Biological Realms in Computer Science

Didier Verna
EPITA Research and Development Laboratory
didier@lrde.epita.fr

Abstract

In biology, evolution is usually seen as a tinkering process, different from what an engineer does when he plans the development of his systems. Recently, studies have shown that even in biology, there is a part of good engineering. As computer scientists, we have much more difficulty to admit that there is also a great deal of tinkering in what we do, and that our software systems behave more and more like biological realms every day. This essay relates my personal experience about this discovery.

Categories and Subject Descriptors C.m [*Computer Systems Organization*]: Miscellaneous; D.2.10 [*Software Engineering*]: Design; H.1.2 [*Information Systems/Models and Principles*]: User/Machine Systems – Human Factors

General Terms Design, Human Factors, Languages, Reliability

Keywords Trans-disciplinary models, software evolution, LaTeX

Prologue

It was in 2004, we were having an echography of our first child, and we were excited to see our little baby for the first time. The echographist was very nice and, something rare in the medical field, she was actually talking to us. Probably out of a wish to get us involved as parents, she started to explain things and enumerate all the good points as they were coming: you can see that your baby has the correct number of fingers, [...] the heart is well formed, [...] the neck is not too fat, so she's probably not a trisomic, [...] the vertebrae are all here, etc. etc.

And the funny thing is, as she was enumerating all the things that were okay, what I was actually hearing was all the things that could have gone wrong. So in the end, what was supposed to be a joyful moment turned out to be one of the scariest events in my whole life...

Part I
Origins

1. Transversality

TRANSVERSALITY. Thinking *horizontal*. Drawing bridges between apparently unrelated disciplines. What makes this so much fun? Why is it that so many people, especially in sciences, enroll in the Holy Quest of the Common Pattern?

As a researcher, I know that I have always loved to learn. A researcher, before anything else, is an eternal student. But from my own experience however, it seems that the simple joy of learning from your own discipline is nothing compared to that of learning from *outside* your world, especially when this learning process leads to new transversal connections. So it appears that there has got to be more to transversality than a simple thirst for knowledge.

The keys to transversality and its importance to humanity are probably *understanding* and *unification*. Religion, as well as Science, aim at explaining the universe as a whole. Franois Jacob [1977] puts it like this:

> *It is a requirement to the human brain to put order in the universe. [...] One may disagree with the explanatory systems offered by myths or magic, but one cannot deny them unity and coherence. [...] Actually, despite their differences, whether mythic,magic or scientific, all explanatory systems operate on a common principle. In the words of the physicist Jean Perrin [1914], the heart of the problem is always "to explain the complicated visible by some simple invisible".*

He further explains the importance of imagination in the process (whether mythic or scientific), but notes that in Science:

> *imagination is only a part of the game. At every step, it has to meet with experimentation and criticism.*

The interesting thing in this need to confront the possible with the actual is that experimentation leads to parceling: one can only test for the validity of a theory on very specific and localized behaviors.

Actually, the beginning of modern science can be dated from the time when such general questions as "How was the universe created?" [. . .] were replaced by such limited questions as "How does a stone fall?" [. . .]
Scientific knowledge thus appears to consist of isolated islands. In the history of sciences, important advances often come from bridging the gaps. They result from the recognition that two hitherto separate observations can be viewed from a new angle and seen to represent nothing but different facets of one phenomenon.

This is one particularly brilliant way of explaining *understanding*, as a means to *unification*. The same idea is underscored by Antoine Danchin [2009b] who sees in it a measure of scientific *progress*:

As Science progresses, there is a steady decrease in the number of postulates on which it has to rely for its development.

Finally, as a reaction to Franųois Jacob's argument, Uri Alon [2003] draws an explicit bridge between biology and engineering-based disciplines by noting a "fundamental scientific challenge: understanding the laws of nature that unite evolved and engineered systems".

2. Transversal models

IT so appears that transversality, as a response to a very profound urge of humanity, is a very old concern. Transversality is crucial in the scientific domain, to the point that it can encompass non-scientific disciplines. For instance, consider the tremendous impact of *Design Patterns*, originating from the architectural work of Alexander et al. [1977] on Software Engineering [Buschmann et al. 1996, 2007a,b; Gamma et al. 1994; Kircher and Jain 2004; Schmidt et al. 2000]. Linda Rising [2009] recently described how every computer scientist she knew was so excited about the publication of the GoF book:

We were all so excited we literally rushed to the book store. When I arrived there, the queue already extended outside the store, up to the pavement.

Sometimes, a transversality (or trans-disciplinary) concern is so deeply buried in our subconscious that we might not even realize it is there, or at least not completely grasp the extent to which it could be applied. With 30 years of retrospect, one might reconsider the work of Lehman [1980], undoubtedly a pioneer in the study of software evolution, and remark that his vocabulary is very biology-oriented: "life cycles", "evolution", "hostile environment" are just a few examples. Interestingly enough, no reference to biological studies appear in his founding paper (as we will see later, biologists had already started to connect with computer science at that time). His conclusion seems to indicate some level of awareness on the transversality of his work, although maybe not to the extent that would strike us today, as again, there is no mention of biology:

Many of the concepts and techniques presented in this paper could find wide applications outside the specific area of software systems, in other industries, and to the social and economic systems.

One particularly transversal topic in modern science, today, is the study of *networks*. Steven H. Strogatz [2001] explains why:

The current interest in network is part of a broader movement towards research on complex systems. In the words of Edward O. Wilson [1998], "The greatest challenge today, not just in cellular biology and ecology but in all of science, is the accurate and complete description of complex systems.

And so he concludes:

In the longer run, network thinking will become essential to all branches of science as we struggle to interpret the data pouring in from neurobiology, genomics, ecology, finance and the World Wide Web.

As a computer scientist, I think I have always been fascinated by transversality, perhaps without completely realizing it until recently, when I started the work described later in this essay. My parents being both biologists, I also grew up in this particular domain and I still feel a specific interest in the connections between computer science and biology. It so happens that these connections are both ancient and numerous. This is what we are going to explore now.

3. From computer science to biology

IN 1972, an interesting connection (not the first one) from computer science to biology was established in relation with the work of Alan Turing [1937] as described by Antoine Danchin [2009b]:

Carl Woese [1972] attempted to associate the downstream process of translation with the tape-reading metaphor of the Turing Machine, linking it with the creation of complexity during evolution.

Since then, many more bridges have been drawn, and biologists make a constant use of tools from computer science to gain a better understanding of their research field. This is particularly true in the recent domain of *systems biology* [Alon 2007], in which the study of complex biological systems, such as a cell's transcriptional regulatory network, relies heavily on network and graph theories [Yan et al. 2010]:

Over the past decade, the study of networks has emerged as an interdisciplinary research field [. . .]. Networks not only serve as backbones to study the emergent properties of complex systems, but they also provide an abstract framework that facilitates the cross-disciplinary comparison of different adaptive complex systems, ranging from biological systems to technological ones. Cross-disciplinary comparison between biological systems and commonplace systems such as organization hierarchies and engineering devices should be of particular interest to systems biologists.

Sometimes, the connections from computer science to biology bring us to puzzling extremes. For example, Iyer et al. [2001] demonstrated that in the study of protein functions, some experimental results might turn out to be less reliable than computer-based simulations ("in-silico" experiments, as Antoine Danchin would put it).

4. From biology to computer science

As biologists were getting inspiration from computer science tools, ideas and models, the same thing happened the other way around. One very early example of a connection from biology to computer science lies in the very term "Object-Oriented", the origin of which can be attributed to Alan Kay [Kay 1993; Ram 2003], although its meaning has largely diverged from its original one today:

It was probably in 1967 when someone asked me what I was doing, and I said: "It's object-oriented programming". [. . .] I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages.

Nowadays, many ideas and models from biology are used in computer science. Artificial Intelligence [Norvig 1992], most notably, is a discipline full of paradigms inspired by biology. Neural networks [Haykin 1994] get their inspiration from the way the brain works and are commonly used in applications such as character recognition. Genetic algorithms [Banzhaf et al. 1998] make computer programs evolve in a Darwinian fashion. And of course, computer viruses are similar to their biological counterparts in several ways, using hosts to replicate and spread themselves. These are just a few examples.

5. Discovery *vs.* invention

At this point, it is worth taking a step back to reconsider the very nature of the connections between computer science and biology that we have just exhibited. In all cases, the approach used by the scientist is a voluntary one. As mentioned earlier, imagination helps a lot, but in the end, there is always the will to grab a model here, and apply it there. When biologists use statistical methods, network or graph theories, they are purposedly applying "foreign" models to their own domain in order to get a better understanding of it, and for instance, be able to predict the behavior of complex biological systems. Conversely, when computer scientists use neural networks or genetic algorithms, they are also purposedly applying "foreign" models to their own domain in order to be able to solve a problem that would otherwise be difficult or simply impossible to solve with more conventional means.

These kinds of connections can be seen as *intentional* ones. They are made *on purpose*, with a very practical goal in mind. But what if connections existed before we even realized it? What if common, inherent behavioral patterns were to be found in both computer science and biology? In other words, are there any bridges to be *discovered* rather than *invented*?

As a matter of fact, there are. Some of them have even been suspected to exist for quite a while now. The notion of "genetic program" dates back to the 60s, and the term "program" refers explicitly to computer science. At that time, however, the link between genetic and computer programs was only seen as a metaphor. In a particularly illuminating article, Antoine Danchin [2009b] explains that there is much more to the genetic program than just a metaphor. He mentions the early involvement of information and Number Theory in biology, through the famous book by Douglas Hofstadter [1979]: "Hofstadter showed that the genetic code [. . .] behaves exactly like Gödel's code". He further emphasizes on the separation between the genetic program and the rest of the cell, just as a computer program is separated from the computer, hereby following the principles exposed by Turing [1946] and von Neumann [1958]. Evidence of this actual separation is provided by several studies. For instance, successful transplantation of an entire genome from one species to another has been achieved [Lartigue et al. 2007]. In the same vein, it has been experimentally proven possible to have cells perform logical tasks [Buchler et al. 2003; Elowitz and Leibler 2000]. Danchin further explains that it is this very separation between the genetic program and its execution which allows for DNA molecules to be expressed in foreign cells, hence forming the basis of what is called "genetic engineering".

Antoine Danchin is not the only one to support the idea that there is more to biology and computer science than a simple metaphor. Uri Alon [2003] underscores that the study of biological networks leads to the *discovery* of "good-engineering principles in biochemical circuitry", those principles being *modularity* [Hartwell et al. 1999], *robustness* [Savageau 1971] and "the use of recurring circuit elements" [Fell 1997]. He also points out that in spite of the torments of evolution, biological systems, however different, always converge towards these engineering principles.

All of these points seem indeed to indicate that there are patterns to be *discovered* rather than *invented*, which link computer science and biology together. Not only those links result from the will of applying trans-disciplinary models; some also *pre-exist* before our own awareness of them. This is perhaps what is most fascinating about this kind of transversal research. Discoveries, as opposed to inventions, give you an even stronger feeling that there is an order in the universe, far beyond our own comprehension. Every little piece of newly acquired understanding makes you realize that there is much more that you actually don't know. This idea can also be very scary, because it implicitly means that in the end, we are not in control of anything. This may not only explain the existence of myths and magic, but also, in the scientific domain, the fact that unifying theories are sometimes difficult to accept. For instance, Antoine Danchin [2009b] mentions that "few investigators would easily accept that there is more than a crude metaphor behind the analogy between cells and computers".

6. Tinkerers or engineers (or both) ?

MY personal tale begins here, with this intimate, and apparently shared conviction that there always was a deep connection between biology and computer science, but also, with this weird and frustrating feeling that something still doesn't quite add up. Something wrong in the picture. In retrospect, I think it took me *years* to finally understand what that feeling was. To be pedantically precise, it took me years, one night, one dream, and a Monday morning. In order that share that experience, we first need to shed some light on two very interesting personalities: the *tinkerer* and the *engineer*.

Franҫois Jacob [1977] describes evolution and natural selection as a *tinkering* process:

[Natural selection] works like a tinkerer — a tinkerer who does not know exactly what he is going to produce. [. . .] Evolution behaves like a tinkerer who, during eons upon eons would slowly modify his work [. . .] to adapt it progressively to its new use.

Jacob also underscores the fact that this tinkering process contradicts the idea that Nature achieves perfection:

Evolution is far from perfection. This is a point which was repeatedly stressed by Darwin who had to fight against the argument of perfect creation. In Origin of Species, Darwin [1859] emphasizes over and over again the structural or functional imperfections of the living world.

Retrospectively, I think that I did not really understand what this all meant until that very first echography of my daughter. It took me *that* to realize the extent to which Nature is imperfect. Sometimes, you have to get involved with your own guts to understand things. To understand that the simple birth of a viable child is in fact a *miracle* of Nature (some old studies [Boue and Boue 1975] have demonstrated that half of all conceptions are estimated to result in spontaneous abortion, and usually go unnoticed).

Because of the inherent imperfection of Nature, Franҫois Jacob [1977] also denies the comparison of natural selection with that of an engineered process:

The action of natural selection has often been compared to that of an engineer. This, however, does not seem to be a suitable comparison [. . .] because the engineer works according to a pre-conceived plan [and] because the objects produced by the engineer, at least by the good engineer, approach the level of perfection made possible by the technology of the time.

This point of view doesn't feel right however. Are we really engineers, as biologists like to see us, or are we in fact just tinkerers? Or both? As software developers, do we really work according to a "pre-conceived plan"? Is our software really approaching "the level of perfection made possible by the technology of our time"? The more I thought about it, the more it sounded unrealistic. On the contrary, as I grew up as a computer scientist, I realized more and more every day how *imperfect* the software I worked on (or with) was. As one of the core maintainers of the XEmacs[1] text editor, I can tell how imperfect the code-base is, with herds of developers having followed each other over time, and no-one understanding the application as a whole anymore. As a contributor to many other free software projects, I can tell how the majority of them, however well designed originally, turn into a collection of layers of patches on top of layers of patches eventually, making software evolution strangely resembling that of the human brain: a "superposition of new structures on old ones" as Jacob puts it.

7. The trigger

ONE last but particularly striking example is that of the LaTeX typographic system. As a maintainer of several LaTeX packages, I can tell how messy the LaTeX world is, but perhaps I should let David Kastrup do so instead [Walden 2006], in his typically eloquent and picturesque fashion:

It is a wildly inconsistent mishmash and hotchpotch of ad-hoc primitives and algorithmic solutions without noticeable streamlining and general concepts. A thing like a pervasive design or elegance is conspicuously absent. You can beat it around to make it fit most purposes, and even some typesetting purposes, but that is not perfection.
[. . .] have you taken a look at the complicated mess that the LaTeX core is, all with fragile and robust commands and encodings and whatever else?

The case of LaTeX is important because it is through it that I eventually came to realize what had been bothering me for years: there *is* a common pattern between biology and computer science which goes the opposite way of what is most commonly acknowledged.

[1] `http://www.xemacs.org`

Sometimes, we are much more tinkerers than we are engineers.

Because the LaTeX world is composed of thousands of software components like classes and styles, every LaTeX user faces the "compatibility nightmare" one day or another. With such great intercession capability at hand (LaTeX code being able to redefine itself at will), a time comes inevitably when the compilation of a document fails, due to class/style conflict. I tried several times to come up with a systematic approach, or at least some general principles on how to handle class/style cross-compatibility in a smooth and gentle manner, but ultimately failed, because the situation is just too complex.

Classes and styles are born, die, interact with each other, compete or cooperate, very much as living organisms do at the cellular level. Classes and styles evolve constantly, sometimes even in a backward-incompatible way. Styles may conflict not only with classes but with other styles as well. Styles may be made aware of classes or other styles, but classes may be made aware of styles as well. Then, there is the influence of the end-user who will combine all available material in a rather unpredictable way, possibly with his/her own personal additions, or even modifications to the available features.

This vicious circle basically never ends and leads to a paradoxical "If it ain't broke, then fix it" situation in which complex trickery is added to classes or styles, not to make them work out of the box, but to prevent potential breakages resulting from interactions with the outside world. In the end, the only realistic conclusion is that there is no solution to this problem, both because the system is too liberal, and because the human factor is too important. One cannot force a package author to write good quality (for some definition of "quality"), non-intrusive or even just bug-free code. One cannot force a package author to keep track of all potential conflicts with the rest of the LaTeX world, let alone fixing all of them by anticipation. One simply cannot prevent *software evolution*.

Facing this somewhat pessimistic conclusion, it is all the more intriguing to acknowledge the fact that the system still globally works (free software works, indeed [Raymond 1999]). Despite the complexity of what happens behind the curtain, documents *are* being produced, and in some way, seeing a freshly compiled document pop up on the screen is like witnessing a miracle of Nature. When it doesn't compile, you don't really know why, but when it does compile, you really don't know why. In spite of all the things that could go wrong, viable LaTeX documents can be produced, just as viable babies can be born. *That*, was my trigger.

It was after a Sunday evening of struggle with the ***CurVe*** class[2] which was conflicting with whatever style at that time. I was so frustrated to have lost my evening on yet another completely idiotic conflict that it pretty much ruined my night. I probably slept ruminating on the mess that LaTeX is, and the next Monday morning, I woke up after dreaming about my daughter's echography and at the same time with this vision of the LaTeX *biotope*, an emergent phenomenon whose global behavior cannot be comprehended, because it is in fact the result of a myriad of "macro"-interactions between smaller entities, themselves in perpetual evolution. I literally *saw* LaTeX documents as living beings defined by some geneTeX material (a term which I coined for the occasion) provided by their class, constantly evolving in order to survive gazillions of `\renewcommand` attacks inflicted upon them by those nasty little viruses called styles.

I also instantly *knew* that there was more to this vision than a simple metaphor. So I started to dig. I started to learn about cells and viruses and as my knowledge was increasing, I went from discovery to discovery. Ultimately, this research led to the publication of an article [Verna 2010] at the TeX Users Group conference, for the 2^5th (32th) birthday of TeX. The interested reader may retrieve this article online[3].

Part II
Ascension

8. The engineer as a tinkerer

LET us take a retrospective tour of what our Holy Quest of the Common Pattern has brought us. In "Evolution and Tinkering", Franọis Jacob [1977] was arguing that Nature works by tinkering, as opposed to engineering (which means planning and aiming towards perfection). In "Biological Networks: The Tinkerer as an Engineer", Uri Alon [2003] notes that under this assumption, it is very surprising to discover good engineering practice in evolved biological systems. In "Bacteria as Computers Making Computers", Antoine Danchin [2009b] goes even farther by exhibiting a very profound connection between living organisms and computers, between genetic code and software programs. What my personal tale has brought me, though, is a completely opposite vision of those things: some computer science systems behave as biological ones, instead of the other way around. In other words, for as much as the "Tinkerer as an Engineer" view is valid, so is the "Engineer as a Tinkerer" one. And I think that we, as computer scientists, should give much more credit to this view of things than what we are presently willing to.

How to do so? To quote Alon [2003] again: "The program of molecular biology is reverse-engineering on a grand scale". Well, if we are to restore the balance, we, as computer scientists, need to *reverse-tinker* our software systems on a

2 `http://lrde.epita.fr/~didier/software/latex.php`

3 `http://lrde.epita.fr/~didier/research/publis.php#verna.10.tug`

grand scale just as well. Not only for the LaTeX world. In doing so, we may very well discover some disturbing evidence on how our own creation actually works, and by that, hopefully sched some new light on software evolution in general [Brooks 1975; Lehman 1980].

Fortunately, the "engineer as a tinkerer" view, in other words, the biology-oriented perspective on software evolution seems to be gaining some momentum. Stephanie Forrest [2010] for instance, seems to acknowledge this perspective:

As programmers, we like to think of software as the product of our intelligent design, carefully crafted to meet well-specified goals. In reality, software evolves inadvertently through the actions of many individual programmers, often leading to unanticipated consequences. Large complex software systems are subject to constraints similar to those faced by evolving biological systems, and we have much to gain by viewing software through the lens of evolutionary biology.

But how can we explain that so many people are still reluctant to embrace this perspective then? How is it that we still "like to think of software as the product of our intelligent design"? Probably because there is a widespread confusion between *determinism* and *predictability*.

9. Determinism *vs.* predictability

COMPUTER scientists evolve in an intrinsically deterministic environment. The computer is like the Terminator, blindly and coldly executing the instructions that it has been given. When the computer does not do what we want, it still does what the software tells it to do. In turn, the software is just what we wrote, and sometimes, what we wrote may not be exactly what we *intended* to write. But the computer will still obey the instructions it has been given. When there is a bug (and there are [Hovemeyerand and Pugh 2004]!), the bug is not in the computer, but in the computer scientist who has programmed it.

Yet, we have a tendency to think that because the machine is so deterministic, its behavior is also completely predictable. This is unfortunately not true. How can I predict that my program is going to execute normally when there is in fact a bug in the compiler that I have used? If I change the name of one internal macro in my *CurVe* class for LaTeX 2_ε, how can I predict that this new version will not introduce a clash with one of the 3000+ classes or styles in the TeXlive distribution[4], let alone with every single user's internal hacks? Although determinism can certainly help in explaining behaviors *a posteriori*, it certainly is no crystal ball.

In that context, the unpredictability problem is to be seen in relation with the notion of *deterministic chaos* (or "butterfly effect", a term attributed to meteorologist Lorentz) [Heylighen 2002]:

In the last few decades, physicists have become aware that even the systems studied by classical mechanics can behave in an intrinsically unpredictable manner. Although such a system may be perfectly deterministic in principle, its behavior is completely unpredictable in practice. This phenomenon was called deterministic chaos.

Because of its non-linear nature, a chaotic system would require an extensive (possibly infinitely precise) knowledge of its initial or current conditions in order to be able to predict its behavior. Unfortunately, this knowledge is often impossible to acquire. For instance, it is clearly not possible to know the exact topology of every LaTeX installation on Earth, in order to predict whether a specific change in a style will introduce a new clash somewhere. The LaTeX ecosystem, in fact, is currently in a state of deterministic chaos, probably as are many other complex software systems.

According to Antoine Danchin [2009b], the lack of distinction between determinism and predictability seems to affect Biology just as well. The *reductionists* [Lewontin 1993] deny the possibility of a deterministic molecular biology because they think it would render Nature completely predictable. On the contrary, Danchin stipulates that the "lack of prediction is not due to a lack of determinism, but to a creative action that results in novel information", hereby also arguing that information [Shannon and Weaver 1949] should be regarded as a first-class ingredient of Nature, along with energy, matter, space and time. Danchin also underscores that a key ingredient in creating "novel information" lies in such controversial topics as *adaptive mutations* [Cairns and Miller 1988; Danchin 1988], in essence, genetic programs resulting in creating new genetic programs or modifying themselves. This is in fact a well known behavior in high level reflexive programming languages. It is known as *intercession*.

10. Predictability *vs.* control

UNPREDICTABILITY doesn't quite answer the question though. After all, unpredictability can be an achievement of it own. For instance, the ability of artificial intelligence, through machine learning, to deliver programs that fulfill their goal in an unpredictable way is considered a success. So why is it, really, that we still don't want to see the biological realm there is in computer science? Apart from the tinkering aspect, which is obviously not very shiny, a deeper reason lies in the notion of *control*.

Biologists (in fact, humans in general) knew from the start that they didn't have any kind of control on their field of investigation. How can we possibly control the creation of Nature when we are in fact part of that very creation? If we could, we would be Gods. There was a time, on the other hand, when computers were so limited that the whole chain, from the program to the electrical signals crossing the transistors, was under the control of a single person. In

[4] `http://www.tug.org/texlive`

a simple system, determinism leads to predictability which in turn provides control. And we *love* to be in control. The problem is that we still more or less think that this assertion is valid. To quote Gabriel and Goldman [2006]:

We feel we are in control of our current software applications because they are the result of a conscious design process based on explicit specifications and they undergo rigorous testing.

But at the same time, we have to face the truth: as software complexity increases, we *lose* control over it. The fact that it can be so frustrating to lose control proves that we are aware of it already, at least partially. Frustration? Yes, although maybe coupled with a feeling of shame, because contrary to the biologists, what we lose control on is a product of our own creation (or is it?). Gabriel and Goldman [2006] also quote Weizenbaum [1976] about this:

The programmer moves in a world entirely of his own making. [...] Indeed, the compulsive programmer's excitement rises to a fevered pitch when he is on the trail of a most recalcitrant error, when everything ought to work but the computer nevertheless reproaches him by misbehaving in a number of mysterious, apparently unrelated ways. It is then that the system the programmer has created gives every evidence of having taken on a life of its own, and certainly, of having slipped from his control. [...] For, under such circumstances, the misbehaving artifact is, in fact, the programmer's own creation.

However frustrated and shameful we might feel, maybe we should consider the possibility that we will never regain control on our software. Many examples already show this trend. In my personal experience, I estimate that 80% of the time I spent on LaTeX recently was devoted, not to actually develop the software, but to fix conflicts and incompatibilities. In other words, because the software is out of control, it spends most of its time on the surgeon's table instead of growing as a normal "living" being. Therefore it should be clear that by desperately trying to regain control over it, what I did was in fact to prevent its development and evolution, not unlike a overly possessive parent would imprison their child.

11. Rise of the machines

If we are to loose control of our software, then who is going to control it? The only possible answer to this question is both fascinating and scary: the software needs to control itself. Science-fiction authors have been developing this idea for a long time, and beyond just software, to the extent of complete autonomous machines (robots). Computers making computers in some way. Interestingly enough, our fear of losing control is very apparent in such science-fiction showpieces as Terminator, Matrix or Battlestar Galactica, as every time we lose control on our machines, it leads to a disaster[5].

Is our software already in control of itself? Not by a long shot, and here again, biology precedes us. Antoine Danchin [2009b] explains that:

the paleome includes a set of genes that are not essential for life under laboratory growth conditions [Fang et al. 2005]. Many of these genes code for maintenance and repair, and may be involved in perpetuating life by restoring accuracy and even creating information during the reproduction process [Danchin 2009a].

Maintenance and repair. This sounds pretty much like what I've been doing 80% of the time on my LaTeX packages recently. The only difference is that whereas I am doing it *manually*, biological cells seem to do it *automatically*. Therefore, we can argue that the machines will have risen for good when they are able to control themselves, watch over themselves, maintain and repair themselves, in other words, *recreate* themselves perpetually. This vision of reflexive systems is called *autopoiesis* by Maturana [1981] and a particularly fascinating vision of such a future for our software is given by Gabriel and Goldman [2006] in their essay "Conscientious Software".

12. Darwin's radio

In a way, software has already started to take control over itself, although in a rather limited fashion yet. People are investigating on using techniques inspired from biology in order to provide for automatic program maintenance and repair [Weimer et al. 2010]. Garbage collection is an example of a process by which the system "fixes" its own memory leaks, by removing explicit control over memory allocation/deallocation from the programmer. The programmer loses control, the program gets it. Gabriel and Goldman [2006] also point out that the notion of garbage collection actually appeared out of John McCarthy's desire to *not* control the memory explicitly. In other words, his desire to *yield* control to the machine [McCarthy 1978].

The garbage collection example is interesting because it reifies the idea of a system watching over itself: when you deliver a Lisp application for instance, you deliver not only the part which does the actual work, but also a part which is supposed to maintain the whole thing in a healthy state with respect to memory management. We could also argue that the ability of Lisp to deliver applications embedding a compiler and even a debugger go more or less along with the same lines. The global system is delivered with parts that are

[5] One notable exception to this rule is the famous character Data in Star Trek. Data considers himself imperfect because he doesn't feel emotion, and only strives for more humanity. It is amusing to realize that his notion of perfection actually consists in inheriting the flaws of his creator. Being as imperfect as Nature can be.

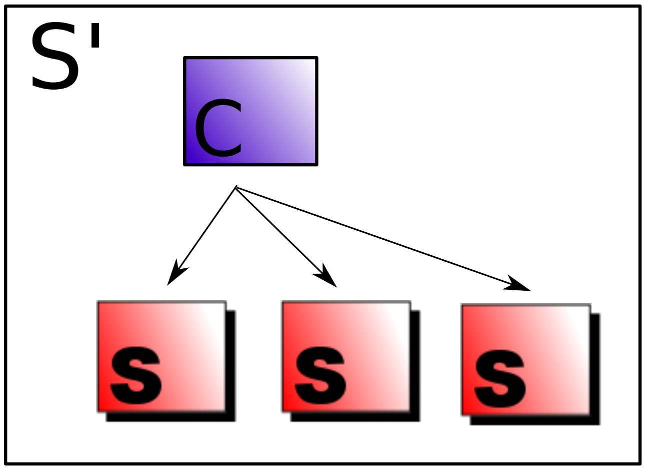

Figure 1. Meta-system transition

devoted to maintenance and repair of the system itself, rather than just focusing on the task to perform.

A part of a system in charge of controlling the rest of it can be called a *meta-system*, which brings an interesting parallel with the notion of "meta-system transition", a concept originally developed by Valentin Turchin [Heylighen et al. 1995; Turchin 1977]. Starting from a set of systems S (figure 1), a meta-system transition is the essentially recursive process by which those systems are agglomerated and collectively supervised by an additional control module, in order to form an upper-level system S'.

The classic example in biology is the emergence of multicellular organisms, through a process of (cell) duplication, differentiation and establishment of control. In such a case, the transition is *homogeneous*. A case of *heterogeneous* transition would be based on the agglomeration of originally different and independent organisms, such as those living in *symbiosis*.

The use of the meta-system transitions theory to model evolution exhibits a process by which systems of increasing complexity emerge spontaneously:

Most of the time this complexity increase, and evolution in general, occurs rather slowly or continuously, but during certain periods evolution accelerates spectacularly. This results in changes which from a long term perspective may be viewed as momentous events, separating discrete types of organization.

In a fascinating science-fiction (not so much!) novel, "Darwin's Radio", Greg Bear depicts the sudden birth of a new subspecies of humans, a phenomenon surprisingly close to a meta-system transition.

I think we have much to gain by regarding software evolution in terms of meta-system transitions. The introduction of garbage collection, for instance, can be considered as one. I have the feeling that our software might be on the verge of a new transition, towards complete autonomy, that is. As we have seen already, more and more people are starting to work in that direction. Whereas the "rise of the machines" is still regarded as science-fiction, it seems that the process has in fact already started. As our software grows more complex and less controllable, it looks very much as if new meta-systems were emerging, only without proper control modules yet. It's up to us to invent them.

Epilogue

This journey in the transversal world of biology and computer science has brought me a lot of things. For one, I hadn't realized to what extent concepts from engineering are applicable to biology. More and more biologists seem to acknowledge the intricate relations between those two worlds. Some of them push the relation as far as considering cells as computers and claiming that life may turn out to be understandable, if not predictable. I have also come to realize that the opposite view is true however. For as much as there's computer science in biology, there's also biology in computer science. Nature is engineered as much as software is tinkered, and I think we need to acknowledge that much more than we are currently willing to.

It may help to eventually realize that the silicon-based world is much more biological than we ever would have thought, and as such, maybe we haven't actually invented it. Only discovered it. A biological realm we didn't want to think about before.

It also seems that we are living in an era of paradox. Because complexity can only increase, we are gradually losing control on our software, as it grows bigger, more versatile, more reactive, more expressive. This is to the point, now, that it may be worth considering not to try and control it anymore, but to program it to control itself. Software making software to control software. It is a paradox because we usually like to think that understanding means prediction and hence control. On the contrary, it appears that the more we understand the world, whether carbon-based or silicon-based, the more we realize that we may need to let go of control. But do we really want to do that? Is it really a good thing? This question alone is probably worth another complete story. . .

References

C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

U. Alon. Biological networks: The tinkerer as an engineer. *Science*, 301(5641):1866–1867, Sept. 2003.

U. Alon. *An Introduction to Systems Biology*. Chapman & Hall, 2007.

W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1998.

A. Boue and J. G. Boue. *Physiology and Genetics of Reproduction*, 4b:317, 1975.

F. Brooks. *The Mythical Man-Month*. Addison Wesley, 1975.

N. E. Buchler, U. Gerland, and T. Hwa. On schemes of combinatorial transcription logic. In *National Academy of Sciences*, number 9, pages 5136–5141, Apr. 2003.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture*, volume 1. Wiley, 1996.

F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*, volume 4. Wiley, 2007a.

F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*, volume 5. Wiley, 2007b.

J. Cairns and J. O. S. Miller. The origin of mutants. *Nature*, 335: 142–145, 1988.

A. Danchin. Origins of mutants disputed. *Nature*, 336:527, 1988.

A. Danchin. Natural selection and immortality. *Biogerontology*, 10 (4):503–516, Aug. 2009a.

A. Danchin. Bacteria as computers making computers. *FEMS Microbiol Rev.*, 33(1):3–26, Jan. 2009b.

C. Darwin. *The Origin of Species by Means of Natural Selection*. John Murray, 1859.

M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, Jan. 2000.

G. Fang, E. P. Rocha, and A. Danchin. How essential are non essential genes? *Molecular Biology and Evolution*, 22:2147–2156, 2005.

D. Fell. *Understanding the Control of Metabolism*. Portland Press, 1997.

S. Forrest. The case for evolvable software. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, OOPSLA'10, pages 1–1, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0203-6. doi: http://doi.acm.org/10.1145/1869459.1869539. URL `http://doi.acm.org/10.1145/1869459.1869539`.

R. P. Gabriel and R. Goldman. Conscientious software. In *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 433–450, New York, NY, USA, 2006. ACM. ISBN 1-59593-348-4. doi: http://doi.acm.org/10.1145/1167473.1167510. URL `http://doi.acm.org/10.1145/1167473.1167510`.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

L. Hartwell, J. J. Hopfield, S. Leibler, and A. Murray. From molecular to modular cell biology. *Nature*, 402(6761):47–52, Dec. 1999.

S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1994. ISBN 0023527617.

F. Heylighen. `http://pespmc1.vub.ac.be/chaos.html`, 2002.

F. Heylighen, C. Joslyn, and V. Turchin, editors. *The Quantum of Evolution. Toward a Theory of Metasystem Transitions*, volume 45 of *World Futures: the journal of general evolution*, 1995. Gordon and Breach.

D. Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, 1979.

D. Hovemeyerand and W. Pugh. Finding bugs is easy. In *Onward!*, Dec. 2004.

L. M. Iyer, L. Aravind, P. Bork, K. Hofmann, A. R. Mushegian, I. B. Zhulin, and E. V. Koonin. Quod erat demonstrandum? the mystery of experimental validation of apparently erroneous computational analyses of protein sequences. *Genome Biology*, 2(12), 2001.

F. Jacob. Evolution and tinkering. *Science*, 196(4295):1161–1166, June 1977.

A. C. Kay. The early history of smalltalk. In *Second ACM SIGPLAN Conference on History of Programming Languages*, HOPL-II, pages 69–95, New York, NY, USA, 1993. ACM. ISBN 0-89791-570-4. doi: http://doi.acm.org/10.1145/154766.155364. URL `http://doi.acm.org/10.1145/154766.155364`.

M. Kircher and P. Jain. *Pattern-Oriented Software Architecture: Patterns for Resource Management*, volume 3. Wiley, 2004.

C. Lartigue, J. I. Glass, N. Alperovich, R. Pieper, P. P. Parmar, C. A. H. III, H. O. Smith, and J. C. Venter. Genome transplantation in bacteria: Changing one species to another. *Science*, 317 (5838):632–638, Aug. 2007.

M. M. Lehman. Programs, life cycles, and laws of software evolution. In *IEEE*, volume 68, pages 1060–1076, Sept. 1980.

R. C. Lewontin. *The Doctrine of DNA: Biology as Ideology*. Penguin Books, London, 1993.

H. Maturana. Autopoiesis. In M. Zeleny, editor, *Autopoiesis: A Theory of Living Organization*, page 21 30. New York, North Holland, 1981.

J. McCarthy. History of programming languages. In *The First ACM SIGPLAN Conference on the History of Programming Languages*, pages 217 –223, Los Angeles, 1978.

P. Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, 1992.

J. Perrin. *Les Atomes*. Alcan, 1914.

S. Ram. Dr. alan kay on the meaning of "object-oriented programming". `http://www.purl.org/stefan_ram/pub/doc_kay_oop_en`, 2003.

E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly, 1999.

L. Rising. The benefits of abstraction in patterns. ACCU Keynote, 2009.

M. A. Savageau. Parameter sensitivity as a criterion for evaluating and omparing the performance of biochemical systems. *Nature*, 229:542–544, 1971.

D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, volume 2. Wiley, 2000.

C. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois, Illinois, IL, 1949.

S. H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, Mar. 2001.

V. Turchin. *The Phenomenon of Science. A Cybernetic Approach to Human Evolution*. Columbia University Press, 1977.

A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *London Mathematical Society*, volume 2, pages 230–265, 1937.

A. M. Turing. A. M. turing's ace report of 1946 and other papers. *Charles Babbage Institute Reprint Series for the History of Computing*, 10, 1946.

D. Verna. Classes, styles, conflicts: The biological realm of LaTeX. In B. Beeton and K. Berry, editors, *TUGboat*, volume 31, pages 162–172. TeX Users Group, 2010.

J. von Neumann. *The Computer and the Brain*. Yale University Press, 1958.

D. Walden. An interview with david kastrup. `http://www.tug.org/interviews/kastrup.html`, 2006.

W. Weimer, S. Forrest, and C. L. G. T. Nguyen. Automatic program repair with evolutionary computation. *Communications of the ACM Research Highlight*, 53(5):109–116, 2010.

J. Weizenbaum. *Computer Power and Human Reason: From Judgement to Calculation*. W. H. Freeman & Company, 1976.

E. O. Wilson. *Consilience: The Unity of Knowledge*. Knopf, 1998.

C. Woese. The evolution of cellular tape reading processes and macromolecular complexity. In *Brookhaven Symposia on Biology*, volume 23, pages 326–365, 1972.

K.-K. Yan, G. Fang, N. Bhardwaj, R. P. Alexander, and M. Gerstein. Comparing genomes to computer operating systems in terms of the topology and evolution of their regulatory control networks. In *National Academy of Sciences*, number 20, pages 9186–9191, May 2010.