Software and Biological Evolution: Some Common Principles, Mechanisms, and a Definition

Davor Svetinovic and Michael Godfrey
University of Waterloo, Waterloo, Canada
{dsvetino, migod}@uwaterloo.ca

Abstract

In this position paper, we explore some of the principles and mechanisms that are shared by both software evolution and biological evolution. Our goal is to highlight some of the commonalities between them, to point out some interesting questions that are raised, and to engender discussions within the software engineering research community on this topic. Our preliminary discussion suggests that new light can be shed on the nature of software evolution itself, and we hypothesize that further study will lead to a better understanding of the various forces that underlie software evolution. We conclude by proposing a new definition for software evolution that reflects this perspective.

1 Introduction

Theodosius Dobzhansky wrote "Nothing in biology makes sense except in the light of evolution" [2], which reflects the level of importance and use of knowledge about evolution within the biological sciences. Although an analogous statement about the importance of evolution in software engineering would probably be too strong, we consider that many of the existing methodologies and techniques in software engineering would be greatly improved by taking into account a large body of software systems evolution knowledge.

Since software evolution is a young science, we wonder what insights we can gain thorough exploration of similarities with more established evolutionary disciplines, and biological evolution in particular. In this paper, we explore some basic principles and mechanisms of biological evolution, and how they might relate to the study of software evolution. The ideas presented in this paper are inspired by Basalla's work on technological evolution [1] and Colby's clarification of the misunderstandings in biological evolution [3].

The purpose of this paper is to demonstrate that there are interesting parallels between software and biological evolution. We hope that this will encourage discussion within the software evolution community, and promote the future study of the influences and links among different evolutionary disciplines.

2 Evolution Versus Change

We begin our discussion with a definition of biological evolution, taken from a popular and well respected website on evolutionary biology [3]:

> *[Biological] evolution is a change in the gene pool of a population over time*.

We note that a *gene* is a hereditary unit that is passed along from one generation to the next, sometimes with changed values, sometimes not.[1] Similarly, the *gene pool* is the set of all gene "values" for a given species or population.

Within biology, not all changes are considered to be evolutionary. What is *not* an evolutionary change? Chris Colby writes [3]:

> *Evolution can occur without morphological change; and morphological change can occur without evolution. Humans are larger now than in the recent past, a result of better diet and medicine. Phenotypic changes, like this, induced solely by changes in environment do not count as evolution because they are not inheritable; in other words, the change is not passed on to the organism's offspring. Phenotype is the morphological, physiological, biochemical, behavioral and other properties exhibited by a living organism. An organism's phenotype is determined by its genes and its environment. Most changes due to environment are fairly subtle, for example size*

[1] More precisely, to use a computer metaphor, one can think of the gene as a memory location on the chromosome, and the *allele* as the abstract value — encoded as a sequence of DNA — that is stored in the gene. Thus, it is the alleles, or sets of abstract DNA values, that are passed along generations, rather than the genes.

differences. Large scale phenotypic changes are obviously due to genetic changes, and therefore are evolution.

How does this carry to software world? What does constitute an evolutionary change in a software system, and what is merely a *phenotypic* change? What is a software phenotype? How ought we to classify changes in software?

To explore possible answers to these questions, we must define what a *software population* is. We believe the most appropriate and logical parallel between population or species, in biological terms, and the ones in software terms, is to consider a species to be analogous to types of software, or software families such as operating systems, compilers, word processors, and so on. This implies in turn that the notion of an individual living being, in the biological sense, is analogous to the notion of a particular product in software world (for example, Linux as an operating system, Microsoft Word as a word processor, and so on). If this were to be true, we might define a *true* evolutionary changes as those that occur at the level of a product family, and affect (more or less) all of its members. We can think of such changes as *essential*, since they are incorporated in all members of the family, while the individuals that do not incorporate them tend to die out. For example, in the case of operating systems the appearance of 32 bit processors (and more recently, 64 bit processors) caused a major evolutionary ripple. Those operating systems that failed to support the new hardware tended to die out, while most of the survivors successfully provided support for the new hardware. Another example is support for virtual memory, which allowed operating systems to permit more memory intensive applications to be active at the same time. The first case is an example of adaptation in direct response to a change in the environment — the hardware on which the OS runs — while the second is an example of an internal change (caused by design rather than random mutation) that gave a competitive advantage to those who successfully adopted it.

By extension, *phenotypic* software changes can be defined as the changes that occur and affect one product through its lifetime, and do not belong to previously mentioned group of changes. This group consists mostly of changes that are less crucial to be incorporated into the product, but give it some competitive edge when compared to the other products from same family. Evolutionary changes consists mostly of changes that must be incorporated into the system in order for the system to stay in widespread use.

Many phenotypic software changes have been studied, either directly or indirectly, as part of studies in software maintenance. These studies give us insight in how a product changes over time, but usually ignore why and how more essential changes appear and affect product families. The results of these studies contribute to the improved development of that and similar products and may improve the processes involved. On the other hand, we believe that detailed studies of evolution in terms of evolutionary changes would, in addition, contribute to the improvement of software business, software technology forecasting, and so on. The difficult question is how do we study and classify such changes, and apply knowledge about them? We will leave this question unanswered for now. Instead, we now explore the different mechanisms that work from the basic principles of *continuity* and *diversity* to cause evolutionary change in the software world.

3 Evolution Mechanisms

The essence of evolution is the incorporation of the *variation*. As such, whatever mechanisms are causing the change, they must either increase or decrease the variation with the population. So, there are mechanisms that (1) *increase variation* in the population, and (2) mechanisms that *decrease variation* in the population.

3.1 Variation Increase

In biological evolution, mechanisms that increase variation are *mutation*, *recombination*, and *gene flow*.

Mutations are caused by mistakes that occur during DNA replication. The gene sequences are altered by these mistakes, giving the new characteristics to the being that carries it. These mutations can have positive and negative effects.

Recombination is basically gene shuffling. It occurs during reproduction, when different combinations of alleles are formed. It occurs both between genes and within a gene. It contributes to the variation since these new alleles and their combinations are introduced into the gene pool.

Gene flow is the mechanism that acts in such a way that genes from one population enter the gene pool of another. This occurs rarely between distant populations. Also, gene flow from one part of population to the other, which were separated for a long time so that changes were incorporated in both, contribute to the variation since it can produce new changes.

If we want to make a parallel between these mechanisms and mechanisms that exist in software evolution, one cannot help but to remark upon the first difficulty: all three include reproduction. On the other hand, all possible mechanisms that introduce variation in the software world are introduced by intentional human actions. So how do they relate?

We take the stand that it does not matter at all. The important fact is that there exist the mechanisms that introduce change and variation in both. Who or what is the reason for this change and variation, and how they get incorporated into the population is not of great importance. The methodology and consequences stay the same as long as the whole

process is the same. Even in the natural world, the reproduction ways are radically different and yet treated the same from the evolutionary point of view.

Some mechanisms that contribute to variation in software technology include *invention*, *environmental forces*, and *product combinations*.

Invention leads to variation. Some of the factors that lead humans to invent are psychological factors, intellectual factors, social and cultural factors, and economic factors. These are the general forces that drive invention [1]. There are numerous cases in software world where this is obvious. For example, the vision of a computer on every desktop, dreams about worlds where robots perform most manual tasks for humans, and so on. Although scientific knowledge was not of great importance for some types of technologies (technology is older than science [1]), in software it plays the primary role. The appearance of software technology is due to advances in scientific knowledge, which is also the factor that makes all improvements and innovations possible. It is therefore important to note the correlation between evolution in science with the evolution of software. Another example, if we study the evolution of computer viruses as a kind of software, we discover different social and cultural issues that have contributed to their appearance and spreading. It is one of the instances where relatively unsophisticated pieces of technology have appeared in socially unacceptable groups and affected the rest of society; viruses have brought out some negative sides of computer technology, provoking revolt in the other social groups. The open source software development community is still another example where social and cultural phenomena have played a major role in the evolution of software systems [4].

Environmental factors are driven by the hardware technology used to support software. There are many ways in which this influences software evolution. For example, the emergence of the emulators as a software family has been directly driven by hardware technology. The difference in an operating system that is ported to a variety of platforms is another example. Also, advances in hardware technology allow for new ideas to be applied; for example, faster processors and cheaper memory have catalyzed research in model checking and intelligent search engines.

Product combination is a force that introduces variation. For example, integration of word processors, spreadsheets, and so on, into the office suites, has invoked the appearance of the software used for these product bindings, collaborations, and synchronization. Also, the evolution of the web browsers to application platforms through open formats, interfaces, and plugin architectures. This is the way that more complex systems are built from less complex ones.

If one wants to make the parallel between biological and software evolution, one could argue about the similarity of mutation to the incorporation of new relatively original features into the product. Recombination could be treated similar to the product combinations and gene flow could be compared to the improvements due to the flow of knowledge or similar.

3.2 Variation Decrease

The second group of evolution mechanisms, in biological evolution, consists of *natural selection* and *genetic drift*.

Natural selection is the difference in reproductive capability. Some populations reproduce and evolve much faster than others. There are different positive and negative factors that contribute to this. What is common to all of them is the requirement for them to act is diversity in the population. Natural selection acts on an individual level; that is, individuals are somehow selected, and their genes propagate into the next generations of the population. There are different kinds of natural selection, such as sexual selection, environmental selection, and so on. The important fact to remark is that natural selection may not lead toward "improvements" in the population; it works in order to improve the population in the short term and not in the long term; that is, natural selection is effectively a greedy algorithm. Since the short-term conditions may be "false" ones (i.e., local greedy optimization or short-lived ones), the population can be badly adapted to what comes in the future. There is also the misconception that natural selection "pushes" the population into some more adapted state, but this is not true since selection operates only on the variation produced by mechanisms that we mentioned earlier.

Genetic drift is the change in the frequency of alleles in the population. Usually these changes are due to chance. Also, fast drops in the population size contribute to the genetic drift.

So, both natural selection and genetic drift decrease the variation in the population. The basic mechanism is evident; some mechanisms introduce variation and others select which variations will propagate and become a representative characteristic of that population. Since most of these variations and selection forces are random, it is not possible to expect that the population is moving towards some utopian ideal. The best chance for a population to improve is if there is an unlimited amount of variation supplied, and mechanisms that act upon that variation are severe and of the same kind that will be present in the future.

In the software world, there are many factors that selectively act upon variation. Some examples are economic factors, military factors, social and cultural factors, environmental forces, and internal forces.

Economic factors also act indirectly as a selection force. There are a number of cases when financial support was of critical importance for the future of some newly discovered technology. Marketing is also one aspect that contributes to

the propagation of some kind of novelty.

Although *military factors* are not usually of great importance for most software technology today, they were the primary ones responsible for its selection at the beginning of the software industry. For example, the Internet itself and many DARPA-sponsored projects fall into this category.

Social and cultural factors are present in almost all selection processes. One example of how the social and cultural factors shape the evolution of technology is the attitude of followers of open source movement toward commercial technologies. The non-acceptance from their side of commercial technologies limits the propagation of those technologies and their evolution in these groups.

Environmental forces perform selection both directly and indirectly. For example, if some software product needs a lot more processing power than is commonly available, it will not be used widely, which will directly affect its ability to spread. In the meantime, another superior technology may appear and when there is enough of processing power, the other one will evolve faster and be accepted more widely than the first one.

As Lehman famously remarked, *internal forces* are rooted in the inherent complexity of software development [5, 6]. These are the forces that can slow down the evolution of a product. Some tasks are very hard to complete, and products that support them are very complex; this complexity slows down the evolution, and products that are more manageable are much more likely to evolve.

4 Conclusion and a Definition

In this position paper, we have performed an early exploration and comparison of software and biological evolution. As we have seen, even this early exploration reveals many similarities from basic principles to underlying mechanisms. This opens up many important questions for future research. Should we proceed in this direction and work on the establishment of a more detailed comparison? Could we transfer some research methods from biological evolution to software evolution? What are the concrete connections between software and biological evolution — how software impacts human evolution? All these and many other questions need answers!

To conclude, we would like to propose a new definition of software evolution that reflects the discussion in this position paper:

> *Software evolution is change in the essential properties of a software product family over time.*

Our next goal is to explore what, exactly, those *essential properties* are.

References

[1] G. Basalla. *The Evolution of Technology*. Cambridge University Press, 1988.

[2] T. Dobzhansky. Nothing in biology makes sense except in the light of evolution. *The American Biology Teacher*, 35:125–129, 1973.

[3] Introduction to evolutionary biology. `http://www.talkorigins.org/faqs/faq-intro-to-biology.html`; accessed April 25, 2005.

[4] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, p. 131, Washington, DC, USA, 2000. IEEE Computer Society.

[5] M. M. Lehman and L. A. Belady, editors. *Program evolution: processes of software change*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[6] M. M. Lehman, D. E. Perry, and J. F. Ramil. Implications of evolution metrics on software maintenance. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, p. 208, Washington, DC, USA, 1998. IEEE Computer Society.