Biomimicry as a super systems metaphor for software engineering?

Samuel Mann

Lesley Smith

Otago Polytechnic
Dunedin, NZ

smann, lsmith @tekotago.ac.nz

Abstract

This paper examines the use of biomimicry in software engineering. By adopting the models of nature, we might hope to work more sustainably and produce more sustainable products. Could this be a way to the paradigm shift we have been looking for? To this end, perhaps nature and biomimicry could be super system metaphors for the development of sustainable software products.

In software development the system metaphor has been adopted as a core practice by the agile community. Kent Beck, author of Extreme Programming Explained (2000) defines a system metaphor as:

"a story that everyone - customers, programmers, and managers - can tell about how the system works."

The paper describes system metaphors and then examines work in this field.

1 Introduction

In this paper we explore options to give software engineering a heart of green.

Blevis (2007) describes a broad approach that brings together computing (HCI) and sustainability in a way that benefits both streams. He argues that sustainability "can and should be a central focus of interaction design." But, Blevis later (2008) writes "the moment you decide sustainability is an issue with respect to interaction design and the design of interactive devices is the moment you realize how complex the business of deciding what to actually do about it is. It is not just a simple matter of calculating the energy and environmental costs of manufacturing, use, salvage, and disposal of one technology over another".

This Supplementary Proceedings paper appeared at the *21st Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2008), Auckland, New Zealand. Samuel Mann and Mike Lopez (Eds). Reproduction for academic, not-for profit purposes permitted provided this text is included. www.naccq.ac.nz

So what is it a matter of?

Blevis suggests five principles to underpin Sustainable Interaction Design: (i) linking invention and disposal (ii) promoting renewal and reuse (iii) promoting quality and equality (iv) decoupling ownership and identity and (v) using natural models and reflection.

In this paper, we focus on the last of these principles and ask, what is the nature of this "use natural models and reflection". Although Blevis focuses on Interaction Design, we look more broadly at software engineering as a whole.

2 Sustainability in software engineering

Following Hanks *et al.* (2008), we focus on the link between sustainability and information technologies in two broad senses: (i) *sustainability through software engineering* —how systems can be used to promote more sustainable behaviours; and (ii) *sustainability in software engineering*—how sustainability can be used as a critical lens in the design of technologies themselves.

We focus on software engineering because, in addition to software driving demand for new hardware, software drives a wider set of values, beliefs and behaviours in society such that sustainability should be considered in software development process.

Software engineering has perhaps already adopted a sustainable approach without explicitly realising it - Agile proponents at least. The language of software engineering is already close to sustainability – discussions on stakeholders, interaction, and system requirements perhaps give evidence to an extant systems approach.

Blevis proposes a rubric, which, while intended for artefacts of interaction design, can be extended to wider software engineering:

The items of Blevis' rubric are:

1. Disposal - does the design cause the disposal of physical material, directly or indirectly and even if the primary material of the design is digital material?

2. Salvage - does the design enable the recovery of previously discarded physical material, directly or indirectly and even if the primary material of the design

3. Recycling - does the design make use of recycled physical materials or provide for the future recycling of physical materials, directly or indirectly and even if the primary material of the design is digital material?

4. Remanufacturing for reuse - does the design provide for the renewal of physical material for reuse or updated use, directly or indirectly and even if the primary material of the design is digital material?

5. Reuse as is - does the design provide for transfer of ownership, directly or indirectly and even if the primary material of the design is digital material?

6. Achieving longevity of use - does the design allow for long term use of physical materials by a single owner without transfer of ownership, directly or indirectly and even if the primary material of the design is digital material?

7. Sharing for maximal use - does the design allow for use of physical materials by many people as a construct of dynamic ownership, directly or indirectly and even if the primary material of the design is digital material?

8. Achieving heirloom status - does the design create artifice of long-lived appeal that motivates preservation such that transfer of ownership preserves quality of experience, directly or indirectly and even if the primary material of the design is digital material?

9. Finding wholesome alternatives to use - does the design eliminate the need for the use of physical resources, while still preserving or even ameliorating qualities of life in a manner that is sensitive to and scaffolds human motivations and desires?

10. Active repair of misuse - is the design specifically targeted at repairing the harmful effects of unsustainable use, substituting sustainable use in its place?

Recognising that this incorporation of sustainability is complex, Blevis is not claiming to have all the answers (or even all the questions). He does though pose some valuable research questions which he breaks into two sections. The first group concerns policy and prediction:

(a) How can the effects of information technologies on unsustainable behaviours be measured?

(b) How can the effects of harmful use of information technologies be predicted, or simulated?

…(f) Who is responsible for ensuring that design with the materials of technologies is directed towards sustainability?

His second category of questions concerns motivating sustainable behaviours by means of sustainable interaction design:

(a) How can digital artifice be designed such that people will prefer sustainable behaviours to unsustainable ones? …(c) How can renewal & reuse of digital artifice be made to be more attractive than invention & disposal in the view of interaction designers and in the public view?

3 Computing and natural science

A related driver for this work is the realisation that "computation is everywhere" (Denning, 2007), not least of which is the natural sciences. Denning continues "the old definition of computer science - the study of phenomena surrounding computers - is now obsolete. Computing is the study of natural and artificial information processes." In his article, Denning quotes Ken Wilson, who describes computation as the third leg of science (joining the traditions of theory and experiment) and Nobel Laureate and Caltech President David Baltimore: "Biology today is an information science".

Ecology in particular provides many examples of systems thinking, where the linkages between biological and environmental factors are explored in relation to human interventions. Computing can be used to model these interactions; conversely computing systems can be modelled on the ecosystem, capitalising on similarities in language and patterns. Odum (1996) states "because ecology is an integrative science, it has tremendous potential to provide a communication bridge between science and society".

4 Metaphor in software engineering

In software development the System Metaphor has been adopted as a core practice by the agile community. Kent Beck, author of Extreme Programming Explained defines a system metaphor as:

"a story that everyone - customers, programmers, and managers - can tell about how the system works." p. 179.

Wake and Wake (2002) argue that we seek a system metaphor for several reasons:

Common Vision: To enable everyone to agree on how the system works. The metaphor suggests the key structure of how the problem and the solution are perceived. This can make it easier to understand what the system is, as well as what it could be. **Shared Vocabulary:** The metaphor helps to suggest a common system of names for objects and the relationships between them. This can become a jargon in the best sense: a powerful, specialized, shorthand vocabulary for experts. Naming something helps give you power over it. **Generativity:** The analogies of a metaphor can suggest new ideas about the system (both problems and solutions). For example, the metaphor, "Customer service is an assembly line". This suggests the idea of a problem being handed from group to group to be worked on, but it also raises the question, "What happens when the problem gets to the end of the line - does it just fall off?" This can bring out important issues that might otherwise lurk and fester. **Architecture:** The metaphor shapes the system, by identifying key objects and suggesting aspects of their interfaces. It supports the static and dynamic object models of the system.

Some commonly used system metaphors are the spreadsheet, an assembly line, the shopping cart (used in e-commerce), a desktop or fridge door (information freely available, editable by all) and a library (information access and control systems).

The need for an underlying principle can be seen in Don Gotterbarn's work on ethics (2002). His premise is quite simple - if you're going to build something, think about the impacts. With Simon Rogerson he developed the concept of a Software Development Impact Statement (SoDIS) as a model which can be used to identify the ethical dimensions of a software development project and to identify ways to mediate the potential negative consequence of that project. The approach is developed from a distilling of 28 codes of ethics and is based around considering harm. When the approach was first developed it first considered harm caused by the computer, this was expanded to responsibility as a professional and now the questions are much broader.

What is the relationship between Don's ethics and sustainability? We attempted to draw a Venn diagram and failed: is ethics the superset? is sustainability? are they largely overlapping sets? or are they the same thing through different lens?). Don's suggested approach to integrating sustainability is to include "environment" as a stakeholder. The structured questions "Is there harm on the <environment>?" would then prompt consideration of sustainability. Unfortunately, this has limited success. We found that this was too much of a blank piece of paper - our students just left it blank. Don believes that this could be solved with more active prompting and questioning. He gives examples of how he challenges students with consequences they hadn't thought of. Unfortunately the rest of the world doesn't have Don in their classroom (or development team) facilitating discussion. We also worry about the ability of computing professionals to identify unexpected consequences resulting from incremental and perhaps insidious change in fields increasingly remote from their core expertise.

Somehow we need a balance between a checklist and rules (which we know doesn't work) and a blank piece of paper with an instruction to 'use professional judgement'.

Don gives an example that is useful here. He talks about a remote car starting device - a button to press as you approach your car so that it is ready to run by the time you get to it. This would be a reasonably easy thing to program and to model. In UML terms we would have an actor and a use case involving "unlock" to which we add "start", a defined system boundary makes it easy to write code. How do we get the developer to think about the consequences of this? In places other than the US most cars are manual - remotely starting them is going to run someone over.

Is it simply a matter of having a bigger bounding box in UML - could we instigate a new symbol - the dotted world box? How much bigger? One option is the inbuilt redundancy used by engineering, but in software this is not always a good thing - particularly with regard to complexity of logic. He says that the "how do we know when we're done?" question has long been the problem – how big a blank piece of paper are we giving people? He sees the answer as a series of concentric rings with diminishing impact and that you work these rings until you reach "the edge of your intellect". This doesn't solve the problem of the student/developer who is happy with having reached the edge of their intellect/impact in the first ring.

The intention of the super-system metaphors we propose is to prompt people to think of their computing developments in terms of biological models then the sustainability impacts would be more obvious without it becoming a box ticking exercise.

Whitehead (2007) also stresses the importance of the metaphor in software engineering. He argues that the "collaborative work to create software artefacts is the collaborative work to create models of the software system". The metaphor that underlies the model of collaboration is therefore critical to the outcome of software engineering. When this metaphor is wrong, the outcomes can also be wrong, as Winschiers and Paterson (2004) describe for ICT developments in

Namibia. They argue that the whole paradigm of ICT transfer needs a rethink if ICT is to meet the needs of non-Western cultures.

4.1 Biomimicry as metaphor

Biomimicry is described as:

Biomimicry (from bios, meaning life, and mimesis, meaning to imitate) is a design discipline that studies nature's best ideas and then imitates these designs and processes to solve human problems. Studying a leaf to invent a better solar cell is an example of this "innovation inspired by nature."

The core idea is that nature, imaginative by necessity, has already solved many of the problems we are grappling with. Animals, plants, and microbes are the consummate engineers. They have found what works, what is appropriate, and most important, what lasts here on Earth. This is the real news of biomimicry: After 3.8 billion years of research and development, failures are fossils, and what surrounds us is the secret to survival. (Biomimicry Institute, 2008)

The link to sustainability is strong. Biomimicry Guild argues that biomimicry design processes…

Are sustainable: Biomimicry follows Life's Principles. Life's Principles instruct us to: build from the bottom up, self-assemble, optimize rather than maximize, use free energy, cross-pollinate, embrace diversity, adapt and evolve, use life-friendly materials and processes, engage in symbiotic relationships, and enhance the bio-sphere. By following the principles life uses, you can create products and processes that are well adapted to life on earth. (Biomimicry Guild, 2008)

There is clearly much biologically inspired computing: genetic algorithms; neural networks; viruses and so on. At a genetic level nature certainly stores information, and expresses this information. Further, nature has communication systems, and sometimes this forms complex information systems.

The value of biomimicry has not gone unnoticed. There is a Bio-inspired computing conference (Suda and Tschudin 2006) with sessions including bio-inspired security systems, networks, search algorithms, congestion control. In *Biomimicry for optimisation, control and automation*, Passino (2004) has an extensive chapter on foraging behaviours as a basis for search optimisation. Vertegaal and Poupyrev (2008) recently edited a special edition of CACM on organic user interfaces:

These three general directions together comprise what we refer to in this section as Organic User Interfaces: User interfaces with non-planar displays that may actively or passively change shape via analogue physical inputs. We chose the term "organic" not only because of the technologies that underpin some of the most important developments in this area, that is, organic electronics, but also because of the inspiration provided by millions of organic shapes that we can observe in nature, forms of amazing variety, forms that are often transformable and flexible, naturally adaptable and evolvable, while extremely resilient and reliable at the same time. We see the future of computing flourishing with thousands of shapes of computing devices that will be as scalable, flexible, and transformable as organic life itself.

Clearly, there are biological metaphors that could be used in software development. The bigger question, is whether biomimicry itself could provide a metaphor to transform software development? – a super metaphor?

Almost all software development follows an engineering based project management paradigm. The underlying model is one of business, only very occasionally does science get a look in with scientific method sneaking in only in the form of test based development.

The development methodology described by Biomimicry Guild can be seen to closely align with the software development processes (see, for example, the Agile Development Framework, Mann and Smith 2006).

We've already seen that there could be benefits in biological metaphors. These could be facilitated with questions such as "How does Nature achieve this function in this environment?" Find champion adapters by asking "whose survival depends on this?".

Thompson (2008) argues that:

The team is in itself a super-organism and as such it needs to be treated in ways that enhance and support its complex and interconnected nature. If you can see the team as a whole and not as the mere aggregation of the individual parts that make it up, you can discover how much more productive, reliable and efficient a virtual team can be.

4.2 Ecosystem as metaphor

In *Sustainable Software Development: An Agile Perspective* Tate (2006) dismisses building construction as a metaphor for software development as building projects are largely static and have a clear finishing point. Instead he uses a coral reef metaphor for the software development industry since, like a coral reef, successful software "inhabits a complex and continually evolving ecosystem…" and "the development team needs to interact with and foster its ecosystem."

A successful software product plays the role of the underlying coral in the reef. The coral is what makes the entire ecosystem possible. None of the other organisms can survive without it. …continually evolving, growing, and changing. The reef is incredibly complex yet also fragile and vulnerable, just as a software product's ecosystem is prone to disruptive technologies or competition. This level of complexity, growth, constant change, evolution, and vulnerability I believe accurately portrays the complex world that today's software products inhabit.

While many Agile references to sustainability derive from a goal of maintaining work pace (through small teams and 40 hour weeks) and long term survival, nevertheless the notion of being aware of the environment in which the industry exists, and the need to nurture that environment, are apparent. By definition, lean methodologies are easier on people, use fewer resources and consider the needs of the stakeholders at every stage of the development and beyond (Boehm, 2004).

Tate describes sustainable development as a pace that can be maintained indefinitely but "this does not mean slower development - quite the opposite", nor a lessening of rigour:

Sustainable development requires a singular focus on a form of technical excellence that regularly provides useful software to customers while keeping the cost of change low….(and be) …able to deal with change, not be afraid of it or view it as a risk

Tate describes a mindset that the team is in it for the long haul as underlying sustainable development. The team adopts and fosters principles and practices that help them continually increase their efficiency, so that as the project gets larger and more complex and customer demands increase, the team can continue at the same pace while keeping quality high and sanity intact. They do this by continually minimising complexity, revisiting their plans, and paying attention to the health of their software and its ability to support change. The key points include a working product at all times; fix then code (not code then fix); test driven development and so on. These are all worthy goals, but this approach is clearly not informed by Blevis' lens of sustainability. It does not describe a culture that is considers the full scope of sustainability (An appreciation of importance of environmental, social, political and economic contexts…etc). One suspects though, that a workplace coming to grips with the ecosystem metaphor and following Tate's advice would be more likely to see the bigger picture.

The original title for Highsmith's (2002) Agile Software Development *Ecosystems* was Agile Software Development *Methodologies*. But, as Highsmith explains, it was changed "because (methodologies) didn't fit with the focal points of Agile development – people, relationships, and uncertainty". Ecosystem was used to "describe a holistic environment that includes three interwoven components - a "chaordic' perspective, collaboration values and principles, and a barely sufficient methodology"

Highsmith argues that the word ecosystem "conjures visions of living things and their interactions with each other", is "thought of as dynamic, ever changing environment in which people and organisations constantly initiate actions and respond to each other's actions" and "focuses us on the dynamic interactions of individuals and teams rather than on the static lines on organisation charts". Unfortunately, this is the last mention of what we might consider a sustainability or biologically inspired use of ecosystem throughout Highsmith.

In January 2008, Computerworld reported IDC's annual top ten predictions for the NZ computing industry. Number one is the impact of the skills crisis in NZ computing slowing investment. Number five is:

Green IT - issues behind the hype will shine through

What interests us most though is their repeated use of the term "ecosystem":

3. Emergence of the software as a service ecosystem - proposition will convert into penetration

and

7. Will the future of the channel ecosystem be revolution or evolution?

This use of "ecosystem" is perhaps unfortunate. It is not an indicator of a greater understanding through a biological metaphor. They are not alone in this. For example, Benkler's (2001) "series of technological shocks to the economic and technological ecosystem within which information is produced" demonstrates use of the word with no biological underpinnings at all.

Others demonstrate a greater understanding. Trimeche *et al.* (2005) describe end user experience in a multi-device ecosystem, meaning just *system* (although their system includes social context, spatio-temporal context). Similarly, Lentz and Bleizeffer

(2007) use IT ecosystems to infer complexity (and synonymously with *environment*) but they do discuss evolving technologies and integrated software and hardware interactions with associated human systems.

In the context of multimedia semantics, Scherp and Jain (2007) used ecosystem to be mean *framework*. Their Semantics Ecosystem is based on the relationships between physical world and mental model (plus products of the mind). This model has flows in both directions: analysis and synthesis. They do though include concepts borrowed from systems, particularly notions of feedback and emergent semantics.

David Messerschmitt and Clemens Szypersk's (2003) book Software Ecosystem renamed the software *industry* the software *ecosystem* and examined the complexities of our business in terms of larger systems. Unfortunately, Messerschmitt and Szypersk don't go far beyond the emphasis on the interconnectedness of a variety of issues surrounding software development.

In the preface Messerschmitt and Szypersk argue:

The software industry is itself very complex, with many complimentary products necessary to for a systems solution and complex alliances and standardisation processes needed to meet the needs of numerous stakeholders. Together, the software suppliers, standardisation bodies, content suppliers, service suppliers, and end-user organisations form a complex web of relationships. The "ecosystem" metaphor is truly descriptive.

The overarching theme of the book is that software is different.

Remember the benefits of a metaphor – common vision, a shared vocabulary, architecture and generativity. All of these, especially the last, benefit most from actually using the metaphor - borrowing from the idea to create new understandings. It is pointless to have a metaphor and then ignore it. Messerschmitt and Szypersk do recognise the value of cross disciplinary thought:

it is interesting to compare software to other industries, looking for parallels that might offer insights to the software industry

and

some insights and perspectives on software from the economics profession are described in chapter 9.

Nowhere though, do Messerschmitt and Szypersk actually use the ecosystem metaphor. And they are missing so many opportunities. They do take something like a systems approach - they look for simple laws to describe flows of information and then try to consider these in larger systems. On page 34, for example, they examine Moore's law and how it might operate over different scales.

Since previous technologies (like transportation or electrification) did not advance according to Moore's law, there must be something distinctive about the material information technologies

It is a shame that a whole field of environmental science is ignored: landscape ecology. It has much to say about scaling effects. Missed opportunities for new understandings continue:

- there is discussion about heterogeneity without reference to biodiversity;
- specialization and change without evolution;
- networks with out food webs;
- value chains without food chains;
- productivity without photosynthesis;
- models of development and maturity without succession;
- community without community;
- competition without competition;
- change without disturbance;
- components without demography;
- cooperation without relationships;
- interfaces without edge effects;

In addition to missed opportunities, things seem awry from an ecosystem perspective. "Environments" are described that in the various chapters that stop at the boundaries of that chapter (users, hardware or network). A model of the internet is provided where the heterogeneous internet consists of separated (but communicating) homogeneous environments. Other factors crucial to ecosystems are missing entirely: stochastic events, feedback, limiting factors.

Messerschmitt and Szypersk perceptively state:

The daily operation of most software systems requires human intervention

…these functions are called system administration.

Again, we could be so much further ahead if they then went on to borrow from the field of human ecosystems. Instead they have a chapter (9) on economics. There, the abstract discussions on supply, risk, complementarity, competitive advantage, time, and re-use are all crying out for some actual consideration of ecosystems.

Fortunately, other authors are further along this continuum. Hsi (2004) captures a computing ecosystem framework, using niche as "conceptual fitness".

In biology, ecosystems describe a defined envelope of physical, chemical, and biological processes within a space and time. More generally, an ecosystem is "a system of interacting species in a particular environment". We define a computing ecosystem as a set of use contexts that use computing to fulfil goals, contained within an environment of interest. A computing ecosystem can be a single person and a handheld PDA or a multi-national company of database management specialists. In a computing ecosystem, the organisms are the computing functions that users apply towards achieving their goals.

But then they seemingly abandon the metaphor and perform an analysis based on morphology, ontological excavation (from graph theory) and Use Case mapping.

Chilley (a panellist in Fraser *et al.* (2005) discussion on software robustness) describes how

Inherently complex, this new world is also more problematic than previously. Whereas traditional end to end systems can be likened to chains, linear systems where the weakest link will constrain the overall strength of the whole, the emerging service-driven systems are non linear, more likely to change their links and associations over time, changing their configuration to suit the demands of the moment. The structure of the underlying information ecosystem and the various cyber terrains it inhabits is also evolving and becoming more opaque.

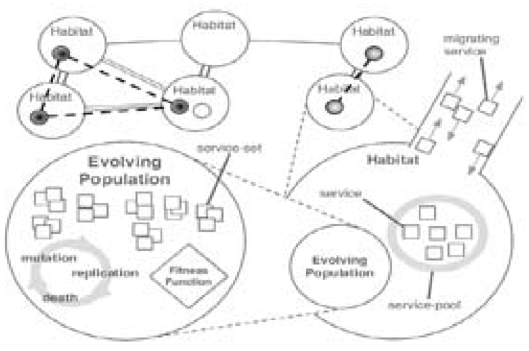
In the same panel Gabriel (2005) described systems that are called such systems autopoietic or self-creating.

We can note that the theory and practice of software as it exists and as it is taught is antithetical to this notion (current software is allopoietic or other creating—it is the manufacturing model). Therefore, I would claim that though it is possible to make some software systems robust (for a while) using current languages and techniques, we can never fully succeed by proceeding this way. Our languages and techniques—design, implementation, methodologies, etc—are inflexible (by design), having been created to be consistent with / mimic simplistic mathematical models. As long as we use arithmetic and crude cartons as the basis for all programming, we will move only slowly toward robustness.

A better approach is to compartmentalize (modularize / encapsulate / spatially compartmentalize) those parts of our software that do things like computing away from those parts that constitute its living architecture—"living" in the perhaps literal sense, but definitely not "living" in any sense related to how we compute today. And this latter type of software—the living part—will not be produced (or manufactured) in the same ways as the former, nor in the same sorts of languages.

McCormack (2007) describes the abstraction of natural ecosystem processes. Real ecosystems are highly complex and diverse systems, so we need to be careful about the processes we wish to mimic and how well suited (or otherwise) they are to the goal. McCormack's examples are in creative discovery: artworks: eg Eden: This work consists of a complex artificial ecosystem running in real-time on a two-dimensional lattice of cells. The ecosystem consists of three basic types of matter: rocks, biomass, and evolving agents. If a rock occupies a cell, agents or biomass may not. Agents attempting to move into a cell occupied by a rock will 'feel' pain and suffer energy loss.

Briscoe and De Wilde (2006) also explicitly use an ecosystem metaphor. They describe a business optimisation technique "inspired by natural ecosystems": a "digital ecosystem model". They illustrate this with figures that would not be out of place in Odum (1996). (Figure 1).



...in the Digital Ecosystem there will be multiple evolving populations in response to similar requests. This is shown in Figure 4 where the shades of the evolving populations indicates similarity in the requests being managed. For example, the four habitats in the top left may all be travel agents, the three with evolving populations may be looking for package holidays within the same continent. A great holiday package (services) found and used in one habitat, will then be migrated to the other connected habitats... where it will be integrated into any currently active evolving populations

Figure 1: Ecosystem image and text from Briscoe and De Wilde (2006)

Scacchi (2007) uses the ecosystem metaphor on a different level in an analysis of Free and Open Source development. There, recognising that the "growth patterns (of FOSS) seem to challenge the well established laws of software evolution", they suggest ecological thinking as the basis for further research:

"Persistent self-sustaining organisation or project community…Coevolution of interdependent software systems and standards for interoperation within an FOSS ecosystem represents an opportunity for research that investigates understanding such a software evolution process through studies supported by modelling and simulation techniques".

5 Conclusion

Blevis argues that we should use natural models and reflection as an underlying principle in interaction design. In this paper we have examined whether we could adopt biomimicry in software engineering. By doing so we might hope to work more sustainably and produce more sustainable products.

While biomimicry and bio-inspiration has been successful in specific areas of computing, we are yet to see a successful application at a paradigm level.

Instead, we have observed repeated use of terms such as ecosystem without any benefit the metaphor might have otherwise afforded. This approach entirely ignores ecological science; it does not even vaguely approach sustainability.

So, with the term software ecosystem suddenly gaining prominence along with the recognition of green computing, we suspect people are putting two and two together and getting five. Feel free to use the word, we are not word-use Luddites (despite ecosystem being explicitly coined for the ecological sciences), but please don't think that by putting ecosystem behind everything means you are acting sustainably. The term greenwash comes to mind.

A final word of caution. Despite the use of inherently natural models, and the strong links to sustainability given above, biological metaphors taken out of context may not be at all sustainable. Nature is not always a benign operation; much of what we see in nature is a result of fierce battles in the competition for scarce resources – primarily light. A whole systems approach would be crucial in such a paradigm.

6 References

Adams, J. B. (2008). "Computational science as a twenty-first century discipline in the liberal arts." J. Comput. Small Coll. **23**(5): 15-23.

Barnes P. (2007) Capitalism 3.0: A guide to reclaiming the commons. Berrett-Koehler San Francisco 207 http://capitalism3.com/files/Capitalism_3.0_Peter_Barnes.pdf

Beck, K. (2000) eXtreme programming Addison-Wesley 190 pages

Benkler, Y. (2001). "The battle over the institutional ecosystem in the digital environment." Commun. ACM **44**(2): 84-90.

Biomimicry Institute (2008). *"What is Biomimicry?"* Retrieved 12 June 2008 from Biomimicry Institute. Website: http://biomimicryinstitute.org/about-us/what-is-biomimicry.html

Biomimicry Guild (2008). *"Introduction to Biomimicry"* Retrieved 12 June 2008 from Biomimicry Guild. Website: http://www.biomimicryguild.com/guild_biomimicry.html

Blevis, E. (2007). Sustainable Interaction Design: Invention and disposal, renewal and reuse. Conference on Human Factors in Computing Systems, San Jose, California, ACM. 503 - 512

Blevis, E. and S. Blevis (2008). "SUSTAINABLY OURS Images of sustainable interactions: seeing with the lens of sustainability." interactions **15**(3): 27-29.

Boehm, B. and R. Turner (2004). Balancing Agility and Discipline – A Guide for the Perplexed. Boston, Addison Wesley.

Briscoe, G. and P. De Wilde (2006). Digital ecosystems: evolving service-orientated architectures. Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems, Cavalese, Italy, ACM.

Denning, P. J. (2007). "Computing is a natural science." Commun. ACM **50**(7): 13-18.

Fraser, S., D. Campara, et al. (2005). Fostering software robustness in an increasingly hostile world. Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, San Diego, CA, USA, ACM. 378-380

Gotterbarn, D. (2001) Keynote: Understanding and reducing project failure: the ethics of project management, In 14th Annual NACCQ

Conference (Ed, Mann, S.) NACCQ Napier, pp41-51.

Highsmith, J. (2002). Agile Software Development Ecosystems, Addison-Wesley. 404p

Hsi, I. (2004). Measuring the conceptual fitness of an application in a computing ecosystem. Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research, Newport Beach, CA, USA, ACM. 27-36

Jefferies, R.E. (2000) Extreme Programming Installed, Addison Wesley. Boston. 288

Lentz, J. L. and T. M. Bleizeffer (2007). IT ecosystems: evolved complexity and unintelligent design. Proceedings of the 2007 symposium on Computer human interaction for the management of information technology, Cambridge, Massachusetts, ACM.

Mann, S. and L. Smith (2006). Arriving at an agile framework for teaching software engineering. 19th Annual Conference of the National Advisory Committee on Computing Qualifications, Wellington, New Zealand, NACCQ in cooperation with ACM SIGCSE.

McCormack, J. (2007). Artificial ecosystems for creative discovery. Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England, ACM. 301-307

Messerschmitt, D., Szyperski, C. (2003) Software Ecosystem: Understanding an Indispensable Technology and Industry , MIT Press, 2003

Mogul, J. C. (2006). "Emergent (mis)behavior vs. complex software systems." SIGOPS Oper. Syst. Rev. **40**(4): 293-304.

Odum, E. P. (1996). Ecology: A bridge between science and society. MA, Sinauer.

Passino, K.M. (2004) Biomimicry for Optimization, Control, and Automation. Springer

Scacchi, W. (2007). Free/open source software development: recent research results and emerging opportunities. The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers, Dubrovnik, Croatia, ACM. 459-468

Suda, T. and C. Tschudin (2006). Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems, Cavalese, Italy, ACM.

Scherp, A. and R. Jain (2007). Towards an ecosystem for semantics. Workshop on multimedia information retrieval on The many faces of multimedia semantics. Augsburg, Bavaria, Germany, ACM. 3-11

Tate, K. (2005). Sustainable Software Development: An Agile Perspective, Addison Wesley Professional, NY.

Thompson, K. (2008) http://www.bioteams.com/

Trimeche, M., R. Suomela, et al. (2005). Enhancing end-user experience in a multi-device ecosystem. Proceedings of the 4th international conference on Mobile and ubiquitous multimedia, Christchurch, New Zealand, ACM. 19-25

Vertegaal, R. and I. Poupyrev (2008). "Introduction (Special section on Organic User Interfaces)." Commun. ACM **51**(6): 26-30.

Wake, W.C. (2002) What is the System Metaphor? In Wake, W.C. The XP Series. In: Extreme Programming Explored. Addison-Wesley Chp6

Whitehead, J. (2007). Collaboration in Software Engineering: A Roadmap. 2007 Future of Software Engineering, IEEE Computer Society. 214-225

Winschiers, H. and B. Paterson (2004). Sustainable software development. Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, Stellenbosch, Western Cape, South Africa, South African Institute for Computer Scientists and Information Technologists. 274-278