Notes on the Parallels Between Biological and Software Evolution

Ladislav Samuelis
Technical University of Košice
Letná 9, Košice
Slovakia
ladislav.samuelis@tuke.sk

ABSTRACT

This contribution analyzes parallels between software and biological evolution published in software engineering literature. Several papers suggest to the software engineering community that software and biological systems share some useful features of evolution. We compare the driving forces of the software evolution with biological evolution and highlight their different nature. We conclude that searching for parallels between biological and software evolution is a challenging task but at the same time there are limits, which may be hardly observed in the impure research field of software evolution.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Enhancement*; D.2.13 [**Software Engineering**]: Reusable Software—*Reuse models*

Keywords

Software engineering, software evolution, software maintenance, biological evolution

1. INTRODUCTION

Writing this paper was motivated by the author's reading papers in software engineering journals devoted to the parallels between the software and biological evolution, e.g. [22], [15] and [19]. These works suggest that "new light can be shed on the nature of software evolution investigating the parallels between the biological and software evolution". We put some questions: Is it feasible to rely in software development processes on principles of biological evolution? Are there any "parallels" or common "key properties" at all?

The aim of this contribution is to highlight observations concerning the role of biological evolution mechanisms within the context of computer science and software engineering. In particular, it analyzes the abstractions and driving forces of the biological and software evolution. In the conclusion we summarize the possible ways of misunderstandings.

BCI'12, September 16–20, 2012, Novi Sad, Serbia.

Local Proceedings also appeared in ISBN 978-86-7031-200-5, Faculty of Sciences, University of Novi Sad.

2. SOME HISTORICAL NOTES ON SOFTWARE EVOLUTION

The notion of software evolution, which is closely related and often interchanged with the term maintenance was already introduced in the middle of the seventies when Lehman and Belady examined the growth and the evolution of a number of large software systems [13]. They proposed 8 laws, which are often cited in software engineering literature and are considered as the first research results gained by observing the evolution of large software systems.

The term software evolution has emerged in many research papers with roots both in computer science and software engineering disciplines [16]. Nowadays, it has become an accepted research area. In spite of the fact that the science of software evolution is in its infancy, formal theories are being developed and empirical observations are compared to the predicted results.

Lehman's second law states the following: *An evolving system increases its complexity unless work is done to reduce it.* Due to the consequences of this law and due to the increased computing power, research in software and related areas accelerates and very often causes confusion and inconsistency in the used terminology.

Research on software evolution is discussed in many software related disciplines. Topics of software evolution are subjects of many conferences and workshops, too. In the following paragraph we will briefly characterize the interpretation of the notion of evolution in the short history of software engineering.

The notions of program synthesis or automated program construction are the first forerunners of the evolution abstraction in software engineering. Papers devoted to these topics can be found also in the research field of automated program synthesis. Practical results achieved in the field of programming by examples are summarized, for example in book [7]. The general principle of these approaches are based on the *inductive inference* principle [1].

The term evolution was originally a synonym for the automation of program construction and for the discovery of the reusable code – i.e. searching for loops. Later on, when programming technologies matured and program libraries and components were established in practice, the research field of pattern reuse [8] and engineering component-based systems [2] drew attention of the theory and practice to component engineering. In other words, a slight shift to component-based aspect is observed in the course of program construction. We may say that the widely used term of customization was stressed and later on this term also

merged with the notion of evolution. Of course, this shift was heavily supported by the object-oriented programming languages, which penetrated into the industrial practice in the 1980s.

Since it was a necessity to maintain large and more complex legacy systems, the topic of program comprehension came into focus and became more and more important. Program comprehension is an activity drafted, for example, in the paper of Rajlich and Wilde [18] as: *Program comprehension is an essential part of software evolution and software maintenance: software that is not comprehended cannot be changed. The fields of software documentation, visualization, program design, and so forth, are driven by the need for program comprehension. Program comprehension also provides motivation for program analysis, refactoring, reengineering, and other processes.*

This is in compliance with the idea that our understanding of the domain problem incrementally evolves and learning is an indispensable part of program comprehension. Rajlich in dealing with the changing paradigm of software engineering stresses the importance of the concept location. He argues that the volatility of the requirements is the result of the developer's learning. Thus learning and understanding or comprehension is indispensably coupled with the concept of evolution [17].

These ideas are the forerunners of domain engineering as described by Bjorner in his work [3]. We add that mental activities associated with understanding are dealt within the cognitive sciences and it is important to realize that, for example, not all software design concepts (intangibilities in the mind) can be formalized.

The scattered results from the above mentioned areas lead to the attempt to establish the taxonomy of software evolution [5]. Further areas of the contemporary research, which deal more or less with the evolution concept, are software merging [14], measurement of the flexibility and complexity of the software [6], and software visualization for reverse engineering [12].

It is also obvious that the evolution principle in the biological interpretation heavily attracted and influenced the research of the evolution in software engineering. The work of Nehaniv et al. critically warns the software engineering community about the non-obvious traps when the evolution principles valid in biology are mechanically applied to the area of software artifacts [15].

These short notes highlighted the interlacing of software related disciplines and how they mutually influence each other.

3. DOES SOFTWARE DEVELOPMENT BENEFIT FROM THE MECHANISMS OF BIOLOGICAL EVOLUTION?

Our position is that abstractions and mechanisms of biological evolution have limited impact on our understanding of software evolution. It is a challenge to question and clear the abstractions derived from the parallels in the impure field of software evolution. Questioning parallels between the biological and software systems reveals the limits of their applicability.

Let us cite from the work of Svetinovic and Godfrey [21]:

> [...] 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.

In fact this approach neglects at all the importance of the change mechanisms. That is why we cannot discuss about the differences or similarities between them.

The next sentence says:

> Who or what is the reason for this change and variation, and how they get incorporated into the population is not of great importance. [...]

If we want to analyze the parallels between the biological and software evolution then it is of great importance to characterize them. First of all, we have to distinguish between the processes that model the domain and processes that select the appropriate model for the domain. The selection of the appropriate model (or software development) is a non-deterministic process and that is why it cannot be fully automated, too.

Let us put the question: How does genetic algorithms relate to software evolution?

If we accept the standpoint that software models processes of a certain domain (natural or artificial) by algorithms (not only) then those algorithms may be inspired (among others) also by natural evolution (see, for example, the Wikipedia definition of the genetic algorithm). The answer is clear, genetic algorithms belong to the world of algorithms (the abstractions applied to a certain domain may be valid or not).

If we accept the standpoint that changes "are introduced by intentional human actions" (as stated by the authors in the same paper) then we also have to admit that the changes are non-deterministic (it is also possible to select other algorithms than genetic for modeling a particular problem).

The conclusion is that we have to be aware and not to confuse non-deterministic human activities with the deterministic processes of the natural evolution.

The following sentence of the authors says:

> The methodology and consequences stay the same as long as the whole process is the same.

Accepting the fact that the processes are not the same intrinsically (software development is non-deterministic process – based on creative *thinking and understanding*) then we have to avoid temptation (very comfortable) to write about "some common principles, mechanisms between the software and biological evolution".

Otherwise we come to the conclusion that software development is a deterministic process (some natural process), which is contradiction with our initial standpoint (it is non-deterministic).

To sum up, we have to differentiate strictly between the processes that we model and the thinking-understanding (this was not yet modeled). These processes belong to two "distinct worlds". If we superficially "merge" these two worlds (intentionally or unintentionally) then we deliberately give up seeking the truth and may construct false theories.

The same work states also:

> 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 origin features into the product. Recombination could be treated similar to the product combination and gene flow could be compared to the improvements due to the flow of knowledge similar.

We note that it is easy to find common abstractions on the observational level and, for example, to draw an analogy between the biological mutation and the incorporation of new features into the product. At the same time we have to declare that here we arrived to our boundaries. We have to fix that *gene flow* is deterministic but *flow of knowledge* is unknown and non-deterministic.

In other words; software is modified by actions of human beings, while biological evolution is not the result of actions by human beings. Biological evolution is forced by "some natural" forces. Processes that drive the organism's genotypes and phenotypes [9] are deterministic and cannot be compared non-deterministic flow of knowledge applied to the man-made artifacts.

This idea is expressed from another aspect by Jazayeri [11]:

> [...] Not the software itself evolves, but our understanding and the comprehension of the reality. [...]

This approach stresses the importance of program comprehension and the tools that support comprehension (see also the work of Jackson [10] on tools for supporting understanding).

In summary, software evolution evolves according to our understanding of the domain, which we want to model by the computer. From this point of view knowledge about biological evolution does not support our understanding of the domain under consideration (or universe of discourse).

Considering a software's life cycle phases, we may observe (and have to observe) in all steps (as domain engineering, requirements engineering, specification, design, implementation, test, and operation) learning activities, which in fact means the *evolution of our understanding* of the modeled phenomenon.

Much more effective, than to seek parallels between biological and software evolution is to follow the ideas in works of Bjorner [4]. He introduces the notion of domain engineering and his triptych dogma says:

> [...] Software cannot be designed without a robust understanding of the requirements and the requirements cannot be prescribed without a robust understanding of the domain. [...]

As far as we know, the only way of achieving a robust understanding of the domain is attentive work in the domain in order to gain experiences by trial and error. This is also called *inductive inference* and the product is our intangible understanding of the idea to be modelled. Following this thread of thought; the result of inductive inference is a *deductive system* (software or model), which has to be mapped on the computer.

In fact, it is not possible to build a software system without thorough understanding of the domain (which we do anyhow through all phases of software development life cycle). The more deeply we understand the reality the more the software could be evolved.

We must note that the *Software Engineering Method and Theory* initiative [20] does not mention parallels between biological and software evolution at all.

4. CONCLUSION

Summing up, the existing disciplines of software engineering approach the phenomenon of evolution from various aspects. These approaches have their own history and theoretical roots; they are in various branches of computer/computing science and are treated from the philosophical point of view, too. It can be guaranteed that new techniques and research areas will emerge in the near future and will further deal with the phenomenon of evolution.

We hope that the above-discussed ideas will focus the attention of software engineers toward the reconsideration of the obviously claimed statements, which are against the facts observed in reality. In other words, to provoke a discussion about the feasibility of biological and software evolution parallels in software development.

5. ACKNOWLEDGMENTS

This work was supported by the project KEGA 040TUKE-4/2011: Modern Software Engineering in Education – Proposal of the Structure and Realization of Actual Software Engineering Subjects for Informatics Study Program at Technical Universities.

6. REFERENCES

[1] http://en.wikipedia.org/wiki/Inductive_inference.

[2] F. Allen. The history of language processor technology in ibm. *IBM Journal of Research and Development*, 25(2):535–548, 1981.

[3] D. Bjorner. Domain engineering: Technology management, research and engineering,. http://www2.imm.dtu.dk/ db/jaistmono.pdf.

[4] D. Bjorner. From domains to requirements. methodology contributions to domain analysis and requirements engineering. http://www2.imm.dtu.dk/ db/from-domains-to-requirements.pdf,.

[5] J. Buckley. Towards a taxonomy of software change. *Journal of Software Maintenance*, 17(5):309–332, 2005.

[6] A. Eden. Measuring software exibility. *IEE Software*, 153(3):113–126, 2006.

[7] A. C. et al. *Watch What I Do: Programming by Demonstration*, chapter Chapter "Programming by demonstration". Cambridge University Press, 1993.

[8] M. Fowler. *Analysis Patterns: Reusable Object Models*, volume Object Technology Series. The Addison-Wesley, 2000.

[9] http://en.wikipedia.org/wiki/Genotype-phenotype_distinction.

[10] M. Jackson. *Automated Software Engineering*, volume 15, pages 275–281. Kluwer Academic Publishers Hingham, Dec. 2008. ISSN: 0928-8910.

[11] M. Jazayeri. Species evolve, individuals age. In *In International Workshop on Principles of Software Evolution*. ACM, Sept. 5-6 2005.

[12] R. Koschke. Software visualization, state-of-the-art survey. In *Software Visualization for Reverse*

Engineering, volume 2269, pages 113–126. Springer LNCS, 2002.

[13] M. Lehman and L. Belady. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

[14] T. Mens. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 28(5):449–462, 2002.

[15] C. Nehaniv, J. Hewitt, B. Christianson, and P. Wernick. What software evolution and biological evolution don't have in common. In *SOFTWARE-EVOLVABILITY '06: Proceedings of the Second International IEEE Workshop on Software Evolvability*, pages 58–65, Washington, DC, USA, 2006. IEEE Computer Society.

[16] L. J. Osterweil. What is software? *Automated Software Engineering*, 15(3):261–273, 2008.

[17] V. Rajlich. Changing the paradigm of software engineering. *Commun. ACM*, 49(8):67–70, 2006.

[18] V. Rajlich and N. Wilde. The role of concepts in program comprehension. In *Proceedings of the IEEE International Workshop on Program Comprehension*, pages 271–278. IEEE Computer Society Press, 2002.

[19] A. Sampaio. Software phenetics, phylogeny and evolution. In *Software Evolvability, 2007 Third International IEEE Workshop on*, pages 60 –66, oct. 2007.

[20] http://www.semat.org.

[21] D. Svetinovic and M. Godfrey. Software and biological evolution: Some common principles, mechanism, and a definition. In *Proceedings of the 8th IWPSE*, Lisbon, 2005.

[22] L. Yu and S. Ramaswamy. Software and biological evolvability: A comparison using key properties. In *Software Evolvability, 2006. SE '06. Second International IEEE Workshop on*, pages 82 –88, sept. 2006.