The Software Market as A Complex Evolving System

Ben Goertzel
University of Nevada, Las Vegas

1. Introduction
Markets are complex evolving systems. One consequence of this is that a market left to itself does not necessarily provide “freedom” for the actors who compete within it. Quite the contrary: each market self-organizes itself into a certain structural/dynamic pattern, and this pattern places definite constraints on those who wish to compete.

A perfect example of this is the software market. Much more blatantly than markets for more conventional goods, the software market displays all the typical features of a complex, self-organizing system. Of course, every complex system has certain characteristics all its own. But my goal here is to argue that a great many aspects of the software market may be understood as manifestations of general system-theoretic principles.

Several theorists (Goodwin, 1990) have recently pointed out the relevance of chaos theory to economic analysis. However, chaos in the technical sense is just deterministic pseudo-randomness. What I am talking about here is something related but (as I see it) much more interesting: not the emergence of pseudo-randomness from simple dynamics, but the emergence of intricate, creative structures and processes from simple dynamics.

Self-Organizing Optimization
What sort of “general system-theoretic principles” are we talking about? Let us begin at the beginning. As I see it, what binds economics, psychology, physics and biology together is a common reliance on the concept of optimization. Economics studies entities acting to maximize profit. Evolutionary biology studies entities acting to maximize fitness — and so does immunology. Psychology studies entities acting to maximize pleasure, or power, or reward. Physics studies systems seeking to achieve minimal, or stationary, action.

The crucial question is, how deep does this parallel run? For sake of concreteness, let us narrow our question down to the parallel between economics, brain/minds, immune systems and ecosystems. These are all optimizing systems par excellence.

In biological parlance, what we want to know is: is this parallel between these four systems an homology or merely an analogy? Are these four systems similar in the shallow way that birds’ wings and bats’ wings are similar? Or are they similar in the deeper way that bats’ wings and rats’ legs are similar?

Birds’ wings and bats’ wings are similar in form only; the processes underlying their construction and maintenance are quite different. The connection between them is merely an analogy. On the other hand, bats’ wings and rats’ legs are fundamentally quite similar, despite their superficial differences. They are related by a genuine homology — a similarity of underlying structure.

As you may have surmised, I believe that the parallel in question is more than just an analogy. Not only are economies, brain/minds, immune systems and ecosystems all optimizing systems, but the different systems all share a significant body of optimization processes. In particular, I believe that at least four specific optimization strategies are common to economies, ecosystems and brain/minds (though not necessarily immune systems, which are inherently simpler):

1) Self-referential synthesis of processes (Goertzel, 1993b)
2) Evolution by natural selection (Goertzel, 1993a)
3) “Sexual reproduction” operators (Goertzel, 1993a)
4) Multilevel perception and control (Goertzel, 1993, 1993a)

This list is a short one, but it is too long for a single paper. So, let us focus on items (1), (2) and (3), leaving the final one for later consideration.

These are weighty issues. But, luckily, our goal here is a relatively lightweight one. We seek merely to describe processes (1), (2) and (3), and to argue that they all play a role in the development of the computer software industry. We will show, by means of simple examples, that these three purportedly universal strategies of complexity are in fact present in this particular complex system.

Our objective is not to make dramatic new predictions about the next big-selling program, but rather to illustrate that the software market obeys the same rules as other complex systems. Therefore,
the emphasis will be on the three processes rather than the software industry. However, while abstract theoretical analysis is at one remove from day-to-day marketing decisions, it may well be useful in prognosticating the long-term future of the software industry. This issue will be briefly addressed at the end of the paper.

**General Systems Theory**

Before we get started, a brief methodological note is in order. Many theorists have expressed skepticism as to the existence of general “laws” of complex system behavior. But, while skepticism is an admirable quality, it should not be used to block off reasonable lines of scientific investigation. On an abstract level, complex systems are confronted with similar problems: problems of recognizing and forming patterns. The claim of general systems theory is that these problems possess certain solutions which are not only optimal, but far, far superior to any other solutions. The four processes listed above are, in my opinion, excellent candidates for the role of “drastically superior solution” to problems of pattern recognition and formation.

It is crucial to be absolutely clear on this point: there is as yet no proof that the pattern recognition and formation problems confronting complex systems possess certain solutions which are vastly superior to all others. This is a highly plausible mathematical conjecture, and a useful working hypothesis.

I have spoken about “general systems theory,” but the word “general” should not be overinterpreted. For example, the immune system uses evolution by natural selection, and a sort of self-referential form creation, but only a weak version of multilevelization, and no creative crossover. This is, I suggest, because the problems with which it is confronted are relatively simple. It must recognize and create patterns, but only easy patterns — patterns of bumps on molecules. A system which had to deal with subtler patterns, could not make do without more of the strategies of complexity. Economic systems, I suggest, are more akin to ecosystems or brains than to immune systems. They deal with subtler types of patterns, and thus must use the more sophisticated strategies.

My choice of the software market as an example is not accidental. Just as some organisms are more complex than others, so are some markets more complex than others. The corn market, I would suspect, displays rather little of the five strategies of complexity outlined above. It is a much simpler entity. The software market is replete with self-organization and logical subtleties.

**2. Evolutionary Economics**

Let us begin with process (1) — evolution by natural selection. The original “evolutionary economics” was, of course, Herbert Spencer’s (1878) theory of Social Darwinism. According to Spencer, just as the best organisms will tend to win the fight for survival, so will the best individuals tend to win the fight for socioeconomic success. But of course, what many of Spencer’s contemporaries suspected, we now know: this crude version of natural selection explains neither biological nor economic change.

Modern biologists have realized that there is no context-independent way to characterize one organism as “better” or “fitter” than another. The word “fit” must rather be interpreted as in the phrase “this shoe fits me better than that one.” One organism is fitter than another, with respect to a given environment, if it matches the environment better. The modern interpretation of the theory of evolution by natural selection, therefore, states that survival is roughly proportional to “match” with the environment.

Since the environment of each organism consists largely of other organisms, this leads to a system of equations. And it also leads to a complex dynamics. The survival of each species is roughly proportional to its match with other species, so if one species should change by chance, then the others may be encouraged to change along with it.

This is an exciting development — good biology and good philosophy. But there is one important caveat: for all this to be meaningful, it must be possible to determine which of two organisms better matches its environment, by some method other than observing which one survives more effectively. In The Evolving Mind (Goertzel, 1993a), I have used ideas from theoretical computer science to formulate a generally applicable algorithm for determining the degree of match between an organism and its environment. However, this algorithm is formidable difficulty to apply in particular cases; so for the analysis of specific problems, one must avail oneself of classical mathematical optimization principles, or plain old specialized intuition.

**Structural Fitness**

To explain how theoretical computer science leads one towards a general definition of the “match” between an entity and its environment, we must begin with what Gregory Bateson called “the Metapattern.” This basic epistemological/
ontological axiom, central to the philosophy of Charles S. Pierce and the anthropological linguistics of Benjamin Whorf, states very simply that the living world is made of pattern. In The Structure of Intelligence and The Evolving Mind this idea is developed into a rigorous theory, based on the definition of a pattern as a representation as something simpler.

More explicitly: a process $w$ is a pattern in an entity $x$ if

1) the result of $w$ is a good approximation of $x$

2) $w$ is simpler than $x$

The degree to which $w$ is a pattern in $x$ — a technical quantity which we will need to speak about only occasionally — is the product $AB$, where $A$ is the ratio of the simplicity of $w$ to the simplicity of $x$, and $B$ is one plus the error of the approximation of $x$ by $w$.

This begs two question: what is a “result”? and what is “simpler”? The easiest way of specifying these concepts is to take a computational point of view. For example, if an “entity” is a binary sequence, and a “process” is a computer program, then the simplicity of an entity or a process may be defined as its length. This ties in nicely with Gregory Chaitin’s (1987) algorithmic information theory: the algorithmic information $I(x)$ is the simplicity of the simplest pattern in $x$. A simple example is the binary sequence $x = 101010101010101010101010101010101010101010101010101$. The program $w = "\text{Repeat} \ '100' \ 21 \text{ times}\)” is a pattern in $x$, because it results in $x$, and is shorter.

Next, having defined “pattern,” the concept of structure follows immediately. I define the structure of an entity as the set of all patterns in that entity. This is a fuzzy set, since different patterns provide different degrees of simplification.

Formally, I denote the structure of $x$ as $St(x)$. One may define the structural complexity of an entity $x$ as the “size” of the fuzzy set $St(x)$. This is a measure of the “total amount of pattern” in $x$: it captures formally the sense in which a person is more complex than a flower, which is more complex than a virus. But the definition of “size” here is a little bit subtle — one has to subtract out for overlapping patterns — and there is no need to go into it here.

And finally, having talked about patterns and fuzzy sets of patterns, we must introduce one operation relating patterns with fuzzy sets of patterns. This is the operation of emergence. Most simply, a process is emergent between $x$ and $y$ if it is a pattern in the “union” of $x$ and $y$ but not in either $x$ or $y$ individually. More generally, a process is emergent between $x$ and $y$ if the degree to which it is a pattern in the union of $x$ and $y$ exceeds the sum of the degree to which it is a pattern in $x$ and the degree to which it is a pattern in $y$. The set of all patterns emergent between $x$ and $y$ is denoted $Em(x,y)$, and it is defined by the equation $Em(x,y) = St(x U y) - St(x) - St(y)$.

Now we may return to the problem of defining fitness. I define the structural fitness of an organism $O$ as the size of the set $Em(O,E)$, where $E$ is the environment of the organism. If there are patterns in the union of the organism with its environment, which are not patterns in the organism or the environment individually, then the structural fitness is large. Perhaps the easiest illustration is camouflage — there the appearance of the organism resembles the appearance of the environment, so that $Em(O,E)$ contains a repetition, the simplest kind of pattern. But symbiosis is an even more convincing example. The functions of two symbiotic organisms match each other so effectively that it is easy to predict the nature of either one from the nature of the other.

My favorite illustration of structural fitness is Robert MacArthur’s classic study of bird feeding habits. MacArthur showed how five different species of warbler avoid competition while feeding on bud worms in the same spruce tree: each species spends more than half of its time in a certain specific region of the tree. A consequence of this is that, given the feeding habits of any four of the species, one can compute the feeding habits of the remaining species. This is an emergent pattern between each species and its environment — it is a computational simplification that appears only when one considers the species and its environment (consisting of the tree and the other species) as a whole.

Under the pattern-theoretic view, then, the extent to which an environment evolves by natural selection is the extent to which the survival rate of organisms within it is correlated with their structural fitness. In this sense, it is clear that ecosystems evolve by natural selection to a fairly high degree. Modern immunology (de Boer and Perelson, 1991) implies that immune systems do also. According to (Goertzel. 1993: 1993a), so do associative memory networks. What I am claiming here is that so do entities in the socioeconomic realm.

According to the approach outlined here, what it means to say that socioeconomic systems evolve by natural selection is the following:

All else equal, given two socioeconomic entities, the one which generates the most
emergent pattern in conjunction with the entities that surround it will tend to survive longer.

This is a general maxim: it may be applied on the level of systems of government, on the level of firms, or — as is our main interest here — on the level of individual products.

Note that this is not exclusionary definition of fitness. Just because structural fitness is correlated with survival, this does not imply that there is not some other type of fitness which also correlates with survival. At present there does not seem to be any other general, objective way of quantifying the notion of “best match” — but this observation says little about what the future may bring.

Fitness of Governmental Systems

The confusion between fitness-as-absolute-superiority and fitness-as-best-match has been most acute in evolutionary biology. But it has also played a role, albeit a subtler one, in economic thinking. For instance, in these post-Gorbachev years, one often hears the claim that American-style democratic capitalism has proved itself to be the best form of government. But even if one concedes that capitalist democracies have fared better than nations subscribing to other forms of government, this does not imply that capitalist democracies are superior in any absolute sense. It means only that they have adapted best to their particular environment — to the particular mix of developed and underdeveloped nations, and the specific mix of cultures, that characterizes our world.

In the same vein, many political writers (both leftist and rightist) have argued that the failure of Russian communism was ultimately due to the existence of the capitalist Western nations. What these writers are saying is that communism is fit relative to an environment of other communist nations, but unfit relative to an environment of capitalist nations. In terms of our structural definition of fitness, this argument is certainly not a priori implausible.

Fitness of Computer Software

In the context of computer software, things are even more clear-cut. The program which succeeds on the market is not necessarily better in any absolute sense. It is merely fitter in the sense of generating the most emergent pattern in combination with the other software on the market. For instance, the graphic art program CorelDraw, which is based around interpolation with Bezier curves, is designed to be compatible with the Windows 3.0 operating system. If tomorrow I wrote a graphic design program with twice as many useful features as CorelDraw, it still might not be as fit as CorelDraw, unless it were also compatible with Windows 3.0.

The reason is that, when using CorelDraw in conjunction with Windows, one has certain capabilities that are not present in CorelDraw alone, nor in Windows alone. One has certain emergent capabilities. For instance, CorelDraw cannot export bitmaps — a serious drawback for the graphic artist. But because CorelDraw is run through Windows, one can save a CorelDraw screen in the Windows Paintbrush program, and then use the features of Windows Paintbrush to save the screen into a bitmap file. In other words, the combined entity (CorelDraw, Windows) has the capability to let the user draw a picture using Bezier curves, and save it in bitmap form. But this capability is possessed by neither CorelDraw nor Windows on its own.

The experienced computer user will be able to list five or ten similar examples from her own experience — programs that are valuable because of the way they interact with other programs, rather than due to any “intrinsic” qualities. By merely studying the code of two programs, one cannot tell which program will be more useful. One has to know what sort of computing environment the programs will be used in.

Conclusion

We have used the theory of pattern to provide a general, mathematically and conceptually rigorous idea of the concept that economies evolve. The ideas which we have derived fit in nicely with the more philosophical approach to evolutionary economics taken by Kenneth Boulding (1981). As Boulding so rightly points out, equilibrium theory was inadequate from the start; ever since its conception, its primary virtue has been the absence of a well-articulated alternative. Evolutionary and general-systems-theoretic economics promises to provide such an alternative.

3. Sexual Reproduction Operations

Biologists have long been amazed at the ability of natural selection to solve difficult mathematical optimization problems. For example, it is well known that the hexagons in a bee’s eye are precisely the size required to optimize resolution in bright light. And Krebs, Kacelnik and Taylor (1978) studied a group of great tits in an aviary containing two food dispensers. Each dispenser, when landed on by a bird, had a certain characteristic probability of releasing food. They found that the birds visited the two machines according to the optimal strategy.
dictated by Bayesian probability theory and Laplace’s Principle of Indifference.

This type of work shows that the evolutionary process has a very high capacity for the optimization of mathematical functions. It was part of the inspiration for John Holland’s invention of the genetic algorithm—a ingenious technique which seeks to solve mathematical optimization problems by mimicking biological evolution. As hinted above, this algorithm is intimately bound up with the logic of neural self-organization.

The genetic algorithm rests upon a simple mathematical model of the genetic process, which I will call binary genetics. Binary genetics is an oversimplification, but is an excellent computational tool, and it is also useful conceptually: it reveals aspects of the genetic process that tend to be obscured by the complex peculiarities of biological structures.

Let us assume that the genetic material of any entity can be encoded in a straightforward way as a binary sequence. From the point of view of DNA it would be more natural to consider sequences over 4 different digits, and from the point of view of protein evolution it would be more natural to consider sequences over a few dozen different digits. But for the sake of simplicity, let us stick with binary sequences. Re-coding a base 4 or base 40 sequence in binary is not a particularly arduous task, and its algorithmic complexity is negligible.

Following Holland (1975), then, let us call a binary sequence of length N a genotype. Let A denote the space of phenotypes—e.g. organisms or proteins, or in the next section neural clusters. Each genotype x denotes a certain probability distribution P_x on A, where P_a(x) denotes how likely it is that genotype x gives rise to phenotype a. For simplicity, it is often assumed that each genotype leads infallibly to exactly one phenotype (that, for each x, P_x takes on one particular value with probability 1).

The individual bits of the binary sequences may be called “genes”, and the values 0 or 1 are the “alleles”, the possible values which the genes can take.

Each genotype x possesses a certain fitness. In general, the fitness of a phenotype at time t depends upon: 1) the phenotypes of other evolving entities at time t, 2) the non-evolving environment at time t. We may write the fitness of x as F(P_x,E), where E is the environment at time t, inclusive of other evolving organisms. In genetics, it is often assumed that F(P_x,E)=F(P_x), which is allowable if the effect of a phenotype on its environment is negligible. However, in Chapter 4 we considered a number of arguments against this assumption.

In the context of binary genetics, genetic mutations assume a particularly simple form: a genotype mutates in the i’th position when its i’th bit changes from 0 to 1 or from 1 to 0. And sexual reproduction, or crossover, is almost as simple. Sexual reproduction takes parts of the genotype of one parent and combines them with parts of the genotype of another. In order to correspond with real genetics, this should be done by splicing. The two parent genotypes, say x and y, should be partitioned in terms of binary sequences P_i, q_i so that the length of P_i is the length of q_i:

\[ x = \varphi_p \varphi_q \ldots \varphi_p \varphi_q \]

An offspring of x and y is then any binary sequence which differs in only a few places from something of the form \( r \varphi_q \ldots r \varphi_p \), where \( r \) is equal to either \( p \) or \( q \). The reason why it is allowed to differ in a few places is, of course, that mutations may occur in the process of reproduction.

If the effect of a phenotype on its environment is ignored, then the process of binary evolution yields an intriguing algorithm for function optimization. Although this is a woefully inadequate analysis of evolutionary process, there is much to be learned from it anyway. In function optimization applications, k is usually fixed at 2, so that crossover involves taking the beginning of one sequence and splicing it onto the end of another. However, it is not clear that this serves any purpose other than simplification; in some cases a higher k may lead to more efficient optimization.

To make things concrete, let’s write down an arbitrary example. Suppose one wants to find the maximum of a function \( f: [0,1)^{10} \rightarrow \mathbb{R} \), say

\[ f(x_1,\ldots, x_{10}) = x_1 + x_2^2 + \ldots + x_{10}^{10} \]

To solve this problem using the genetic algorithm, one first settles on a degree of accuracy. Say one requires six digits of accuracy. Then, in binary, this means 20 bits of accuracy. One can then encode each element \( x=(x_1,\ldots, x_{10}) \) of \([0,1)^{10}\) to within 20 bits of accuracy, as a binary sequence of length 200.

There are several ways of doing this encoding. The standard method, however, is as follows: the first 20 bits of the sequence correspond to the first 20 bits of the binary expansion of \( x_1 \), the second 20 bits correspond to the first 20 bits of the binary expansion of \( x_2 \) and so on. Each binary sequence of length 200 is a genotype, and the corresponding vector \( x \) is the corresponding phenotype. The value of \( f \) at the phenotype of a sequence is called the fitness of the sequence. In this case, the maximum fitness occurs at the phenotype \((1,1,1,1,1,1,1,1,1,1)\), and the genotype which best approximates this value
is 11111...11111.

In order to optimize the function $f$, one takes an initial population of, say, 50 sequences (following the biological analogy, these sequences will sometimes be called individuals). At each time step, the old population is turned into a new population according to the following three steps:

1) some individuals mutate;
2) some pairs of individuals reproduce sexually — generally one puts a bias in the process so that fitter pairs are more likely to reproduce;
3) a new population is selected, biased in some way say, toward the new offspring, or toward the most fit.

Although it tends to be rather slow, this algorithm is remarkably effective at optimizing "difficult" objective functions $f$.

There are many variants to the algorithm (Goertzel, 1992a). But the basic concept is always the same. Some population of entities is selected. Parts of entities are combined with parts of other entities to form new entities. From the combined pool of new and old entities, a smaller pool of entities is selected, with some bias toward the fittest. The genetic algorithm is an abstraction of the process of **evolution by natural selection of a sexually reproducing population**. Its effectiveness demonstrates quite elegantly that sexual reproduction is an effective optimization technique.

**Crossover of Software**

The applicability of genetic optimization to the evolution of software is quite obvious. New programs are formed by combining parts of old programs. This happens on several levels. First of all, programmers write new programs by combining various programming tricks which they have learned from observing how other programs are put together. And secondly, new programs are explicitly written to combine features of two or more previous programs.

These processes are very evident, for example, in the animation programming involved in video games. A new video game is made by combining the best features of previous video games, and adding a couple new twists. Super Mario Brothers is a combination of features from maze games, in which the player wanders through various explicit and hidden passages in search of some goal, and features from combat games, in which the player attacks and is attacked in real time. It is crossed over from maze games and combat games, and in this case the offspring has proved fitter than either of the parents. There is, apparently, a certain emergent satisfaction involved in playing Super Mario Bros. which is not present in playing either maze games or combat games alone.

Similar examples may be found in word processing, programming languages, and all other branches of the software market. For example, the C language, with its notion of pointers, may be understood as a crossover of Pascal with assembler. And C++ may be thought of as a crossover of C with Smalltalk.

**4. Self-Referential Creation**

Now let us turn to process (3). In his recent book *Self-Modifying Systems* (1991), George Kampis makes the following extremely useful definition:

An abstract component-system can be defined by the following properties:

a) — there is a finite set of non-dividable and permanent building blocks, drawn from a given pool
b) — there is an open-ended variety of the different types of admissible components, built up from the building blocks according to some composition rule (which may be explicit or implicit)
c) — the components of the system are assembled and disassembled by the processes of the system such that every admissible component is also realizable. (p.199)

Roughly speaking, a component system consists of a collection of components, each of which can act on other components to produce new components.

The main biological example of a component system is a “molecular soup” full of organic molecules acting on one another to form new molecules. Psychologically, on the other hand, one is supposed to think of ideas acting on each other to produce new ideas. For illustrative purposes, however, Kampis suggests that the reader visualize the “non-dividable building blocks” as LEGO blocks, and the “admissible constructions” as different possible structures buildable out of LEGO blocks. One must merely imagine that each LEGO structure contains some appropriate means for acting on other LEGO structures to produce new LEGO structures.

Kampis’s “Main Theorem” is that component-systems are uncomputable — they cannot be simulated by Turing machines. However, as I have argued in (Goertzel, 1993b), they can potentially be simulated by stochastic computers — computers which incorporate a truly random element.

To see this, let us pursue the LEGO example.
It would be easy to build a computable LEGO universe following Kampis’s instructions. For the set of all LEGO structures is countable, and may therefore be mapped into the set of binary sequences, in a one-to-one manner. And each binary sequence may be represented as a Turing machine program, i.e. as a map from binary sequences to binary sequences. Therefore, using Turing machines, each LEGO structure could be interpreted as a function acting on other LEGO structures. The only problem with this arrangement is that it does not satisfy clause (c) of the definition of component-system. Not every LEGO structure is realizable by our dynamics. Only some computable subset of LEGO structures is realizable.

Next, suppose one adds a random element to one’s Turing machine. Suppose each component of the Turing machine is susceptible to errors! Then, in fact, every possible LEGO structure becomes realizable! Structures may have negligibly small probability, but never zero probability! This is an example of a component-system which is computable by a stochastic Turing machine. So, while Kampis’s Main Theorem shows that component-systems are not computable by Turing machines, no such theorem holds for stochastic computers!

**Immune Systems as Component-Systems**

In (Goertzel, 1993b) it is argued that immune systems and brains qualify as component-systems according to Kampis’s definition. Here let us just consider immune systems — by far the simpler of the two cases. The immune system is complicated as well as complex, and it contains many different kinds of cells. But the simplest mathematical models deal only with B-cells, and that is what we shall do here. Let us begin with the approach of de Boer et al (1991), in which each antibody type in the immune system is associated with an integer sequence of length N.

To be realistic, of course, antibodies should be modeled as three-dimensional rational arrays, since they are three-dimensional objects; but for the points we are making here, it is immaterial whether antibodies are associated with 1-D or 3-D arrays. In the 1-D model, we may think of a B-cell as a pair $V_i = (A, B_i)$, where $B_i$ is an integer sequence representing the shape of the B-cell, and A is an integer code sequence. The code sequence specifies what happens when one forms $f(f_i).

Specifically, what happens most of the time when $f(f_i)$ is formed is nothing. But if the conditions are right, the effects can be drastic. Define the raw match between two B-cells $V_i$ and $V_j$ as the maximum number of consecutive bits in which the corresponding sequences $B_i$ and $B_j$ are different.

And define the match between two sequences as $\max(0, \text{raw match}(B_i, B_j) - T)$, where $T$ is some given threshold. In terms of component-systems, then, we may think of the dynamics of the immune system as specifying how a B-cell $V_i$ acts on those B-cells $V_j$ for which match($B_i, B_j$) is large, thus causing the creation of new antibodies.

There is a great deal of biological subtlety involved here. In the crudest formal model, however, what happens is as follows: if $f(f_i)$ is formed and match($B_i, B_j$) is large, then with a certain probability, the cell $V_i$ is killed by the cell $V_j$ (in fact, this killing takes place indirectly, via antibodies; but we do not need to consider these details here). But when the proportion of B-cells with shape $B_j$ that are killed falls within a certain critical range, then cells of this shape are stimulated to reproduce. New B-cells are created.

Some of these new cells are identical to $V_i$, and some are new types $V_k$, which have have shape sequences similar, but not identical to $B_i$. This is somatic hypermutation. It is the creation of new B-cells by certain old B-cells acting on other old B-cells. There is randomness in the process because there is no deterministic way of telling exactly which new types of B-cells will be created.

This B-cells-only model is an extreme oversimplification. But the more accurate models are similar in spirit. Cells in the immune system act on one another, thus stimulating one another to produce new cells. Sometimes these new cells are copies of old cells, but sometimes they are structurally novel.

**The Software Market As A Component-System**

The relation of component-systems with computer software programs is no less clear. A program, via the medium of a programmer, can act on another program to produce a new program. For instance, Windows 3.0 acted on WordPerfect 5.1, to produce WordPerfect for Windows. Turbo Pascal acted on various mainframe C compilers to produce Turbo C. And in turn, Turbo C acted on these other C compilers — its excellent error-checking features encouraged systems programmers to add error-checking to mainframe C compilers.

This is very similar to the way a B-cell of type $B_i$ can act on a B-cell of type $B_j$, thus causing the creation of a B-cell of a new type related to but not identical to $B_j$. As in the case of the immune system, the acting entity does not physically form the newly created entity. It merely acts as the stimulus for creation. But, given the abstract definition of
component-system, this is good enough.

5. Conclusion

We have considered three general principles of complex system structure and function: natural selection, sexual reproduction, and self-referential creation. All of these principles, we have seen, manifest themselves in the software market.

This analysis has not led us to any radical new conclusions regarding the nature of the software market. Every programmer knows that the usefulness of a program depends on the nature of the other programs with which it will be used. Every programmer knows that programs can influence other programs, and that parts of two different programs can be combined to form new programs. But what the system-theoretic approach does accomplish, is to point out that these features of the software market are not flukes. They are not irrelevant oddities, but rather crucial aspects of complex system behavior.

The systemic point of view serves to guide our thoughts in new directions. For instance, consider the following question: over the next few decades, do you expect the software market to become more or less complex? If your answer, like mine, is the former, then you are faced with another question: what forms will this increased complexity take?

Here the system-theoretic approach shines. If the software market becomes more complex, declares general systems theory, then it will display the universal strategies of complexity yet more prominently. The fitness of a program will depend yet more sensitively on the programs which surround it. Programs will act on one another more drastically, and more rapidly. Combination of parts of programs to form new programs will become more automated, more routine. These are general predictions, but given an appropriate formalism for describing programs, it is possible to make them more concrete.

Finally, let us take a large step back and consider what we have done. We have discussed a very specialized market — the software market. As already pointed out, more conventional markets tend to display much less complexity — in the wheat market or the oil market, the three optimization strategies discussed above are detectable only as subtle hints. But now let me pose another question: of these three markets, which is a better model of the market of the future? If, as I suspect, the correct answer is the software market, then the conclusion is obvious. More and more, markets will come to display the general properties of very complex systems. Conventional economics will become less and less relevant, and system-theoretic economics more and more necessary.

References


Spencer, Herbert (1878). Sociology.