will be familiar to most readers.

Component-based development enables a further separation, between the external service (accessed through a component interface), and the internal component assets or devices that deliver the service. This second separation applies equally on the demand-use side and on the supply side, yielding four ecosystems altogether, as in Figure 2.

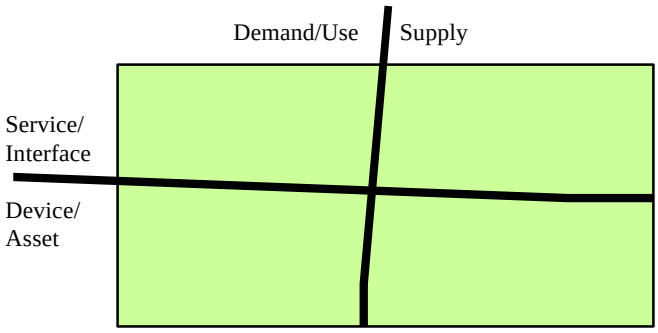

Figure 2: Four ecosystems.

service use ecosystem

activities

- Using services
- Demanding services
- Architecting / configuring use of services
- Subscribing to service publications

ecological principles

An important element of strategic thinking around a business process is to decide: which bits are to be routine and mechanical, consuming as little management time and attention as possible; and which bits are to be strategically interesting, on which management time and attention is to be focused. Thus some business services need to be as boring as possible, while others need to be as exciting as possible. And it's important to get the balance right - too much excitement is painful or stressful, while too little excitement is death. This balance is a critical survival factor for the ecosystem as a whole; we call this the **pleasure principle**.

Furthermore, some services give value to their users by being unique, while others give value to their users by being common. The best-known example of the latter is communication services: how much value you get from your email or fax service depends on how many of your friends and associates are also using email or fax. Thus your decision to use a given service sometimes depends on your estimate of the number of other users. We call this the **connectivity principle**. (It is sometimes known as the **critical mass** principle.)

service supply ecosystem

activities

- Providing / delivering services through stable interfaces
- Architecting services
- Publishing available services

ecological principles

In this ecosystem, services are competing for survival. Between two services, the more available service will usually win over the less available. Hence there is a strong technological and commercial pressure for services to increase their availability; we call this the **availability principle**. (It is sometimes known as the **commodity principle**.)

Some aspects of availability are as follows (depending on the nature of the service):

- Global 24-hour access. Instant response.
- Any hardware and software platform. Available in Arabic, Chinese, English, Hindi, Russian and Spanish.
- Easy to use. Low entry cost. Good support. Minimum learning curve.
- High reliability. Safe and secure. Low risk.

device supply ecosystem

activities

- Architecting devices
- Providing devices to deliver services (build, buy, assemble, reuse)
- Managing devices as assets

ecological principles

In this ecosystem, competitive survival depends on delivering the greatest quantity of service with the smallest amount of work. This is often called **reuse**; software reuse should be focused on achieving **economies of scale** in software, based on effective asset management and knowledge management. We call this the **energy conservation principle**.

device use ecosystem

activities

- Using services
- Demanding services
- Architecting / configuring use of services
- Subscribing to service publications

ecological principles

In this ecosystem, competitive survival depends on getting the expected services (and their associated benefits) from a given configuration of devices. This in turn relies on an ability to predict and control the behaviour of components-in-use, including the emergent properties of large distributed systems. We call this the **quality principle**.

Also relevant in this ecosystem is the ability to easily substitute devices and reconfigure systems. We call this the **flexibility principle.**

Finally, the robustness, flexibility and evolution of the ecosystem depends on a reasonable heterogeneity of software and services. We call this the **biodiversity principle.**

ecological principles

connectivity

network externalities

Economists use the term **network externality** to refer to those costs and benefits that depend on the number of other users.

In many cases, the network externalities are negative. The utility of a car is reduced if there are too many other road users; the utility of a holiday may be reduced if there are too many other holiday-makers.

The most common examples of positive network externalities come from communications technology. A phone or fax has no value to you, if you are the only person that has one. The more people that share this technology, the more valuable it becomes.

Similar externalities apply in many other situations. My choice of word processor is influenced by the fact that I want to exchange word-processed documents with my friends and associates. An organization selecting a software development tool is influenced by the number of other organizations using the tool – among other things, they want to know that there will be lots of people in the job market (available as employees) familiar with the tool.

Success breeds success. To him that has, shall be given more.

standards

Markets, especially for intangible things like software components, need standards. Standards or standard notations for component description as well as standards for component execution (CORBA or COM).

Standards don't have to be universal. Often there are parallel or rival standards. CORBA and COM. PC and Macintosh and UNIX.

Within each market sector, if there are rival standards and competing notations, this adds complexity and reduces network externalities. But it doesn't follow that there should only be one standard, even within a single sector of the market, and certainly not across all market sectors. I certainly don't expect components for nuclear power stations to be documented in the same way as components for banking systems. (See also comments on biodiversity below.)

The practical value of a standard depends on one thing alone: the density of the population adhering to the standard. Technical purists may prefer Betamax to VHS, but it was VHS that achieved the critical mass. Microsoft understands this very clearly.

critical mass

Critical mass denotes a point where a given density of interaction is reached, causing an explosion to occur. This is a very good metaphor for what happens with communication technologies such as phone, fax and email. The term is also used loosely to denote the size of a market or other ecosystem.

size and survival

In understanding this ecological principle of critical mass, it's important to distinguish the desirable (perfect) from the essential (good enough).

You may prefer to have a single universal standard. You may prefer to have a single universal platform. You may prefer to have a single search engine that will find every component in the universe, and present its description in a single universal notation.

But when you are operating in a given standard, on a given platform, using a given search engine or notation, what matters is how large and diverse a population this does give you access to. Perhaps there are lots of other components operating in a different standard, on a different platform. But if your search engine already gives you access to more components than you need, you must either decide to forget about the components it isn't giving you access to, or use a second search engine.

Ideally, I might want to be able to converse with everybody in the world. But I'd need to learn thousands of languages, which is impossible, even for the most gifted linguist. Most of us can only manage a few languages, and many people get by with only one. The major world languages have a large population of speakers, providing a critical mass of people, reading material, general opportunities. Minor languages, whatever their cultural importance or poetical wealth, lack this critical mass. This is why people who speak Basque or Finnish or Welsh are under greater pressure to learn other languages than those who speak Castillian or Russian or English.

In a real ecosystem (as opposed to Aesop's fables) the animals concentrate on eating the food that is available to them, and don't waste energy regretting the food that might be found in some other ecosystem. If there isn't enough food, they either migrate or die.

conservation of energy

economies of scale

People talk a great deal about **software reuse**. The real benefits of software reuse lie in the **economies of scale** of software production and supply.

lifetime costs

Unlike simplistic notions of software reuse, the scope of conservation of energy is not just the one-time development of a software solution, but the lifetime costs of managing and evolving

the solution. If a software solution goes through many different versions in its lifetime, each version should reuse much of the previous version - the more the better.

recycling intellectual property

The principle of **conservation of energy** is much broader than software reuse. We also want to reuse and recycle intellectual property, however it is bundled, as software or otherwise. This includes capitalizing on existing assets, including of course "legacy" systems.

Recycling, of course, implies movement and use. Conservation certainly doesn't mean hoarding. Competitive advantage doesn't come from possession and preservation of static intellectual property, but from the rapid development and exploitation of new intellectual property.

supply consolidation

Organizational economies of scale are also available through the software supply chain, from developers to retailers. Given the transaction costs in the supply chain, this is more likely to involve horizontal integration (for example, retailers offering a broader range of products, developers addressing a broader range of requirements) than vertical integration (one firm covering the whole supply chain).

flexibility

substitution

Can I change the overall functionality or performance of my system by replacing one component with a similar component, without changing any other component?

(Note: this probably means that the new component has the same interface as the old one, but a different specification. Beware of gurus who tell you that interfaces and specifications are the same thing.)

And if possible, I'd like to do this without having to re-test the whole system.

reconfiguration

Can I change the overall functionality or performance of my system by rewiring the same components? Can the components be plugged together in many different ways, without interfering with their functionality or performance?

biodiversity

vulnerability of monoculture

"While the dominance of a single computing environment -- the one powered by Microsoft software and Intel chips -- offers the benefits of compatibility among machines, some say it may share the vulnerabilities of fields planted with just one crop." [Markoff]

Recent worms and viruses have swept across the Internet, prompting comparison with the ease with which disease organisms sweep through human populations, and their herds and crops.

The best insurance against this vulnerability is the software equivalent of biodiversity: software diversity.

Art Amolsch, editor of FTC Watch, a Washington policy newsletter, is quoted as proposing that no government agency be allowed to run more than 34 percent of its personal computers on one proprietary operating system.

innovation from the fringes

Not only is a homogeneous culture vulnerable to external attack from predators and parasites, it can also be slower to adapt and evolve. In his book Guns, Germs and Steel, Jared Diamond uses this fact to explain why China, once way ahead of Europe in agriculture and technology, slipped behind over the past 500 years.

availability (commodity)

Availability is defined as making things more widely accessible, eliminating the barriers to use. Wherever you want it, whenever you want it. Easy, safe and cheap.

Components should be as available as possible, in order to offer the greatest possible value (utility) to the greatest number of potential users/uses.

retail consolidation

One way of making things more available is bringing them under one roof. A department store or superstore is more convenient for shoppers. An e-commerce website takes this principle of convenience further, and should make as many choices available as possible.

diversity

This is related to the concept of **requisite variety**. A component that runs on many different platforms, a component that offers a choice of operating protocols, a component that can handle multiple data formats - these are components whose variety makes them available to a wide range of users/uses.

granularity

A healthy market needs to offer a good mix of different granularities. This increases overall availability.

Although perhaps most of the early transactions in the software component market have been in fine-grained components, there will be a trend towards a greater choice of granularity.

For some, the transaction costs of buying lots of small components may be greater than the transaction cost of buying a few large components. For others, the reverse will be true. Thus the transaction cost argument works both ways.

ease of acquisition

How easy is it to find, evaluate and acquire a component? Anything that makes it easier and safer - documentation, demonstration versions, case studies, user or analyst recommendations, quality accreditation, clear price-list - increases the availability of the component.

quality (firmness)

assurance

Who can tell me that the component will do what I want it to, in my own environment? What guarantees do I have? If other people have tested or inspected the component, how relevant is this to me?

performance

How can I get satisfactory performance from an assembly of components? If I buy your component, what performance will I get from the system as a whole?

reliability

Does the system work robustly? Is it safeguarded against rogue or poor quality components?

When components fail, do they fail cleanly, or do they cause secondary problems?

feature interaction

Does each component continue to provide the specified service, regardless of unforeseen interactions with other components?

pleasure (delight)

satisfaction (I can't get no)

The pleasure principle is actually a homeostatic principle: keeping tension and stress to a minimum.

attention

The only factor becoming scarce in a world of abundance is attention. [Kelly, p 59]

Some suppliers may interpret this as a need for promotion and publicity: brand image management, advertising, public relations.

We've probably all seen the **INTEL INSIDE** stickers on the outside of computers. Some analysts even question the commercial value of this campaign to Intel. But just imagine if all the other suppliers of all the other components – hardware and software – wanted to put a sticker on the outside of your computer. You wouldn't be able to see the screen!

The whole point of a component is that it should be invisible most of the time. You probably don't have a sticker on your car, telling you what is the brand of spark plugs in the engine. Indeed, the spark plug manufacturer probably doesn't care whether you've even heard of him. The only attention he wants is that of the engine designer, and possibly the engine repairer.

excitement

"The function of the pleasure principle is to make man always search for what he has to find again, but which he will never attain." [Lacan, Seminar VII]

implications

CBD notions

component

A software component involves a relationship between a service interface (in the Supply ecosystem) and a software device (in the Device ecosystem). The device implements the interface, the interface specifies the device.

This is a many-to-many relationship. One interface may by implemented several different ways, by different devices. One device may satisfy many different specifications, describing different interfaces.

In practice, software components often fall short of this ideal definition. It may be more accurate to say that the device claims to implement the interface, while the interface tries to specify the device.

encapsulation

We can characterize encapsulation as a statement about the relationship between the Service Supply ecosystem and the Device Supply ecosystem. Certain aspects of a component are not accessible within the Service Supply ecosystem, and are hidden behind the service interface.

Encapsulation can also be expressed as an approximately equivalent statement about the relationship between the Service Use ecosystem and the Device Use ecosystem.

plug'n'play

The very metaphor of plugging and playing means that we are in the Device Use ecosystem. This notion has no meaning in the other three ecosystems.

reuse

Reuse of software assets primarily makes sense within the Device Supply ecosystem, although it also has relevance within the Device Use ecosystem.

Within Device Supply, reuse equates to economies of scale in software development and maintenance. Within Device Use, reuse equates to economies of scale in software procurement and operation, which is not the same thing. These impact the Service ecosystems only indirectly, to the extent that they affect service variety, cost and quality of service.

Within the Service ecosystems, a different notion of reuse can be focused on the commonality of services and interfaces. In order to exchange word processing documents with my friends and associates, I need a common exchange format. It ought not to matter to me what version of what word processing product they are using, as long as the formats match. I can certainly send faxes to people without knowing what fax machine they have.

design

design for connectivity

If we are trying to increase the connectivity of a component, and to establish a critical mass (density) of use, the design focuses on the following:

- Increase information content
- Look for ways of making component more active, more intelligent
- Expand connections with other components – networks enlarge small advantages

Kelly uses the example of a nail.

- Standard contractor size fits into standard air-powered hammers
- SKU designation fits into retail sales network
- Bar code fits into laser-read checkout system
- Embedded chip warns door of breakage – fits into smart house network

design for flexibility & biodiversity

If we are trying to increase flexibility and biodiversity, the design focuses on the following:

- Distribute intelligence
- Don't just support the execution of transactions, support the design of transactions as well
- Automate / animate change
- Zero latency
- The weaker the interface specification, the more things will fit.

Where is flexibility located?

- in individual component – reuse same component in new situation

- in component kit – substitute component for new situation
- in configuration – plug same components together in new ways
- in architecture – plug together new components for new situation

Where is diversity located?

- in the component kit – alternative components within same kit
- in the configuration – alternative paths and connections
- in the management process

design for availability

If we are trying to increase the availability of a component, the design focuses on the following:

- Whole product – not just software, but also support, documentation, training and other services.
- Consider giving the key components away free – make your money elsewhere.

There are many examples where these tactics have achieved significant market share:

- browser wars
- search engines
- shareware
- Linux Apache

design for quality / reliability

Systems should be robust and fault-tolerant.

Components should tolerate erratic system behaviour.

Examples include military systems and the Internet itself.

design to conserve energy (economies of scale)

If we are trying to exploit economies of scale, the design process focuses on the following:

- Aggressively exploit and anticipate the learning curve

- Align to the scale economies of your business / market
- Repackage and reuse knowledge assets at all levels – working practices, design patterns, templates & software code.

Kelly quotes the example of Fairchild Semiconductor.

- Initial production cost: $100
- Competing (old) product cost $1.05
- We sell ours for same price - at a huge loss.
- Gain 90% market share.
- Within 2 years, selling the same product for 50¢ - at a profit.

design for pleasure

If we are trying to increase the potential for pleasure, the design focuses on the following:

- reduce tension / stress (where this may be a result of excess choice or excess attention)
- support self-preservation
- engage users

Successful design involves a balance of contradictory forces:

- change / nochange
- evolutionary (small change) / revolutionary (large change)
- attention / inattention
- risk / reward

There are several examples of small changes that turned out to have revolutionary potential.

- email as substitute for office memos
- Amazon.com as substitute for traditional bookshop

commercial

types of software supply

CBD increases the separation between demand-driven and supply-driven software organizations. (This applies to in-house software factories as well as to commercial software houses.)

Demand driven	Supply driven
Close relationship to customer base. Focus on the services that are wanted in the selected market. Competing on value.	Close relationship to technology base. Focus on exploiting existing software assets. Competing on price/cost.

In the past, the supply-side could be divided into two modes of software supply.

Off-the-shelf. Some suppliers designed and marketed products for sale into an identified market (or ecosystem). These were standard products, with little or no variation, and were typically expensive to modify or integrate. If many users bought the same product, then the development and marketing costs could be shared between them, reducing the individual cost to each user.

Bespoke. Some suppliers designed and delivered one-off products to a particular customer's specification. These were typically much more expensive per user than standard off-the-shelf products, and took longer to deliver, but were (at least in theory) much closer to the customer's requirements.

Component-based development enables a third mode of software supply.

Mass customization. Suppliers who are able to respond to the needs of a single customer, while achieving economies of scale across multiple customers.

There are many ecosystems containing only off-the-shelf and bespoke suppliers, in which neither mode of supply can eliminate the other. However, as soon as mass customization becomes effective in a given supply ecosystem, then off-the-shelf and bespoke supply are ecologically doomed, and will eventually be eliminated from that ecosystem. These suppliers may survive in the short term by switching to other (perhaps smaller niche) ecosystems, but for how long?

According to this analysis, the growth of CBD creates challenges for both types of software supplier.

challenges for software houses

A software house that specializes in bespoke software development may detect increasing difficulties competing on price. If your competitors are achieving better economies of scale, without compromising quality and flexibility, then they will be able to undercut your prices.

Most software houses still bid for bespoke work on the basis of a simple formula: estimated cost plus contingency plus profit. There may be some opportunities for you to reduce costs or contingency, by improving your software process.

But if your competitors are doing this too, this won't be enough.

There are two possible strategies for survival. You can either move up the "food chain", concentrating on supply and packaging of services (while subcontracting the software engineering side to cheap suppliers in Bangalore). Or you can embrace the ecological imperative: conservation of energy.

Instead of bidding for bespoke work on a cost-plus basis, you must try to determine what the customer is willing to pay.

If this isn't enough to cover your costs, then you need to find a way of satisfying the customer that leaves you with some residual value. If you have developed some software components that you can sell to other customers as well, this might well make up the difference. (There are other forms of residual value, but this is the most likely one for a software house to exploit.)

challenges for package vendors

Meanwhile, a software product vendor with a standard fixed range of products may detect increasing difficulties maintaining market share, or entering new markets. If your competitors can offer more flexible products, with greater availability and lower total cost of ownership, then they can erode your customer base.

The challenge for such suppliers is to leverage the economies of scale, to get wider flexibility and availability from an equivalent device base, and to get much greater internal levels of reuse. This is basically an architectural issue: how to improve the internal configuration and layering of the product. (Some suppliers will choose to keep the benefits of this improved architecture to themselves, while others will choose to open up the architecture to customers and third parties.)

testing

Given this model of the CBD world, two distinct forms of testing are needed. (Similar remarks apply to verification and validation).

intra-ecosystem testing

Testing components and component interactions within one ecosystem.

- For example, within the service supply ecosystem, we may test that services satisfy their specifications. We can also test interactions between a bundle of services.

- For example, within the device supply ecosystem, we may test conformance of components to various specifications or standards.

Most of the available tools and techniques for testing belong to a single ecosystem.

inter-ecosystem testing

Testing components and component interactions across two or more ecosystems.

- For example, testing that a device satisfactorily implements an interface.

- For example, end-user acceptance testing.

Testing across two or more ecosystems needs a collaboration between multiple roles, where each role represents a given perspective within a given ecosystem.

methodological

My characterization of the four ecosystems is preliminary. I'm not even sure that there have to be four of them. I am however fairly convinced of the need for some such model of a number of separate ecosystems, able to support at least the distinctions I want to make and probably some more as well.

We also need to recognize that there is some interfolding of the four ecosystems, in the sense that each may impinge into the environment of the others. However, I don't think this observation forces me to accept the validity of a single whole system.

The idea of dividing a situation into multiple ecosystems has more general applicability. To take another example, a manufacturing company would typically have models of the production process and also models of the sales and marketing process. (The production process could have an enterprise model, an information model and so on. Similarly, the sales and marketing process would have multiple models.)

We could usefully regard the production process and the sales and marketing process as belonging to two separate ecosystems. There are known problems in joining / yoking the information model of one process with the information model of another process, which popular fantasies of the global enterprise-wide information model fail to address. There are similar problems in joining the enterprise model belonging to one process/ecosystem with the enterprise model belonging to the other. Of course, production impinges on sales & marketing, and vice versa, but the coordination between the two processes is not simply a matter of merging the models.

People from the production ecosystem just don't speak the same language, or recognize the same problems, as the people from the sales & marketing ecosystem. Of course they come to a pragmatic accommodation with each other, but this doesn't represent a true meeting of minds,

or shared intentionality. And that's true even when they are all divisions of the same company. And it gets all the more interesting in a distributed or federated situation.

So I think there is a lot of mileage in the general idea of producing models of multiple ecosystems, and problematizing the interactions across ecosystems. I will defend that idea much more strongly that I will defend the particular model I have produced of the CBD world.

However, the four ecosystem model of the CBD world allows me, I believe, to throw some new light on a number of CBD topics.

We use the word **ecosystem**, rather than **markets** or **industries** or **networks**, because it is more general. An ecosystem may contain human agents or organizations, or it may contain intelligent software agents, or it may be a hybrid.

The natural world can be regarded as a single global ecosystem, because there are some connections between all the parts. (Birds may visit even a remote island, carrying new species of plant or insect.) However, for many purposes it is simpler to regard a semi-isolated part as a separate ecosystem.

Some economists use the term **ecologies** rather than **ecosystems**, but I prefer to reserve the term ecology to refer to the scientific study of ecosystems. (I'm no biologist, and I haven't studied how professional ecologists handle these situations. My focus is on ways of modelling that will support business strategy and change management, and also IT planning and technology transfer.)

In delineating an ecosystem, I think one is saying that the interactions within the ecosystem are somehow different to the interactions across the boundary of the ecosystem.

Here's a simple analogy. People work in universities according to a certain logic. They interact with each other in particular ways, compete for particular tokens of success, and so on. People work in industry according to a different logic. I think it's fair to characterize this as two separate (but connected) ecosystems. In the academic ecosystem, there is an ecological principle of "publish or die"; this principle is largely absent from industry. The two ecosystems have different pressures, different time horizons, different ways of thinking about all sorts of problems. This is precisely why it's worth having many bridges and joint activities between the two ecosystems; but these are likely to fail if they don't recognize that there are different principles on each side.

If I'm operating in two ecosystems at the same time, what are the implications of this? Do I become schizophrenic? In many companies, the marketing department operates in a different ecosystem to the production department, and the resulting inter-departmental tensions and conflicts can be very damaging unless carefully managed.

Instead of the biological/ecological metaphor of **ecosystem**, some readers may prefer to think in terms of a **political economy**. However, I think the ecosystem metaphor is more useful than the market metaphor in dealing with these multiple roles and the conflicts between them. A bird that wants to do well in the bird-worm ecosystem has to spend a lot of time pecking at the ground, but a bird that wants to do well in the bird-cat ecosystem has to spend a lot of time flying away. The features or defences that are useful in one ecosystem may be a handicap in another ecosystem. A simple characterization of a market as a set of buyers confronting a set of sellers doesn't go far enough for my purposes.

references

Albert Borgmann. **Technology and the Character of Contemporary Life.** A philosophical inquiry.

(Chicago University Press. 1984.)

A classic account of technological change. Borgmann introduces what he calls the ***device paradigm****, in which technological progress increases the availability of a commodity or service, and at the same time pushes the actual device or mechanism into the background.*

Jared Diamond. **Guns Germs and Steel.** A short history of everybody for the last 13,000 years.

(Vintage Random House, 1998.)

Provides a large look at the history of the world. Explains why the people in some parts of the world developed faster and further than others, without appealing to inherent genetic (racial) differences. The relevance of this book here is its contribution to the topic of biodiversity.

Kevin Kelly. **New Rules for the New Economy.** 10 ways the network economy is changing everything.

(US edition: Viking Penguin; UK edition: Fourth Estate. 1998.)

A systematic analysis of the strategies for successful business in the new world. This is the open, distributed, connected, chaotic world he described in his previous book. Out of Control: The New Biology of Machines, Social Systems, and the Economic World.

John Markoff. "Illness Is Fast Becoming Apt Metaphor for Computers" (New York Times)

http://www.nytimes.com/library/tech/yr/mo/biztech/articles/14worm.html]

Richard Veryard. Related material can be found on the Veryard Projects website: http://www.veryard.com

acknowledgements & contact details

author details

Richard Veryard is a technology consultant, based in London. Please send feedback and further questions to richard@veryard.com.

acknowledgements

David Iggulden provided discussion and feedback at many levels. Vincent Traas asked some good questions. Thanks also to the CBDi Forum, in particular Paul Allen, Ian Graham, Simon Holloway, David Sprott, Lawrence Wilkes and Paul Winstone.

afterword

"A wit has said that one might divide mankind into officers, serving maids and chimney sweeps. To my mind this remark is not only witty but profound, and it would require a great speculative talent to devise a better classification. When a classification does not ideally exhaust its object, a haphazard classification is altogether preferable, because it sets imagination in motion." [Kierkegaard]