

# Software and Biological Evolvability: A Comparison Using Key Properties

Liguo Yu

*Computer Science and Informatics  
Indiana University South Bend  
South Bend, IN, USA  
ligyu@iusb.edu*

Srini Ramaswamy

*University of Arkansas at Little Rock, USA  
Invited Research Professor, LITIS  
INSA-Rouen, France  
srini@acm.org*

## Abstract

*Biological and software systems share a common property from evolution: they need to change and adapt to either a new environment or a new requirement. If the environment or requirement changes, those systems that have high evolvability will survive and others will be eliminated. The evolvability of a biological system has been widely studied and shown to be dependent on several properties: self-organization, modularity, gene duplication, gene robustness, and symbiosis. This position paper discusses the evolvability of a software system with respect to these properties. Our study shows that software systems share similar evolvability properties with biological systems. We conclude that studying and comparing the internal structures as well as the overall evolution process of these biological systems can help us understand software systems from a holistic 'product-lifecycle' perspective thereby helping us develop software systems with better evolvability traits.*

## 1. Introduction

In biology, evolvability is defined as the ability of a population of organisms to adapt successfully to environmental changes [1]. Evolvability is an important characteristic of a biological system. Only those organisms that can change themselves to adapt to changes in the environment can survive. Others will be eliminated by the process of natural selection.

In software engineering, researchers have adopted the concept of evolvability from biology and have used it to represent the capability of a software system to be changed so that it can continue to serve its customers' new requirement needs [2].

The evolution process and the evolvability of the biological systems have been widely studied. In

contrast, the study of software evolvability has just begun. Although biological and software systems are different in many ways, it is now commonly agreed that both of them need to change to adapt to new environments or new requirement. The ones (either a biological organization or a software system) that have high evolvability can continue to live or to be used while others will be discarded by natural selection or market competition.

The knowledge of evolutionary biology has been adopted by computer scientists and software engineers in some fields. For instance, genetic algorithms [3] that are used in programming to find approximate solutions to optimization and search problems are inspired by the evolution process of a biological system, and are more narrowly focused than what we outline in this paper. In this paper, we analyze software evolvability with respect to the key evolvability properties of a biological system: self-organization, modularity, gene duplication, gene robustness, and symbiosis. This study is inspired by previous related research on comparing evolvability of different systems [4]. The objective is to understand how biological evolvability knowledge (as obtained by a biologist) can be successfully used in the study of software evolvability by software engineering researchers.

## 2. Self-organization

Self-organization is a process that an open system (a system that exchanges matter as well as energy with the surroundings) can return to an organized state without being controlled by other external systems [5]. Examples of self-organization are crystallization in physics driven by energy minimization and mutation in biology driven by natural selection.

For software systems, Lehman's Law III [6] first discussed self-organization as: "*E-type system evolution process is self regulating with distribution of*

*product and process measures close to normal.*” Cook et al. [7] further restated software self-organization as “*E-type system evolution processes are self-organizing, producing distributions of system property values that approximate various well-defined probability distributions, including at least normal, log-normal and exponential.*”

Both Lehman and Cook looked at self-organization from the viewpoint of software process measurement. In this paper, we look at self-organization from the viewpoint of system dynamics. We study the component interactions and associated changes in complex systems (biological, software, economic, and social systems) [8].

It is commonly agreed that self-organization is necessary for an open system evolution. For example, if the living environment is changed, an organism (an open system) must be able to adjust itself to adapt to the new environment. Otherwise, it will not fit to the new environment and will then be eliminated by natural selection.

When considering software systems, we can pose two questions. First, is the software system self-organized? Second, if the software system is self-organized, how is self-organization related to evolvability? It has been claimed by some researchers that software systems are not to be considered as self-organized, because a software system cannot change spontaneously (by itself) to respond to a customer’s new requirement [9] [10]. Hence in this sense, software systems are not evolvable.

However, if we consider the software system together with the programmers (maintainers) that work on it to constitute an open system; the customers and the working environment then, constitute the surroundings. When the customer’s requirement changes, the software can change (with the appropriate implementation by the programmers) to another organized state (form) and can continue to serve the customer. In such a case, we can say that software system is self-organized and evolvable.

Another key issue to note is the fine, yet clear difference between adaptation and evolution. Adaptations refer to minor changes in the requirement / environment that may not necessarily change the structure for exhibiting more efficient dynamic behaviors. Evolution however, often leads to better system dynamics brought forth by a repeated incremental refactoring / re-architecting processes that involves both generalization and specialization behaviors. Examples of such software self-organization traits include refactoring and re-architecting, in which, software systems are changed for a better structure and better state under certain

nonfunctional requirement(s), such as understandability, maintainability, and reusability.

The relation between self-organization and evolvability can then be stated as follows: *if a system has high capability of self-organization, does it also have high evolvability?*

Considering biological systems, the ones that can self-organize quickly and easily to adapt to the new environment are more likely to survive; others that are slow and difficult to adjust promptly to environment changes will be eliminated in natural selection.

Similarly for software systems, ideally those that can adapt quickly and easily to the customer’s new requirement will be continually used; others will lose their market share and will be eliminated from the market. Therefore, the time and effort a system spent on self-organizing could be a measure of evolvability. This matches the definition of software evolvability given by Cook et al. [2]:

*“Evolvability is the capability of a software product to be evolved to continue to serve its customer in a cost effective way.”*

Therefore, self-organization can be thought of as the basis for evolution in any open system, including both biological and software systems. Evolvability of a software system can hence be defined as the *ability of a system to support self-organization such that it can effectively evolve to serve a new requirement without appreciable degradation of, and perhaps much better suited to support its current capabilities.*

### 3. Modularity and hierarchy structure

In biology, modularity is the organization of genotype and phenotype features into reusable components [11]. The *genotype* is the specific DNA makeup of an individual. The *phenotype* of an individual organism is either its total physical appearance or a specific manifestation of a trait, such as size, hair color, and skin color. Generally speaking, genotype, together with the environmental variation determines the phenotype of that individual. Because genotype is more important than environmental variations, we only consider the effect of genotype on phenotype. The relation can be expressed as

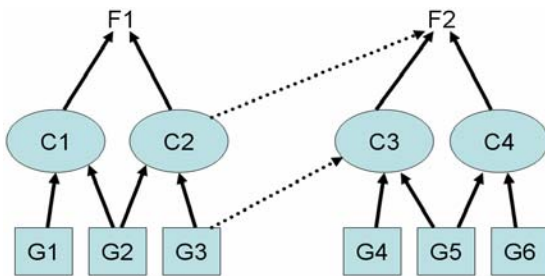
$$\textit{Genotype} \rightarrow \textit{Phenotype} \quad (1)$$

A structure that shows the relations between various genotypes and phenotypes is called genotype-phenotype map. The genotype-phenotype map not only shows the internal relationships of an organism, its structure is also fundamentally important to the process

of a biological system evolution: it determines whether the organism can adapt easily to environment changes by gene mutation and recombination.

Studies show that the genotype-phenotype map can be decomposed into the product of independent smaller dimensions of genotypes and phenotypes [12], which have a modular structure. The modularity structure is a most important property; it can generate the phenotypes that have high fitness for adaptation.

Figure 1 shows the modularity structure of a genotype-phenotype map, in which G represents gene, C represents character, such as morphology (form and structure) of an organ, and F represents function, such as role of an organ. It shows how genotype determines phenotype through modularity (the difference between solid arrows and dashed arrows will be explained in Section 4). For example, function F1 is determined by characters C1 and C2; Characters C1 and C2 are determined by gene G1, G2, and G3 respectively [13].



**Figure 1.** The modular structure of a biological system

Therefore, phenotypes (functions (F) in Figure 1) can be studied on a character by character basis, and characters can be studied on a genotype (genes (G) in Figure 1) by genotype basis. This shows that the biological system is composed of locally integrated units. These units can be considered as modular parts of the organism which integrate together with respect to characters and functions.

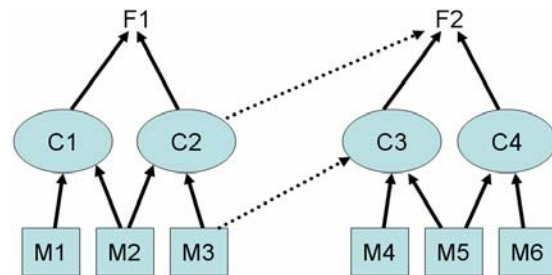
Modularity in the genotype-phenotype map is important for an organism evolution. Consider the example shown in Figure 1. If the living environment is changed and function F1 needs to be changed to adapt to the new environment and because F1 is modularly dependent on genes G1, G2, and G3, only G1, G2, and G3 need to mutate to generate new function F1. Genes G4, G5, and G6 do not need to mutate. Therefore, modularity of the genotype-phenotype map allows small number of genetic changes to adapt to the new environment. This independent mutation of genotypes can reduce harmful

side effects on other phenotypes (functions). Hence modularity enhances the ability of a biological system to generate adaptive variants to adapt to new environment.

Considering software systems, modularity is a widely accepted standard design principle [14]. To achieve high understandability, maintainability, and reusability, software system should be able to be decomposed to manageable components, which can further be decomposed to modules (classes). Figure 2 shows a structure of a software system that is corresponding to the structure of the biological system in Figure 1. In Figure 2, M represents module (class), C software component, and F represents function or service. We can derive another equation that is similar to Equation 1 for software system that shows the relation between modules and functions:

$$\text{Module} \rightarrow \text{Function} \quad (2)$$

As described before, modularity is important for software evolution. Considering the case that the requirement of the software system shown in Figure 2 is changed, we can identify the modules associated with the function and change them accordingly. Effective modularity of a software system thus allows the programmer to easily identify the modules needed to be changed and change them with minimal effort.



**Figure 2.** The modular structure of a software system

However, we note here that in software systems, modules are also interdependent. In this paper, our objective is to look at the evolvability from a biologist point of view. Therefore, other properties of software modularity, such as coupling types, are not discussed here. The reader is referred to our other papers [15] [16] for such issues that explore the relationships between dependencies & maintainability of software systems.

Another similarity between the modular structure of a biological system and a software system is the hierarchical structure as shown in Figure 1 and Figure

2: both systems can be decomposed into several layers. Hierarchical decomposition is an important design abstraction technique to deal with system complexity. The hierarchical layered structure of a biological system allows the reuse of genes and character by different phenotypes [17]. For example, in Figure 1, character C2 is used by both function F1 and F2.

Similarly, in software systems, to cope with the complexity of a very large system, it is not sufficient to just divide the system into simple pieces because the pieces themselves will either be too numerous or too large. A hierarchical modular structure is a solution [18]. Examples are operating systems, network system, database system, that contain different layers with different scales and resolution levels of the associated components.

Therefore, modularity and hierarchy organization which are widely used in software systems also exist in biological systems and experience based observations from software system developers agree with the way biological systems are also structured. Thus, *effective* modularization through hierarchical organization is essential to determining evolvability of both systems.

#### 4. Gene duplication and divergence

A biological system evolves through gene duplication—an extra copy of a gene [19]. Because a gene is under natural selection, many mutations of a gene will lead to the loss of functionality. Once a gene is duplicated, the identical genes can undergo changes and diverge to create two different genes for different functions. Gene duplication is a key mechanism in the evolution of a biological system. Some biologists believe that gene duplication is the most important evolutionary force since the emergence of the universal common ancestor.

In a biological system, modularity alone cannot guarantee high evolvability. It is the combination of modularity and gene duplication that results the system with high evolvability. Again, consider the example shown in Figure 1, and suppose environment is changed and requires function F1 to change to adapt to the new environment. Because F1 is determined by genes G1, G2, and G3, they need to be duplicated and undergo different changes. The combination of different mutated copies of G1, G2, and G3 can generate different functions of F1. If there are no dependencies between G3 and F2, as shown by the dashed arrows, the mutated copies of G1, G2, and G3 that produce F1 that meet the new environment requirement will be selected and others will be eliminated. If there are dependencies between G3 and

F2, the mutation of G1, G2, and G3 may generate new F1 that fits to the new environment. However, this mutation of G3 may also result a new F2 that does not fit to the new environment.

To solve this problem, gene duplication plays an important role. Instead of mutate G1, G2, and G3 directly, several copies of them are duplicated and each undergoes different mutations. The combinations of new G1, G2, and G3 that can satisfy both new requirement of F1 and F2 will be selected. Others will be eliminated.

Therefore, the evolution of a biological system can be represented as a two-step process: (1) environmental change results in gene duplication of old species; (2) natural selection eliminates old species and keeps new species.

Similarly for a software system, modularity alone cannot guarantee high evolvability. As shown in Figure 2, changes made to M3 may satisfy the new requirement for F1. However, it may result in regression fault on F2. Therefore, the relation between G3 and F2 (shown in dashed lines) are potential obstacles to system evolution. One solution for software system is to remove (or minimize) such dependencies between G3 and F2 (shown in dashed lines). However, this may result in low reusability of the system: another copy of C2 and M3 are needed for the implementation of F2.

In a software system, to solve this problem, software programmers usually performs regression testing: to satisfy new requirement of F1, M1, M2, and M3 are changed first; if changes to M3 result errors in F2, M4, M5, and M6 may be changed next.

Therefore, the evolution of a software system can also be represented as a two-step process: (1) requirement change results changes to corresponding modules; (2) ripple changes may be needed on other modules to remove the adverse effect on other functions.

No matter whether it is a biological system or a software system, the existence of multiple relations between genes (modules) and functions complicates the evolution process. Hence more time and effort is required for these systems to evolve.

Another similarity between biological and software systems is the issue of divergence. In biological system, gene duplication can generate various different genes, which is important to produce desired functions. Too little duplication may not be enough for natural selection. Although the evolution of software system is not under natural selection, it is common to see various software solutions that aim to provide similar kinds of functions emerge when the need in a market is

recognized. But, only those software systems that are more competitive than others can survive.

It worth noting that software duplication (copy of modules, components) can also result in software evolution. For example, still consider Figure 2. Suppose function F1 is a driver program for a floppy disk. A new requirement from the user needs a driver program for a CD. To satisfy this requirement, modules M1, M2, and M3 can be duplicated and changed. This duplication is similar to gene duplication in that both of them result in a new function for addressing a new requirement. Although duplicated code is deprecated for software systems, the similar mechanism can happen in both biological system and software system. In object-oriented design of software systems, such copies as referred to as analogous entities. From a software development perspective, such analogous sub-classes can be easily developed if the corresponding relationship with the parent class is through implementation inheritance. It has been reported that most experienced object oriented programmers in industry, in fact, adopt such a technique for quick code development through adaptation across projects; i.e. they prefer implementation inheritance relationship over 'extends' inheritance relationships.

## 5. Gene robustness

In the study of biological systems evolution, it is not uncommon to see that some features of organisms are resistant to change, this is called development constraint. Although the origination of development constraint is still unclear, development constraint is important for a biological system to evolve and keep their intrinsic properties.

One type of development constraint is gene robustness [20] [21] [22]. It refers to the buffering of gene changes against environmental perturbations in the evolution processes. In other words, gene robustness buffers variations in phenotypes by suppressing variations in gene level.

In software systems, gene robustness also finds a corresponding counterpart—product line architectures. In software product lines, there are two kinds of assets, core assets and custom assets. A core asset contains a set of domain specific but application independent components that can be adapted and reused in various related applications. A custom asset contains a set of application specific components. In software product line, core assets, which are designed for reuse within the specific domain, are more stable than custom assets, which are designed for a specific application [23].

To satisfy a customer's new application requirement within the specific domain, we only need to change the custom assets, while leaving the core assets unchanged. Therefore, core assets are more robust to addressing the new requirement.

Other product line similar systems are kernel-based systems, such as operating systems, game systems, and database systems, which contains kernels that are robust to requirement changes, and non-kernels that are more frequently changed. For example, Linux operating system was first designed for Intel 80386 microprocessor. Later its kernel is duplicated and adjusted for more hardware architectures, such as alpha, s390, mips, m68k, and so on [24].

The robustness of core assets in product line and kernels in other systems enables them with high evolvability characteristics within the application domain. It not only reduces the programmer's effort to understand the system, but can also reduce the effort required to make the modification and test the correctness of the modifications.

## 6. Symbiosis

Symbiosis is a close ecological relationship between two different species living together. There are several types of symbiosis depending on whether this relationship has beneficial, harmful, or no effect on these species.

Some biologists [25] [26] consider symbiosis as a major driving force for biological evolution. They think that Darwin's concept of evolution, driven by competition, is incomplete, and claim evolution is strongly based on cooperation and mutual dependence among organisms. According to this theory, species that can evolve in environmental changes are not those that know how to combat but those know how to cooperate.

Generally speaking, software systems are not used independently. For example, an application software system needs to be used together with operating systems, database systems, and network systems, and so on. These systems have the need for symbiotic relationships. Therefore, a property of software evolvability is the ability to cooperate with other systems. Those software systems that can cooperate well with other systems and take advantage of this relationship are more likely to survive in the longer term.

Another example of symbiosis is in distributed systems. Standalone systems have a larger degree of algorithmic complexity but are designed with very little propensity to share information with other such systems. In distributed computing, a distributed

component (agent, service) needs to coordinate with other such components. Components that can take advantage of the capabilities of other components will survive. With respect to incorporating ‘capability’ or intelligence within software systems, this means that a design process must always enable the development of software systems that promote a good communication-centric framework for effective collaboration [27] [28] [29]. For instance, in service-oriented computing, the services with a higher degree of survivability are those with effective information sharing between the components.

## 7. Discussions and future research

In this position paper, we have discussed the evolvability of software system with respect to self-organization, modularity, gene duplication, gene robustness, and symbiosis, which are important properties of biological systems evolvability and identified some design properties that have either been recognized through practice or that need to be achieved for architecting software systems that have evolvability properties. Certainly, the evolvability of a software system is related with many other significant factors that do not exist in a biological system and the time and effort spent on developing such software systems with ‘good’ evolvability traits needs to be adequately justifiable. However, they are beyond the scope of this paper. For example, the distance between two distributed components is an important property of the evolvability of a distributed software system [30]. However, we did not find the corresponding biological property.

Our initial analysis shows that the evolvability of a software system has similarities to the evolvability of a biological system with respect to many properties. Even though our current analysis is not supported by quantitative data, it points out future research directions of using biological analogies to study software evolvability.

One future research efforts will be to focus on the modularity and hierarchical structure of genes in a biological system and modules in the design of a software system. We will study how the genes or modules are correlated to form the gene network or module network and understand how these network structures may affect evolvability.

Another future research topic will be to study the evolution history of the biological system and the software system to see how evolvability takes effect in real world biological system and its relationship to software system evolution.

The paper identifies a critical area for investigation, given that software systems are becoming increasingly complex and interdependent. Studying evolutionary properties such as symbiotic relationships between software systems can help us understand software system evolvability from a very different perspective.

## 8. Acknowledgements

At the time of writing this paper, Dr. Ramaswamy was at INSA-Rouen, France as an invited research professor. He thanks Dr. Mhamed Itmi and his students for several hours of stimulating discussions on related topics. The authors also thank the anonymous reviewers for several interesting and significant observations which improved several aspects of this paper.

## 9. References

- [1] A. Wagner, *Robustness and Evolvability in Living Systems*, Princeton University Press, 2005.
- [2] S. Cook, H. Ji, and R. Harrison, “Software evolution and software evolvability”, Working paper, University of Reading, UK, 2000.
- [3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA, 1989.
- [4] C. L. Nehaniv, “Evolvability in biology, artifacts, and software systems”, *Proceedings of the 7<sup>th</sup> Artificial Life Workshop*, Portland Oregon, 2000, pp. 17–21.
- [5] P. Bak, *How Nature Works: The Science of Self-Organized Criticality*, Copernicus Books, 1996.
- [6] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, “Metrics and laws of software evolution — the nineties view”, *Proceedings of the 4<sup>th</sup> International Symposium on Software Metrics*, CA, 1997, pp. 20–32.
- [7] S. Cook, R. Harrison, and P. Wernick, “A simulation model of self-organising evolvability in software systems”, *Proceedings of the 1<sup>st</sup> IEEE International Workshop on Software Evolvability*, Hungary, 2005, pp. 17–22.
- [8] J. Forrester, *Industrial Dynamics*, Pegasus Communications, 1961.
- [9] M. Conrad, “Towards high evolvability dynamics”, *Evolutionary Systems*, Kluwer Academic Publishers, 1998, pp. 33–43.

- [10] C. Landauer and K. Bellman, "Self managed adaptability with wrappings", *Proceedings of the 1st IEEE International Workshop on Software Evolvability*, Budapest, Hungary, 2005, pp. 29–34.
- [11] H. Lipson, J. Pollack, and N. Suh, "On the origin of modular variation", *Evolution*, 11(No. 8, 2002), pp. 1549–1556.
- [12] L. Altenberg, "Modularity in evolution: some low-level questions", *Modularity: Understanding the Development and Evolution of Complex Natural Systems*, MIT Press, Cambridge, MA, USA, 2004.
- [13] G. P. Wagner, "Homologues, natural kinds and the evolution of modularity", *Integrative and Comparative Biology*, 36 (No. 1, 1996), pp. 36–43.
- [14] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design", *Proceedings of the 8<sup>th</sup> European Software Engineering Conference*, Vienna, Austria, 2001, pp. 99–108.
- [15] L. Yu and S. Ramaswamy, "Component dependency in object-oriented software", Accepted to be published in *Journal of Computer Science and Technology*, 2006.
- [16] L. Yu, K. Chen, and S. Ramaswamy, "Multiple-parameter coupling metrics for layered component-based software", submitted for publication.
- [17] J. Qin, D. P. Lewis, and W. S. Noble, "Kernel hierarchical gene clustering from microarray expression data", *Bioinformatics*, 19 (No. 16, 2003), pp. 2097–2104.
- [18] M. Blume and A. W. Appel, "Hierarchical modularity", *ACM Transactions on Programming Language and System*, 21 (No. 4, 1999), pp. 813–847.
- [19] S. Ohno, *Evolution by Gene Duplication*, Springer-Verlag, 1970.
- [20] B. Hallgrímsson, K. Wilmore, and B. K. H. Hall, "Canalization, developmental stability, and morphological integration in primate limbs", *American Journal of Physical Anthropology*, (S. 45, 2002) pp. 131–158.
- [21] K. E. Willmore, M. L. Zelditch, N. Y. gong, A. Ah-Seng, S. Lozanoff, and B. Hallgrímsson, "Canalization and developmental stability in the Brachyrrhine mouse", *Journal of Anatomy*, 208 (No. 3, 2005), pp. 361–372
- [22] J. Hermisson and G. P. Wagner, "The population genetic theory of hidden variation and genetic robustness", *Genetics*, 168 (December, 2004), pp. 2271–2284.
- [23] L. Yu and S. Ramaswamy, "A configuration management model for software product line", Accepted to be published in *INFOCOMP Journal of Computer Science*, 2006.
- [24] L. Yu and S. Ramaswamy, "Change propagations in the maintenance of kernel-based software with a study on Linux", submitted for publication.
- [25] L. Margulis and D. Sagan, *Microcosmos: Four Billion Years of Evolution from Our Microbial Ancestors*, University of California Press, 1997.
- [26] J. Sapp, *Evolution by Association*, Oxford University Press, 1994.
- [27] S. Ramaswamy, K. Srinivasan, P. K. Rajan, R. MacFadzean, and S. Krishnamurthy, "A distributed agent-based simulation environment for interference detection and resolution", *Simulation*, June, 2001.
- [28] S. Ramaswamy, "Intelligent coordinating entities based control software design", *2000 NSF Design and Manufacturing Grantees Conference*, Vancouver, BC, Canada, January 2000.
- [29] S. Ramaswamy and M. Malarvannan, "Service oriented architectures for Grid computing environments: opportunities and challenges", *International Conference Granular Computing*, Atlanta GA, May 2006.
- [30] C. Luer, D. S. Rosenblum, and A. van der Hoek, "The evolution of software evolvability", *Proceedings of International Workshop on the Principles of Software Evolution*, Vienna, Austria, 2001, pp. 131–134.