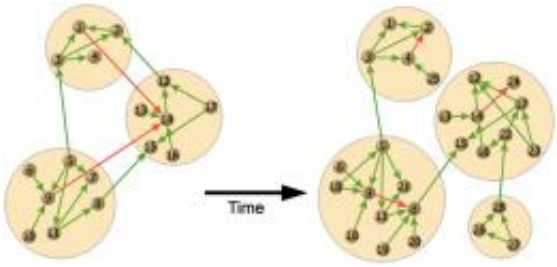
Tinkering with evolution: Ecological implications of modular software networks

December 19 2011, by Stuart Mason Dambrot



Evolution of the modular structure of the network of dependencies between packages of the Debian GNU/Linux operating system. Packages are represented by nodes. A green arrow from package i to package j indicates that package i depends on package j, and a red arrow indicates that package i has a conflict with package j. Packages within a module (depicted by a big circle) have many dependencies between themselves and only a few with packages from other modules. During the growth of the operating system, the modular structure of the network of dependencies has increased: (I) The new packages added in successive releases depended mainly on previously existing packages within the same module, and hence, the size of the modules created in earlier releases increased over time; (ii) the number of modules also increased, although the new modules consisted only of a few new packages; and (iii) the relative number of dependencies between packages from different modules decreased. Moreover, the relative number of conflicts between packages from different modules decreased, whereas those within modules increased through the different releases of the operating system. Copyright © PNAS, doi: 10.1073/pnas.1115960108

(PhysOrg.com) -- In the 1960s, Dr. Lawrence J. Fogel introduced what

would come to be known as *evolutionary programming* to the nascent field of Artificial Intelligence in an attempt to produce intelligent software without relying on neural networks modeled on the brain or human expert-based heuristic programming. Now, researchers in the Department of Ecology and Evolutionary Biology at Princeton University have shown the inverse – namely, that network theory, when applied to software systems, provides surprising insights into biology, ecology and evolution. Specifically, they explored evolutionary behavior in complex systems by analyzing how the Debian GNU/Linux operating system utilizes modular code. The researchers found that how the network becomes more modular over time in various OS installations often parallels that of ecological relationships between interacting species.

Lead researcher Miguel A. Fortuna, who worked with Juan A. Bonachela and Prof. Simon A. Levin, Director of Princeton's Center for BioComplexity, describes the main challenges they encountered in designing and implementing the methods used to analyze OS the evolution. "The main difficulty we had was getting, organizing, and storing the data," says Fortuna. "Notice that the network of interdependent packages of the last release analyzed was composed by more than 100,000 dependencies. "This complexity required that they use structuring query languages (SQL) for managing databases. "We were very careful when identifying software packages through different release – sometimes there could be different versions of the same package within the same release due to the improvements made by developers."

While Fortuna notes that quantifying the increase of the code's modular structure time was the main insight of their study, he points out that reuse of code and software's hierarchical structure were suggested by the pioneering work of Ricard V. Solé and Sergi Valverde in the early 2000s. "The interest that our paper has drawn has helped us to discover

work we did not know about software systems. The idea of using the network of dependencies and conflicts of different releases of the Debian operating system as a case study has facilitated the understanding of how code development evolves over time without the need to go deeper into the details of the code itself."

Another key innovation cited by Fortuna was the team's use of a very precise method to detect the modular structure of the operating system. "We borrowed an algorithm developed by physicists and widely used in ecology nowadays. In fact, this work has been constantly enriched by an interdisciplinary mixture of ideas from biology and physics."

The team already has its eye on ways of improving and extending the current experimental design. "The most important follow-up of our study would be the exploration of proprietary software like the Microsoft Windows operating system," Fortuna comments. "Since Debian is the result of a volunteer effort to create a free operating system, you have the freedom to distribute copies, receive source code, modify the software or use pieces of it in new free programs. The question then becomes, what does the software development pattern looks like when the company developing code doesn't offer this freedom to their users? A comparison of the structure of both development strategies would be more than interesting."

They are also developing a dynamical model to mimic the growth of Debian over time – an effort which, if successful, might let them predict how many packages, dependencies, and conflicts will arise in the next release of the operating system. An interesting question would be," he conjectures, "if there are limits to the number of packages that an operating system can offer to the users without jeopardizing its functionality and robustness. Following our analogy with the biological evolution, we could ask if there is a limit to biodiversity, that is, to the number of species that can coexist in our planet."

Regarding potential analogies with evolution and ecology, Fortuna points to macroevolution – that is, speciation and extinction processes – that he sees as being in some ways equivalent to the creation of new packages and the deprecation of those rendered obsolete from one release to the next. “Does the probability of a species becoming extinct depend on how long it’s been on the planet? In other words, are the most ancient species, like crocodiles, the ones with higher risk of extinction? We can formulate the question, which was already explored by Van Valen in the 1970's, by replacing *species with software packages*. Why do some packages not exist after a subsequent release? Does a new software package created in one of the earliest releases have a high probability to persist over time? What does it depend on? We can calculate these probabilities following the identity of the packages of the Debian operating system through time. The data to do it are available, and we therefore might learn something from software studies that help us answer the biological question – because evolution works as a tinkerer in both cases.”

In relation to the ecological processes, Fortuna illustrates, “When an oceanic island is created colonization and extinction are the main mechanisms that leads to the establishment of a stable community. This community assembly would be equivalent to the package installation process in a local computer. For example, dependencies and conflicts between packages mimic predator-prey interactions and competitive exclusion relationships, respectively. A predator can colonize the island only if the prey it feeds on is already there.”

In Fortuna’s view, the same thing happens with software packages. “A package can be installed in a computer only if the packages it depends on are already installed. Ecologically similar prey species are going to compete with each other in the island for light and nutrients so that the best competitor is going to displace the others, which can then become extinct. Predators feeding on extinct prey are going to disappear as well.

Conflicts between software packages have the same consequences: one package cannot be installed in the computer if it has a conflict with an already installed one, so that those packages depending on it cannot be installed either. This parallelism can help us understand the general principles operating on systems of different nature."

Reminiscent of AI-based evolutionary programming, Fortuna also says that their work might well lead to improved *in silico* models of evolutionary biology and population ecology. "Charles Ofria and his lab at Michigan State University are studying evolution by using self-replicating computer programs able to mutate and evolve over time." The genome of these programs consists of a set of instructions that are executed by the central processing unit (CPU). Some of the mutations imply the insertion of random instructions into the genome. If the mutant program is able to reproduce faster than the others, its genome is going to persist through time.

"It could be interesting to explore to what extent new instructions added to the genome interact with the preexisting ones – that is, whether or not there is a reuse of the genome instructions of these digital organisms and its resemblance with a modular structural pattern," Fortuna observes. "The interplay between ecology and computer science is much more evident if we take a look at the work developed by Luis Zaman, Ofria's graduate student, who is incorporating host-parasite interactions into these computer programs."

Looking further afield, Fortuna describes how other models or applications might be targeted using the team's findings. "The closest study would be the comparison with the development pattern of other GNU/Linux distributions – openSuse, Fedora, Gentoo, and so on – as well as proprietary operating systems like Microsoft Windows and Apple OS X. The information needed to accomplish this task would easily be compiled for the first ones – but it will be much more difficult to get it

for the last ones. The algorithms for detecting modular structures are publicly available. There are also powerful free SQL relational database management systems like PostgreSQL and MySQL to store, organize, and manage the information. So,' he concludes, "the bottleneck is once again data availability."

More information: *Evolutionof a modular software network*, Publishedonline before print November 21, 2011, *PNAS* December 13, 2011 vol. 108 no. 50 19985-19989, doi: 10.1073/pnas.1115960108