# ESSAYS ON SCIENCE AND SOCIETY:

# Software Bioengineering


## Christoph Adami

*Chris Adami is a Principal Scientist in the Exploration Systems Autonomy Section of the Jet Propulsion Laboratory, California Institute of Technology, and Director of the Digital Life Laboratory at the California Institute of Technology. He is the author of "Introduction to Artificial Life" (reviewed in Science 1998 May 8; 280: 849-850.)*


Current trends in software development indicate that we will soon reach the point where program size, together with the increasingly intricate ways in which software needs to interact with other programs, will lead to an impasse. This impasse, observable from the outside by catastrophic system failures with decreasing "uptime" in between, is not unlike what must have befallen the earliest self-replicating molecular systems in the era of the putative RNA world [1]. For self-replicating molecules, the number of copy-errors per molecule scales with the length of the sequence, and puts a stop to further evolution if the average number of errors per sequence becomes too large starts to exceed one. In software design, a constant rate of "bugs" introduced by the programmer also puts a limit on program size, which we may already have approached in certain areas.

While the origin of errors is different in the two kinds of "software" just described, the end result is the same because in both cases the maladapted programs' "fitness" cannot be improved without a major paradigm change in the design process. For self-replicating molecules, the selective pressure to survive in noisy environments has pierced size-limiting boundaries several times in the history of evolution, from the evolution of DNA-based coding and sophisticated molecular error correction mechanisms to perhaps even the evolution of sexual cross-over [1]. Standard software design, meaning the single-author or group efforts to create large packages such as operating systems and office suites, must adjust to the limits of human design capabilities with a major shift in the engineering process of its own. The dilemma created by the "complexity wall", moreover, is not limited to the office domain. The software that operates and controls complex machinery such as planes and spacecraft (flight software) is perhaps even more sensitive to the looming barrier because its failure can translate to the loss of sizeable investments and human life.

From the software engineering point of view, that a 3 billion-line program with roughly 30,000 or more subroutines (such as the human genome) can function at all is a miracle beyond comprehension. However, this software suite is not a product of design, but rather a product of *evolution*. While the mechanisms enabling the evolution of biocomplexity have been studied since the Darwinian revolution, software evolution in the computational rather than biochemical realm is still in its embryonic stage.

The first cautious steps towards wholesale software evolution are now being taken, and corporate research and development, as well as governmental funding agencies, are awakening to the promise, as well as the difficulties, of such an endeavor. Microsoft Research started funding limited research into the evolvability of computer programs in the mid-nineties, whereas NASA has recently initiated a massive effort geared at creating a Center of Bioengineering for the Exploration of Space (CBEES), one of whose objectives is the exploration of the principles underlying the evolution of complex hardware and software systems

Historically, genetic mechanisms in Computer Science have been used in search as well as in synthesis, via the paradigms of Genetic Algorithms and Evolutionary Computation. However, these branches have delivered only limited success because of their limited scope and the often-unquestioned usage of "black box" evolution algorithms that ignore some of the enabling mechanisms that make biological evolution so extraordinarily successful. For example, we now have preliminary evidence that a good percentage of the genes of complex organisms code not for organismic function per se but rather ensure their fault tolerance [2]. While extra robustness genes thus increase genome size, their presence appears to be much more valuable than their contribution to the genetic load, and even may contribute to evolvability. On the contrary, robustness is not a target in standard genetic algorithm fitness functions, resulting in fragile code that is difficult to evolve past its initial adaptation. Equally importantly, the main mechanism creating diversity in Genetic Algorithms, code cross-over, may even *reduce* fitness via loss of alleles if deleterious mutations interact antagonistically [3] or if the supply of beneficial mutations is low-. Thus, a concerted effort in software evolution must have at its basis a sound understanding of the selective pressures that affect genomes, as well as the interactions between populations of genomes and the environment in which they evolve.

At the California Institute of Technology's Digital Life Laboratory [4], we have been using populations of computer programs to study the basic mechanisms of genetic evolution *in silico*. Digital Life, inspired by the metaphorical similarity between genetic codes and computer programs [5], in fact achieves their logical synthesis. Populations of programs live in and adapt to an artificial world created entirely by the researcher, but they must replicate their code in a very real manner in the memory of the computer. Programs grow in complexity by acquiring computational genes for survival [6], and develop mutational robustness [7]. Complex programs evolved in this manner are virtually unintelligible to human programmers because their fitness function is implicit and involves much more than the target a human programmer might set. At the same time, these programs achieve a level of success, in the world to which they are adapting, that is far beyond what human design can achieve.

The *de novo* synthesis of functional programs in simulated environments is not the only alternative to conventional software design. Bio-inspired software-testing environments and human-assisted code evolution are precursors to the pure bioengineering of software that could significantly improve the standard methods. Similarly, "open-source" software development seems to owe its success, according to Linux creator Linus Torvalds, to the conspiracy of "sheer luck" and "survival of the fittest", operating on a pool of

programmers that modify the freely available source code. Indeed, he insists that "…(Linux) wasn't designed", but instead "(it) grew. It grew with a lot of mutations, and because the mutations were less than random, they were faster and more directed than (…) in DNA" [8].

Whatever the incarnation, it is evolution that must someday come to the rescue of design in software engineering. Because of this inevitability, we need to extract universal features from the enabling and limiting factors in the evolution of biocomplexity, and apply them to code evolution. Otherwise, society has to face living with size-limited software, as many viruses still do.


[1] Ridley, M. *The Cooperative Gene* (The Free Press, New York, 2001).
[2] Wagner, A. *Nature Genetics* **24**,355-361 (2000).
[3] Kondrashov, A. *Nature* **336**, 435-440 (1988).
[5] http://dllab.caltech.edu
[5] Ray, T.S, in Artificial Life II (eds. Langton, C.G., Taylor, C., Farmer, J.D., & Rasmussen, S.) p. 372-408 (Addison Wesley, Redwood City, 1991).
[6] Adami, C., Ofria, C., & Collier, T.C. *Proc. Natl. Acad. Sci. USA* **97,** 4463-4468 (2000).
[7] Wilke, C.O., Wang, J.L., Ofria, C., Lenski, R.E., & Adami, C. *Nature* **412**, 331-333.
[8] L. Torvalds, www.kerneltrap.org, December 01, 2001.