The Application of Artificial Evolution in Software Engineering

W.M. De Jong and H. Degens

Dr. William M. DeJong .
INI-Research
Delfgauwseweg 299
2628ES Delft, The Netherlands
Phone: +31.15.2850958
dejong@ini-research.nl

Dr. Ir. Hans Degens
Institute for Biomedical Research
into Human Movement and Health
Manchester Metropolitan University
John Dalton Building; Chester Street
Manchester M1 5GD
Phone: +44.161.247.5686
h.degens@mmu.ac.uk

The Application of Artificial Evolution in Software Engineering

ABSTRACT

We review a project of software-development companies to reduce the cost of software engineering by artificial evolution. We evaluate the extensive computer simulations of the evolution of populations of self-replicating 'digital amoebae' that were done from the perspective of computer science, change theory and evolutionary biology. Although the project revealed that the main mechanisms of evolutionary biology can not expand digital codes into new dimensions and have no software engineering potential, it advances our fundamental understanding of how digital and DNA codes can change and adapt. It appears that the explanatory mechanism of evolutionary biology is in fact a composite of two independent essentially different sub-mechanisms: 1) variation of the DNA code, in particular by the process of recombination of alleles, and selection of beneficial variations and 2) mutation of the DNA code and selection. We outline new directions for research and practical application following from this finding.

1. INTRODUCTION

In 2003 several software-development companies set up a joint venture to investigate the potential of the main mechanisms of biological evolution to produce large and complex digital program codes. The project addressed three questions: 1) Can the mechanisms of evolution be applied to reduce the cost of software engineering? 2) How do digital and DNA codes compare in their response to the evolutionary principles of variation, mutation, recombination and selection? and 3) Can the innovation of software and software development profit from these findings? These questions were addressed in two ways: A) computerized simulation of the transformation of a self reproducing 'digital amoeba' into a more complex 'digital organism' by evolutionary principles, and B) literature study of evolutionary computation and evolutionary biology [1,2].

We will first present and evaluate the results of the evolution of a simple self-replicating computer program subjected to random variation, mutation and selection. Then we will proceed to investigate the change of digital programs at a more fundamental level, applying the dichotomy of change revealed by change theorists: first-order and second-order change, where the former takes place without altering the current basic assumptions or framework in contrast to the latter [3]. Equipped with the mathematical formalization of both kinds of change as the movement of a state vector within and beyond its initial system space, respectively, we review the literature on artificial evolutionary change to see whether both first- and second-order change occurs. Next we compare how digital and DNA codes respond to random change processes. We conclude that the explanatory mechanism of evolutionary biology is in fact a composite of two independent and fundamentally different processes: 1) variation of the DNA code, in particular by the recombination of alleles followed by selection of beneficial variations and 2) mutation of the DNA code followed by selection. Based on the

empirical differences between both processes, we present an operational definition of 'DNA variation' and 'DNA mutation'. The implication is that when studying and discussing biological or artificial evolution, the sub-mechanisms of variation and mutation must be disentangled. This distinction results in a more accurate conceptualization of evolution and its empirical effects, and opens new directions for advancing evolutionary theory and its application.

2. THE SOFTWARE ENGINEERING POTENTIAL OF THE EVOLUTIONARY MECHANISMS

To investigate the software engineering potential of the evolutionary mechanism of variation, mutation and selection, a 'digital amoeba' – called Damoeb – was constructed. A Damoeb is a small (3.3 Kbytes) C++ program that imports two digits from an input file, processes them into another digit and exports it to an output file. What action will be performed depends on the value of a control parameter in the program code of the Damoeb, which has the value 1, 2, 3 or 4, regulating the activation of the corresponding operator: '+' for summation, '-' for subtraction, '/' for division, or 'x' for multiplication, respectively. Also a replication and random variation (RRV) program was constructed for making a copy of a Damoeb and assigning at random a value 1, 2, 3 or 4 to the control parameter of the copy Damoeb, resulting into an α-type Damoeb (+), a β-type Damoeb (-), a γ-type Damoeb (/), or a δ-type Damoeb (x), respectively. The random assigning process was constructed in such a way that the control parameter of the copy Damoeb got the value of the control parameter of the original Damoeb with a 94% chance or one of the three alternative values of the control parameter with a 2% chance each. During a replication time interval τ, a Damoeb would be

allowed to enter the RRV-program once. After an existence of 5 τ, a Damoeb would be deleted.

2.1 Random variation and selection

The first stage of simulation was started with one α-, one β-, one γ- and one δ-type Damoeb. All Damoebs were fed with the digit pair (20, 5) and allowed to replicate freely until the population consisted of about 1000 Damoebs, which were equally distributed over each type. Subsequently, selection rule S1, demanding the output digit produced by a Damoeb to have a value between 0 and 20, was imposed on the population. Only if a Damoeb met the selection criterion it was allowed to replicate. As expected the share of β- and γ-type Damoebs in the population grew strongly at the expense of the α- and δ-types. The α- and δ-types, however, did not become extinct because the RRV-program allows them to arise sporadically from the replication of β- and γ-type Damoebs. After about 6 replication cycles of random variation and selection, the distribution of Damoeb-types adapted to the imposed selection rule and reached a new dynamic equilibrium. When selection rule S1 was replace by rule S2, demanding the output to be greater than 50, the population moved to a distribution with mainly δ-type Damoebs and a very small share of α-, β-, and γ-type Damoebs. When S2 was replaced by selection rule S3, demanding the output to be between 0 and 10 or between 20 and 50, the α- and γ-type Damoebs met the environmental constraint and started to dominate the population (Fig.1).

Subsequent simulations were refined by limiting the presence of digital food, varying its composition (i.e. varying the digit pairs fed into the model), varying the lifetime of the Damoebs and addition of more control parameters for processing the input digits and producing output. In all steps of refinement, the same adaptive behaviour of the population

was observed making the Damoeb simulations an excellent model of the adaptive behaviour of for instance populations of bacteria or finches to environmental changes.

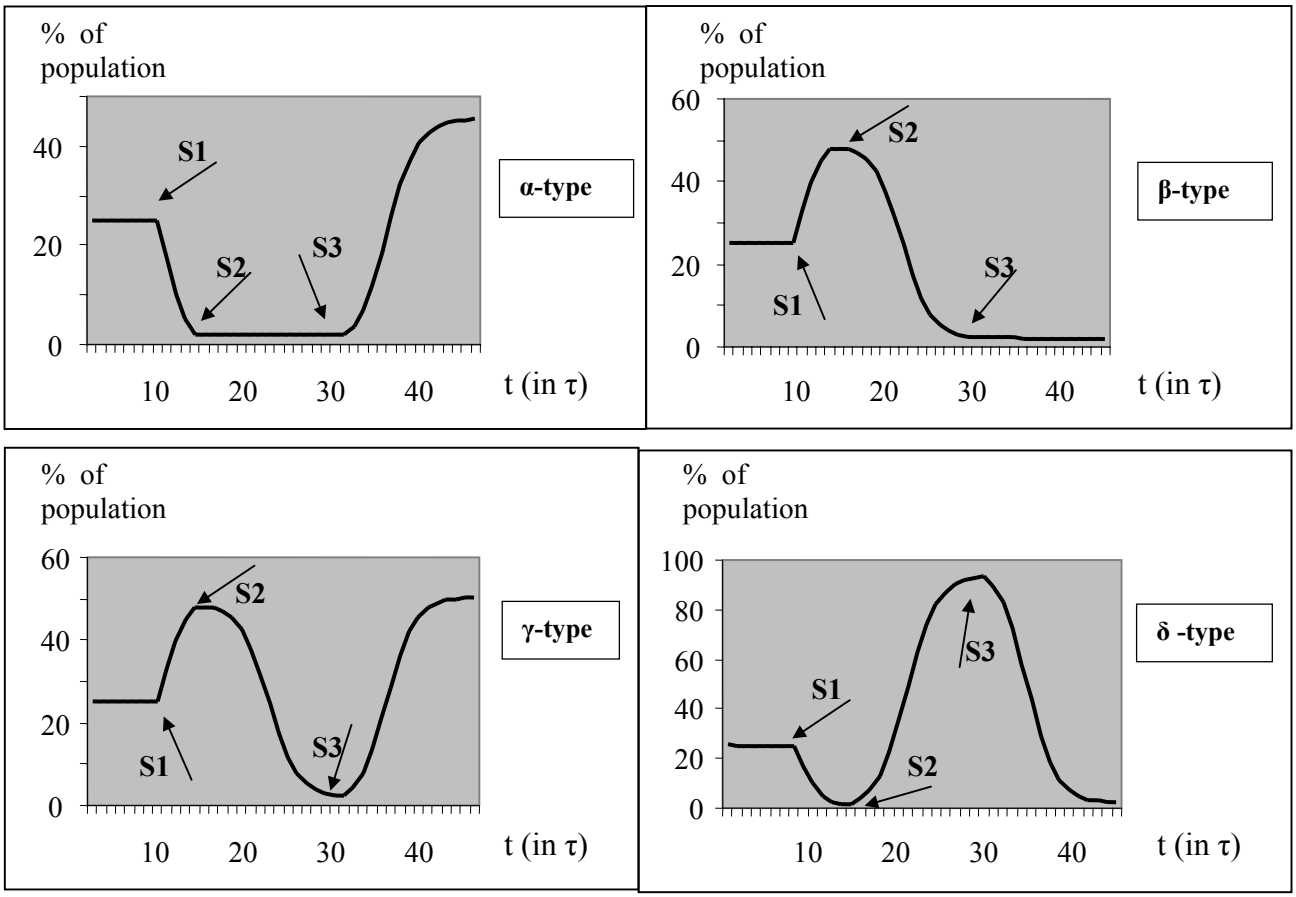

Figure 1: Adaptation of a Damoeb population to selection rules S1, S2 and S3 imposed respectively at 9 τ , 15 τ, and 29 τ, where τ is the replication time of a Damoeb. For explanation of the selection rules see text.

2.2 Random mutation and selection

Since random variation of the control parameter of the Damoeb program only led to effective adaptation of the Damoeb population to environmental changes but not to an increase of the program code, a mutation module was developed. While in digital codes information is represented by binary digits, which can take two discrete states denoted as '0' and '1'

respectively, in DNA information is represented by four specific molecules, which are denoted by A (adenine), C (cytosine), G (guanine) and T (thymine), respectively. A digital code thus consists of a string of 0's and 1's and a DNA code consists of a string of A's, C's, G's and T's. Because mutation of the DNA code happens at the level of the nucleotides, for instance by natural decay of the nucleotides and errors during replication or restoration of DNA damage, simulation of DNA mutation demands mutation of the string of 0's and 1's of a digital code. To do this, we constructed a mutation module that changes the 0's and 1's of the digital code of the Damoebs at random. It should be noted, that similar to DNA, there is a system for mutation protection in digital codes. In each byte – a set of eight bits – seven bits are used to represent information. The seven-bit string 1000001, for instance, codes the letter 'A' and the seven-bit string 0110011 codes the number '3' according to the ASCII-code which is the defacto world-wide standard for the representation of digital information [4]. The 8^{th} bit of each byte – called 'parity bit' – is used for mutation detection. If the number of 1's in the seven-bit string is even, the 8^{th} bit is given the value 1, otherwise it is given the value 0. If one of the bits in the seven-bit string mutates, the number of 1's changes and the value of the parity bit no longer corresponds with the actual seven-bit string resulting in the detection of the mutation, which can then be removed or repaired. Despite the presence of mutation-protection in standard digital codes, the mutation module was constructed and added to the RRV-program.

Firstly, a population of about 1000 Damoebs was generated. Subsequently, each Damoeb copy generated by the RRV-program was put through the mutation module, changing at random 10 bits of the Damoeb's digital code (total size of the code 3.3 Kbytes = 26,400 bits) by turning a '1' into a '0' and a '0' into a '1'. Subsequently the growth of the functionality of the Damoeb was assessed. If the output file expanded by an extra number or character, reflecting the origin of an additional function in the program, the improved Damoeb

would be selected and allowed to replicate, whereas the mutated Damoebs that had not improved were excluded from replication. It was found, however, that after putting a Damoeb through the mutation program, about 95% of the Damoebs ceased to function and produced error messages referring to unreadability of the program or to spelling and syntax errors related of the C++ program language, while the still functioning Damoebs did not show improved functionality. In the next simulation the selection rule that only Damoebs with an improved functionality were allowed to replicate was dropped, giving not yet visible new functionalities a chance to develop further. Thus, all Damoebs that were still functioning after passing the mutation program were allowed to replicate, but even in this case the population became extinct after about twenty replication cycles. In subsequent simulations, the number of mutated bits was decreased to 5 and to 1, and subsequently increased to 100 and finally to 1000. These large mutations simulate the occurrence of a long sequence of small mutations that have no effect and build up into a large mutation that ultimately affects the organism. Yet, the same results were found. In order to be absolutely sure of the effects of random non code-expanding mutation of the Damoebs, each of these experiments was repeated automatically a million times and without exception the result was a rapid extinction of the population (Fig. 2).

Since random mutation of the existing code of the Damoebs did not produce an increase of its functionality but invariably led to extinction, the mutation module was adapted to simulate code-expanding mutations of the DNA by making a copy of a 10-bit string from the digital code of a Damoeb at random and inserting it at random elsewhere into the digital code. Again, the experiment started with a population of 1000 Damoebs. After passing the RRV-program, each Damoeb copy was put through the refined mutation module. It was found that about 97% of the mutated Damoebs ceased to function and produced error messages referring to unreadability of the program or to spelling and syntax errors in the C++

program language, while the still functioning Damoebs did not show any improved functionality. In the next series of simulation, the same approach of dropping the selection rule that only Damoebs with an improved functionality were allowed to replicate was followed to give not yet visible new functionalities a chance to develop further, but again only extinction of the population after as little as twenty replication cycles was found. In subsequent simulations, the experiment was repeated when the length of the at random inserted random bit-string was decreased to 5 and to 1, and subsequently increased to 100 and finally to 1000. The same results were found, also with the large mutations that simulate the occurrence of a long sequence of small mutations that have no effect and build up into a large mutation that affects the organism. Again, the simulations were repeated automatically a million times, and again extinction rather than expansion of Damoebs with new functionalities occurred (Fig. 2).

The Damoeb simulations make clear that random variation of the control parameters of a digital program and selection of beneficial variations is an effective approach to adapt to changing environmental constraints. Random variation of the control parameters, however, does not result in growth or increased functionality of the program. The Damoeb simulations also make clear that random mutation of the 0's and 1's of a digital code or the random addition of 0's and 1's to a digital code result in error messages related to damage of the bytes of the program or to spelling or syntax errors at a higher level of the code, and to abortion of the program. The Damoeb simulations thus did not reveal any software engineering potential in the applied evolutionary principles to expand a digital program with new functionalities.

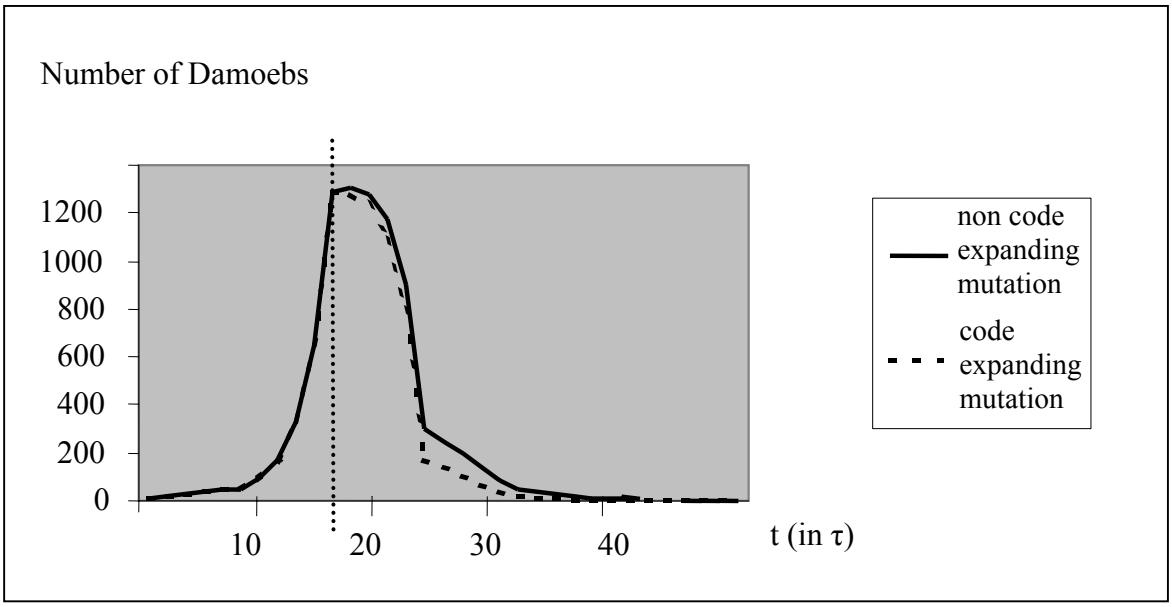

Figure 2: Extinction of a Damoeb population by mutation of the program code, starting at t = 16τ, where τ is the replication time of a Damoeb. For further explanation see text.

3. ASSESSMENT OF THE LITERATURE ON COMPUTERIZED SIMULATION OF EVOLUTION

That the Damoeb simulations using the main mechanisms of evolution did not expand a digital program with new functionalities came as a surprise to the financial supporters of the project. Indeed, our results seem to be in contrast with extensive literature on computer simulations of evolution that indicate that the evolutionary mechanisms have software engineering potential and can expand a simple program into a complex one. In the following section we will discuss these and our reports in more detail, applying the insights of change theory into systems change.

3.1 Dichotomy of change

According to the theory of evolution all changes are essentially equal and differ only in degree. According to change theory, however, a dichotomy of change exists: first-order change versus second-order change [3]. The essential difference between both types of change is revealed when the state of a system is measured along a set of dimensions and represented as a state vector. Then, the change of a system can be represented as the movement of the state vector within or beyond the system space spanned by the dimensions of the state vector at the start of the change process – the 'initial system space'. During first-order change, the state vector only changes within its parameters and keeps moving within its initial system space. During second-order change, the state vector expands into one or more new dimensions and moves beyond its initial system space. This dichotomy of change can be demonstrated by changing a rectangular flat object R. The state of R can be measured along the dimensions 'length' and 'width' and be represented as the state vector (x,y), where x gives the length and y the width of R at a certain moment of time. When the length or the width of R change, the parameters of its state vector change. As a result, the state vector moves within its initial system space spanned by the dimensions 'length' and 'width', and first-order change is present. When R changes into a beam B, the state vector (x,y) expands into a new dimension resulting in a state vector (x,y,z), where z represents the depth of B at a certain moment of time. As a result, the state vector of R moves beyond its initial system space and second-order change has occurred.

3.2 Adaptation of digital programs

In computer science, ways to adapt programs to changing demands or environments is a major field of research. Most programs can be adapted by varying their control parameters. This often takes the shape of changing numbers in an input file of the program according to predefined specifications and within predefined boundaries. A control parameter can also refer to an operator or program module that can be chosen from a given set of operators or program modules, as illustrated by the Damoeb simulations we presented above where the one control parameter of the program was varied at random within the number set {1, 2, 3, 4} activating four different operators – '+', '-', '/', or 'x – providing effective adaptation of the Damoeb population to changing demands. Sophisticated programs – for instance, Windows and Word – possess hundreds of features, each of which can be varied by clicking on a desired feature option in a predefined input list of possible options, activating the corresponding program module. If the working environment changes, different feature options can be selected, often by trial and error, until a recombination of existing variation is reached that fits the demands of the new working environment. When varying the control parameters or the selection of program modules the source code of the program does not change, the state vector of the program keeps moving inside its initial system space, and only first-order change is present.

3.3 Evolutionary programming

The selection of the optimal value of a set of control parameters or feature options can be done by trial and error if the number of possible variations is relatively small. If the performance of a system or program depends on a large number of control parameters or feature options, a structured incremental search method based on random variation and selection can be applied to find the optimal combination. This search method is denoted as

evolutionary computation (EC), evolutionary programming, or genetic computation [5]. In a well-known example of EC, a program produces a string of 28 letters, which function as control parameters. The program offers the possibility to replace each letter with another letter from the alphabet, resulting in 26 possible variations for each control parameter. Sub-sequences of letters in the string of 28 letters that do resemble English words are excluded from further variation. Using this approach, a random string of 28 letters can be transformed into readable English text after a limited number of cycles [6]. It is important to realise, however, that the source code of the program that produces the string of letters, varies the letters, and fixes readable parts, does not change. In terms of change theory, the vector that represents a certain letter combination keeps moving within its initial system space, which is defined by the fixed source code of the searching program. The program thus produces only first-order change as it searches for a position of the state vector within its initial system space that satisfies a predefined criterion.

In a more sophisticated form of EC the control parameters of a program do not consist of numbers or letters, but of program modules that can be picked at random from a predefined list of predefined modules. A well-known example of this type of EC uses program modules that can expand a given line by varying the angle and line length of branching. A string of such program modules will produce a relatively simple tree-like drawing. These tree-like drawings in turn can be subjected to incremental steps of variation and selection until a tree-like drawing is produced that looks, for instance, like an insect [6]. Similarly, if the program modules describe the construction of molecular 'building blocks', to be picked from a given set of modules, large molecular strings can be produced at random, which in turn can be subjected to a process of variation and selection, ultimately resulting in the synthesis of new chemicals or medicines [7]. In the modelling of molecular building blocks, three-dimensional information can be incorporated that describes the folding characteristics of molecules. This

results in the expansion of the set of possible variations from which random choices can be made [8]. In these advanced applications of EC, effective variation is essential. Strings of control parameters that result in a high score on the selection criteria are often recombined. This then is followed by fixing some parts of the resulting string and varying the other parts between certain boundaries. These strategies for producing new variations by recombination of beneficial variations of the control parameters or feature options are often denoted as 'recombination and mutation' [9]. This terminology, however, is misleading as the source code of the program that defines the search strategy and produces and selects random tree-like drawings, or random strings of molecular building blocks, does not undergo mutation; only the control parameters or feature options of the program are varied within prescribed limits or domains according to programmed rules. As a consequence, the state vector of the program keeps moving within its initial system space as defined by the initial, and unchanged, source code of the searching program. Thus also in these examples only first-order change occurs.

In terms of change theory artificial evolution can therefore be described as a field of incremental, computerized, search strategies to find an optimal position or route of a state vector within its initial system space. The search process is automated by a computer program that assesses the value of each position or route that is produced and provides ways for selecting more beneficial positions or routes. Often, artificial evolution can be visualized as the search of an optimal route through a predefined labyrinth by random choice of operations from a predefined set of possible operations and selection of beneficial choices. For example, in the simulations of evolution performed by Yedid and Bell [10] in the so called AVIDA environment [11], where a fixed set of predefined low-level computer instructions are combined at random resulting in independent 'software-robots' that can compete with one another for run time, a string of 80 predefined computer instructions can move a computer processor from a predefined initial state to a predefined end state. By random recombination

of these instructions alternative routes to the end state are sought. In the simulations, strings of instructions that consume little processor time receive competitive advantage and the original route taking 80 instructions can be reduced into several shorter alternative routes that take about 30 instructions only. Again, during this optimization process, only first-order change is present. This applies to all other simulations with the AVIDA environment. In the AVIDA environment, second-order change can only be achieved when the predefined set of processor instructions is expanded by a new type of instruction and the processor in use is replaced by a processor that is able to understand and execute this new instruction. Such a change cannot be achieved by random mutation of the 0's and 1's of the source code of the AVIDA simulation program, but requires the vision, skills and effort of software engineers.

4. COMPARISON OF DIGITAL AND DNA CODES

As discussed above we obtained no evidence from our simulations or from the literature on artificial evolution that random change of a digital codes can result in the development of new functionalities and expansion into new dimensions. According to evolutionary biology, however, DNA codes behave differently when random change occurs. Therefore, we continued to address the following two questions: 1) how do digital and DNA codes compare as to their response to the evolutionary mechanism of variation, mutation and selection?, and 2) how can the innovation of software and software development profit from these findings?

4.1 Variation and selection

In terms of computer science, the DNA of a cell can be seen as a program that defines what proteins can be expressed in the cell. The partitions of DNA that code the proteins can be

viewed as program modules. In a population of sexually reproductive organisms, these program modules – ‘genes’ – are present in several variations – 'alleles'. When organisms reproduce sexually, the alleles of the parent organisms are mixed into new combinations that are passed down to the offspring, resulting in random variation of the DNA code, which can be subjected to selection. Within living memory, farmers have exploited this mechanism both in animal and plant breeding, resulting in the minuscule Chihuahua, the huge Irish Wolfhound, the child-friendly Welsh pony, the fierce Arabian horse, milk or meat cows, juicy or fleshy oranges, single or multi-blossom tulips, summer or winter wheat, etc. Recombination of alleles is not restricted to organisms that possess pairs of chromosomes. Bacteria also are capable of exchanging genetic material, resulting in new allele combinations, which can result, for instance, in the development of resistance against antibiotics [12].

The mechanism of variation of alleles and selection of beneficial recombinations is identical to the mechanism of variation of already existing programme modules in digital programs and selection of beneficial recombinations. This process is illustrated by the first stage of the presented Damoeb simulations. Both during variation and selection of alleles in organisms and the variation of feature options and selection in digital organisms, or programs like Word or Windows, the nucleotides or digits involved remain unchanged. There are, however, some differences between DNA and digital codes in adapting to their environment by recombination of yet existing variation. In digital codes, each feature option has a specific discrete effect on the behaviour of the program, while in DNA codes an allele can cover a broad spectrum of possible effects, depending for instance on its molecular folding, its epigenetic molecular structure, or its chemical environment. The latter, in turn, is influenced by the activity of other genes. This provides significant additional variation, enhancing the adaptability of the DNA code to altered circumstances. Since also the epigenetic structure of

alleles is inheritable [13], beneficial variations of the epigenetic structure can be passed on to future generations, allowing a population to benefit from the related selective advantage for a longer period of time. These possibilities for additional variation – and thus for first-order change – are potentially interesting for the innovation of digital software.

4.2 Mutation, mutation repair and mutation elimination

All physical carriers of digital codes - for instance magnetic tapes or optical disks - are subject to natural decay. Although digital carriers are rather resistant to random mutation, after 10 to 50 years most digital codes are corrupt. Therefore, digital codes are protected against mutations, as described above, and in heavy-duty environments even mutation repair is implemented. When a mutation is detected, the processing of the code is aborted and the program reverts to a still working back up. The higher the demand of the working environment the more back ups are installed [14]. Since mutation repair is costly and difficult to organize and maintain it is not normally implemented in every-day digital data processing. In the experiments with random mutation of the digital code of Damoebs, no provisions for mutation repair were made. As a consequence, the normal mutation detection mechanisms in C++ programs simply produced error messages followed by the abortion of the programs, ultimately leading to the extinction of the mutated Damoeb population.

In comparison to digital codes DNA is highly unstable. For instance, radiation, chemical modification and temperature all affect the integrity of the DNA molecule. Indeed, the DNA in every human cell loses about 5000 adenine or guanine bases per day by depurination and about 100 cytosine bases per day by deamination [13]. Clearly, without the repair of these mutations the 3 billion characters of the human DNA code would turn into utter chaos within a lifetime. Fortunately, the cell contains a large repertoire of DNA repair

enzymes [15]. The repair of DNA mutations is based on the presence of multiple copies of the code. In diploid organisms at least four copies of the code are present (since each chromosome consists of two chromatides and each chromatide consists of two complementary DNA strands). In addition to this, DNA is continuously checked for mutations. The mutation detection, repair and elimination occur at several levels: 1) at the level of a single DNA strand using the sister DNA strand as a template, 2) at the level of the chromatide using the sister chromatide as a template, and 3) at the level of the chromosomes using the homologous sister chromosome as a template [16]. The latter mechanism represents a line of intra-cellular defence against code-expanding mutations by comparing the chromosomes from the father with that from the mother [17]. In comparison with digital codes, the mutation detection and repair of the DNA code is at a much higher level and is of potential interest for the development of fault-resistant digital codes.

Although according to evolutionary theory DNA mutation is a source for improvement of the DNA, expansion with new functionalities and selective advantage, research reveals that random mutation of DNA leads to malfunction, cancer and hereditary diseases, thus causing a reduction, rather than an increase, in fitness of an organism [18, 19, 20]. Consequently, stringent safety protocols are used in industry, laboratories, hospitals, etc. to keep the exposure to mutagenic radiation and chemicals of the DNA to a minimum. The effects of mutation and natural decay of DNA correspond with the effects of the mutation and natural decay of digital codes as encountered in every-day data storage and processing, and are confirmed by the computer simulations of the random mutation of the code of digital amoebae we discussed above.

5. DISCUSSION

We have reviewed a research project that sought to apply the evolutionary principles of variation, mutation, recombination and selection to reduce the cost of software engineering. We found that random variation of the control parameters or feature options of a digital program is an effective strategy to adapt it to continuously changing environmental constraints. The dynamic behaviour of a population of self-replicating digital amoebae – Damoebs – where random variation of four feature options and selection was implemented, appeared to be an accurate description of evolutionary processes, mimicking observations in populations of Darwin finches [21]. Random mutation of the 0's and 1's of the digital code or random addition of 0's and 1's, however, rather than resulting in Damoebs with new functionalities led to their extinction. Since we found no software engineering potential with our simulations applying variation, mutation, and selection, we evaluated the published reports of artificial evolution that did seem to confirm evolutionary theory. We found that evolutionary programming or evolutionary computation (EC), which varies the program parameters or program modules of a computer program at random followed by selection of beneficial variations, can be viewed as an incremental, computerized, search strategy for an optimal position or an optimal route of a state vector in a predefined system space. Therefore, EC only produces first-order change. The literature does not provide reports of computerized simulation of evolution where random processes expand a digital program into new dimensions, resulting in second-order change and software engineering potential. We subsequently compared digital and DNA codes to determine differences that could explain why random mutation of digital codes produces errors, while random mutation of DNA codes, according to the theory of evolution, results in growth of the code and expansion into new dimensions. We did not find such differences, but found that digital and DNA codes are

highly comparable and are both protected against mutations. We found that the absence of mutation repair in a population of Damoebs results into its extinction, and that dysfunction of mutation repair in organisms results in degeneration of DNA, hereditary diseases and cancer, and hence a reduction rather than an improvement in fitness of the organism. Yet, evolutionary theory suggests that dysfunction of mutation repair is necessary to expand DNA codes into new dimensions and provides selective advantage. This contradiction demands further investigation.

5.1 Refinement of the conceptualization of the evolutionary mechanism

Based on our simulations and the discussion above we conclude that random variation of a DNA code, in particular by recombination of alleles and selection of beneficial variations, is a completely different process than the random mutation of the DNA. Table 1 summarizes these differences. From these differences, we derive the following definitions: "DNA variation = the non-antagonized change of the DNA" and "DNA mutation = the antagonized change of the DNA". DNA variation and DNA mutation can thus be distinguished from one another in an operational way.

Table 1**:** Disentangling the evolutionary mechanism into two sub-mechanisms

	Sub-mechanism 1: DNA variation	*Sub-mechanism 2: DNA mutation*
Causes	- recombination of alleles - variation of the readability and impact of alleles by changes in: molecular folding, epigenetic structure, chemical environment	- molecular decay of nucleotides by radiation, heat, or chemicals - unrepaired copy errors
Empirical effects	- continuous adaptation of the DNA code of an organism to changing demands - continuous adaptation of a population to environmental changes	- dysfunctioning - heriditary diseases - cancer
Antagonizing processes	- none	- intra-cellular mutation detection, aborting and repair mechanisms based on the redundancy of the DNA code - intra-cellular code expanding mutations aborting mechanisms - extra-cellular mutation aborting mechanisms
Size of the genome	- unchanged	- unchanged, in case of point mutations or translating copy errors - unchanged or decreasing, in case of the natural decay of the DNA

The distinction of ‘DNA variation’ and ‘DNA mutation’ enhances our insight into evolutionary change, helps to conduct valid simulations of evolutionary change, and opens new directions for future research and application.

5.2 Directions for future research

Computer science usually fulfils the role of servant of evolutionary biology. The finding from our simulations that the explanatory mechanism of evolutionary biology is a composite of two different and fully independent sub-mechanisms positions computer science as an equal partner in the research of evolutionary change and the advancement of evolutionary theory. From that position, we make some suggestions for future research. A first direction is the assessment to what extent the change of organisms or populations over time is due to DNA variation or DNA mutation. The changes of the DNA as a result of recombination of alleles and subsequent selection of beneficial recombinations are not antagonized by the DNA repair machinery but rather actively promoted, and a clear example of DNA variation. The non-antagonized alterations in the readability or impact of alleles resulting from changes in molecular folding, epigenetic structure, and/or chemical environment, are also examples of DNA variation. On the other hand, changes of the DNA code as a result of failing mutation detection, abortion and repair mechanisms, in particular non-repaired reading, writing, and transportation errors when copying the DNA, are heavily antagonized and clear examples of DNA mutation. At the interface between antagonized and non-antagonized changes in DNA, new research questions emerge as: a) Is it possible to disentangle all observations of biological change into changes induced by DNA variation and those induced by DNA mutation?; b) Which of the observed changes in living nature reveal a non-antagonized expansion of the DNA-code that is maintained in the gene pool of the population?; c) Can

dysfunctioning mutation repair in a population produce selective advantage?; and d) How can the antagonized and non-antagonized change of the DNA code be modeled by computer simulations? Research into these questions will advance our understanding of antagonized and non-antagonized changes of the DNA and the processes that affect these mechanisms. Ultimately this understanding may produce new techniques for the prevention of and fight against hereditary diseases and cancer.

A second direction for future research follows from the similarities between DNA and digital codes and relates to the partitions of the DNA that do not code for proteins. Any computer program consists of both information describing 'what' has to be done or produced and information that describes 'how' the process of doing or producing has to proceed. Analogously, it is likely that the non-protein coding part of DNA contains the process information of the DNA code. Recent research has yet identified switches for activating or deactivating genes and clockworks for setting the time when processes have to start or stop in the non-protein coding parts of DNA [22, 23]. From the point of view of computer science interesting research questions are: a) What is de code of the process information in the DNA?; b) Do 'process-genes' play a role, are they present in the gene pool of a species in the form of 'process-alleles', do they recombine during meiosis, and are these recombinations subject to selection processes?; c) How do variation and mutation play a role?; and d) How can the findings on the DNA process information be represented by computer simulations?

5.3 Applications

Although the research project we reviewed did neither reveal software engineering potential of the mechanisms of evolution nor ways to reduce the cost of software engineering, it has supplied the companies that took part in the project with valuable new ideas for the innovation

of software and software engineering. A first area for innovative application is the development of computer games. Both digital and DNA codes can adapt to changing demands by recombination of already existing variation followed by selection. This mechanism is applied widely in computer games by allowing a player to choose hundreds of different features out of sets of variations for each feature, such as the characters that play a role, how they move and look like, their attributes, and the characteristics of their environment. In digital codes, however, each feature option has a specific discrete effect on the behaviour of the program. As a result, the look and feel of a specific situation in the game is always the same and having encountered a certain situation many times players get bored. In DNA codes, in contrast, an allele covers a broad spectrum of possible effects, dependent on, for instance, its molecular folding, epigenetic molecular structure, its chemical environment which in turn is influenced by the activity of other genes. This provides massive additional variation, enhancing the adaptability of DNA codes to changing circumstances. In the development of computer games, this property of DNA codes can be used as a model for expanding the variety of the feature options of computer games, by adding control parameters which are linked to random variation and selection processes and processes of inheriting variations through time.

A second area for application is the dynamic mutation repair of digital codes. Normally, digital codes are protected against mutation and degeneration by aborting a program when the parity bit in a byte changes and switching to a copy of the program code elsewhere. The provisions in organisms to antagonize the mutation and degeneration of the DNA code appear much more refined, dynamic and extensive, and provide many ideas for advancing the mutation repair of digital codes. These ideas may, for instance, lead to the replacement of bytes for digital coding of information by ‘duplex-bytes’ in which a byte and its complement are coupled. Subsequently, mutation of the code can be detected and

dynamically repaired without aborting the program. At this interface between evolutionary biology and computer science, major innovations in software engineering are likely, advancing the reliability of software in heavy-duty environments such as space travelling, where mutation of computer software by high radiation fluctuations is a major problem.

A third area for innovative application of the insights that follow from our research is the field of smart drugs development. The ongoing discoveries of how process information is coded in the non-protein coding parts of the DNA, which boost the development of drugs that target the non-protein coding parts of the DNA [24], can be used for the development of new low level program languages that are operationalised by molecular strings. In the future, such strings may be attached to drugs, triggering them to become active in the right circumstances at the right time.

5.4 Concluding remarks

Computer science usually has the role of 'servant' of evolutionary biology. In this paper, computer science takes the position of an equal partner of evolutionary biology in the research and discussion of evolutionary change. It provides evolutionary biology and the field of artificial evolution with new insights that advance our fundamental understanding of evolutionary processes. Our research reveals that the evolutionary principles of variation, mutation, recombination and selection cannot expand a digital code into new dimensions, and hence are inadequate to result in second-order change. All computer simulations of evolution reported in the literature, including the sophisticated simulations made in the so-called AVIDA environment, produce only first-order change, possess no software engineering potential, and do not confirm that the explanatory mechanism of evolutionary theory is sufficient to transform a simple organism into a complex one. The evolutionary mechanism

appears to be a composite of two fully independent sub-mechanisms: 1) variation of DNA, in particular by recombination of alleles from the gene pool of a species followed by selection of beneficial variations, and 2) mutation of DNA and subsequent selection. We have presented an operational definition for both sub-mechanisms. The disentangling of DNA variation and mutation reveals that DNA variation produces effective adaptation of organisms and populations to environmental constraints, while DNA mutation causes dysfunctioning, cancer, and selective disadvantage. In addition, the disentangling advances the conduct of artificial evolution by computerized simulations and opens new directions for research and application at the interface of evolutionary biology, computer science, oncology and pharmacy, confirming that evolutionary biology – as the study of biological change – is strategic science [25].

REFERENCES

[1} INI-Consult 2003. *Evolutionary programming and cost reduction. Research proposal.* INI-Consult, Delft, The Netherlands.

[2} INI-Consult. 2005. *Evolutionary programming and cost reduction. Evaluation report*. INI-Consult, Delft, The Netherlands.

[3] P. Watzlawick, J. H. Weakland, and R. Fisch. 1974. *Change: Principles of problem formation and problem resolution*. Norton, New York.

[4] Linux Information Project. 2004. "ASCII: A Brief Introduction." At: http://www.bellevuelinux.org/ascii.html

[5] J.A. Foster. 2001. "Evolutionary computation." *Nature Genetics*, 2 (June): 428-436.

[6] R. Dawkins. 1991. *The Blind Watchmaker*. Penguin Group, London.

[7] G.B. Fogel and D.W. Corne. 2003. *Evolutionary Computation in Bioinformatics*. Elsevier Science, San Fransisco, CA.

[8] I. Belda, X. Llorà, M. Marinell, T. Tarrago, and E. Giralt. 2004. *Computer-Aided Peptide Evolution for Virtual Drug Design. Genetic and evolutionary computation*. GECCO. Springer, Berlin.

[9] J.R. Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press. Cambridge, MA.

[10] G. Yedid and G. Bell 2002. "Macroevolution simulated with autonomously replicating computer programs." *Nature,* 420: 810-812.

[11] C. Adami and C.T. Brown. 1994. "Evolutionary Learning in the 2D Artificial Life Systems Avida", pp. 377-381 in: R. Brooks, P. Maes (Eds.), *Proc. Artificial Life IV*. MIT Press, Cambridge, MA.

[12] P. Awadalla. 2003. "The evolutionary genomics of pathogen recombination." *Nature Genetics*, 4 (January): 50 - 59.

[13] B. Alberts, et al. 2002. *Molecular biology of the cell*, 4th edition. Garland Science, New York.

[14] A.D. Singh, and S. Murugesan 1990. "Fault-Tolerant Systems."*IEEE Computer*, 7: 15-18

[15] R.D. Wood, M. Mitchell and T. Lindahl. 2005. "Human DNA repair genes." *Mutation Research*, 4 (Sep), 577(1-2):275-83.

[16] I.C. Friedberg, G.C. Walker and W. Siede. 1995. *DNA repair and mutagenesis*. American Society of Microbiology Press, Washington DC.

[17] J.W. Kimball. 2003. "Crossing Over and Genetic Recombination in Meiosis", at: http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/C/CrossingOver.html

[18] V. Beral, G. Reeves, e.a.. 1992. "Childhood Thyroid Cancer in Belarus." *Nature*, 359: 680-681.

[19] American Institute for Cancer Research. 1997. *Food, Nutrition, and the Prevention of Cancer: A Global Perspective*. Author. Washington DC.

[20] A. Leroi. 2005. *Mutants: On Genetic Variety and the Human Body*. Penguin Group, New York.

[21] H.L. Gibbs and P.R. Grant. 1987. "Oscillating selection on Darwin's finches." *Nature* 327: 511-513.

[22] U. Schibler, U., J.A. Ripperger, and S.A. Brown. 2001. "Circadian rhythms. Chronobiology – reducing time." *Science* 293: 437–438.

[23] M.R. Bennett and J. Hasty. 2007. "A DNA methylation–based switch generates bistable gene expression." *Nature Genetics* 39: 146 – 147.

[24] Y. St-Pierre. 2007. "Drug discovery using the regulation of gene expression." *Expert Opinion on Drug Discovery,* 2, 7: 987-1000

[25] Th.R. Meagher. 2007. “Is Evolutionary Biology Strategic Science?” *Evolution*, Jan.: 239 - 244