# Surfing the Edge of Chaos: Applications to Software Engineering

**Juan C. Nogueira**
**Carl Jones**
**Luqi**
Naval Postgraduate School
2, University Circle
Monterey, CA. 93943 USA
+1 (831) 656 2093
jcnoguei@nps.navy.mil

## Abstract

This paper discusses the problems of software engineering as the weakest link in the development of systems capable of achieving information superiority. Fast changes in technology introduce additional difficulties in terms of strategic planning, organizational structure, and engineering of software development projects. In such complex environment, a new way of thinking is required. We analyze the introduction of complex adaptive systems as an alternative for planning and change. The strategy of "competing on the edge" is analyzed showing the risks and the skills required navigating on the edge. We discuss the feasibility of using this theory in software engineering as an alternative to bureaucratic software development processes. We present also some recommendations that could help to acquire competitive advantage in software development, hence achieve information superiority.

## 1. Introduction

As software systems increased in complexity, software development evolved form a primitive art into software engineering. Methodologies and software tools were developed to help development processes. Most of the present tendencies (DOD-STD-2167A, ISO-9001, SEI/CMM) try to standardize processes, emphasizing planning and structure (Humphrey, 1990). Some authors criticize those approaches stating that they underestimate the dynamics of the software development (Bach, 1994), (Abdel-Hamid, 1997). Others question that activities such as research and development are not addressed by TQM principles (Dooley et al., 1994).

In 1994 Gibbs claimed “despite 50 years of progress, the software industry remains years—perhaps decades—short of the mature engineering discipline needed to meet the demands of an information-age society.” Many researchers have treated the problem using different approaches: tools, formal methods, prototyping, software processes, etc. However, this assertion remains true today.

The typical software engineering process is a succession of decision problems trying to transform a set of fuzzy expectations into requirements, specifications, designs, and finally code and documentation. The traditional waterfall software process failed to accomplish their purpose because it applied a method valid for well-defined and quasi-static scenarios. This hypothesis is far from the reality. Today, modern software processes (Boehm, 1988), (Luqi, 1989) are based on evolution

and prototyping. These approaches recognize the fact that software development presents an ill-defined decision problem and they fail in assessing automatically the risk.

In our view, software development projects present special characteristics that require to be solved in order to achieve an improvement in the state of the art. These particularities affect the strategic planning, the organizational structure, and the engineering applied to software. In these three areas chaos theory can provide clues for possible solutions.

## 2. The strategic planning issue

Traditional approaches to strategic planning emphasize picking a unique strategy according to the competitive advantages of each organization. Porter's five-force approach (Porter, 1980), assumes that there exists some degree of accuracy in the prediction of which industries and which strategic positions are viable and for how long.

In a high-velocity scenario the assumption of a stable environment is too restrictive. Customers, providers, competitors, and potential competitors, as well as substitute products are evolving faster than expected. The introduction of new information technology tools, the Internet and the globalization of the markets are contributing to this phenomenon, and nothing seems to reverse the process. The failure of long-term strategic planning is not a failure of management; it is the normal outcome in a complex and unpredictable environment. A growing number of consultants and academics (Santosus, 1998), (Brown & Eisenhardt, 1999) are looking at complexity theory, to help decision-makers improve the way they lead organizations.

How useful could a map of a territory that is constantly changing its topography be? In fast changing environments, survival requires a refined ability to sense the external variables. Traditional approaches rely on strategic planning and vision. However, in unstable environments planning would not be effective because it is impossible to predict the scenario's evolution in terms of markets, technologies, customer's needs, etc. Organizations relying only on one vision supported by a tight planning, risk paying little attention to the future. Consequently, their sensing organs are blind to foresight the future. A certain amount of inertia and commitment to the plans is required to prevent erratic changes caused by reaction diverse variables.

If the time window of the opportunities is shrinking, a different form of thinking is required. The present technological situation can be described as a fast succession of short-term niches. The ability to change is the key of success for surviving in such a variable environment. In a systemic approach, the General Systems Theory establishes that organizations are systems whose viability depends on some basic behaviors (von Bertalanfy, 1976):

(a) Ability to sense changes in the environment. This is the most primitive form of intelligence, if it is not present the probabilities of survive are minimum.

(b) Ability to adapt to a new environment modifying the internal structure and behavior. The system tries to auto-regulate to survive the crisis in hostile scenarios, or take advantage of the opportunities in favorable ones.

(c) Ability to learn from the past, anticipating the auto-regulation behaviors and structure before the environment change. This ability requires intelligence able to infer conclusions from the past according to the context of the variables sensed on the present.

(d) Ability to introduce changes in the environment, making it more favorable to the system's needs. In this case, the system has developed the technology (know how and tools) to exert power over the environment.

Any mechanical or computing system has some or all of these abilities. We find these same abilities in any form of life. The more developed the system is, the more of the above characteristics has. Darwin's Evolution Theory validates this line of reasoning. Natural selection, acting on inherited genetic variation through successive generations over the time is the form of evolution. Variation is the way used by biological systems to probe the environment presenting many alternatives, some of them ending on failure but a few very successful. This process is an inefficient but very effective way of improvement.

Experiments can provide a certain amount of knowledge about the future. In some sense, probes are mutations in small scale that can cause only small losses. The results give insights to discover new options to compete in the future and stimulate creative thinking. The research investment pays dividends when a new way of competition is discovered altering the status quo's rules.

When the changes in the environment occur too fast, sensing the variables becomes more difficult. It is possible that a specialized organ was not able to react on time to record the metric and transmit the alert. In this case, the system starts to lose information threatening its own viability. When the changes in the environment are too drastic, even if the sensor organs detect the change, the inference organs may not be able to determine an effective course of action because they do not have a previous experience, or because the decision-making process requires more time. This situation also threats the viability of the system in the long run. The effects of drastic variations and high rate of change over systems can be visualized with simple experiments: a) increasing the speed of transmission in a communication channel beyond some limit will provoke the lost of part or the entire message, b) modifying the pH in the soil beyond a certain limit can cause the death of a plant.

The same syndrome can be recognized in any type of organization. We purpose to employ a new strategy. "Competing on the Edge" is a new theory defines strategy as the creation of a relentless flow of competitive advantages that, taken together, form a semi-coherent strategic direction (Brown & Eisenhardt, 1999). The key driver for superior performance is the ability to change, reinventing the organization constantly over the time. This factor of success can be applied to software engineering as well as to other decision problems with similar characteristics.

If the environment is moving, like in surfing, the best way to remain in equilibrium is by being in the rhythm. Successful corporations such as Intel or Microsoft are in perpetual movement, launching new products with certain rhythm. Intel is faithful to its founder's (Moore) law: the power of the microprocessors double every eighteen months. Microsoft has a proportional pace on the software sector.

## 3. The organizational issue

The second unresolved issue is organizational. We think that many of the problems on current software projects have organizational roots. This opinion is also supported by (van Genutchen, 1991)[1] and (Capers Jones, 1994)[2].

Perrow (Burton et al., 1998), introduced a two-dimensional classification of the technology (Fig. 1). The first dimension is the analyzability of the problem varying from well defined to ill defined. The second dimension is the task variability, which means the number of expected exceptions in the tasks.

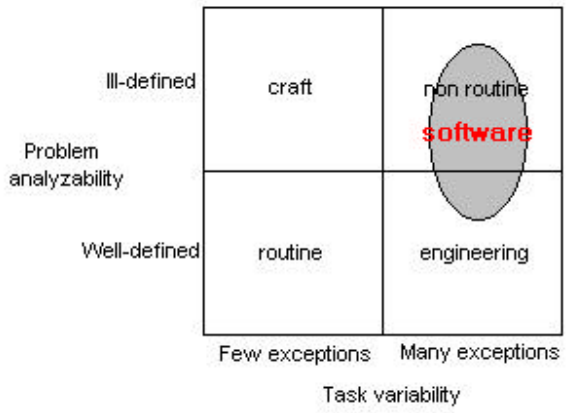


**Figure 1: Perrow's classification of technology**

Software engineering, unlike other forms of engineering, presents decision problems on more than one quadrant. Fuzzy problem definition and high number of exceptions characterize requirements, analysis and design phases. Only code generation and testing phases are in the engineering quadrant. The main volume of activities resides in the non-routine quadrant. Hence, highly skilled personnel, low formalization and centralization, high information processing demand, and coordination obtained through meetings is required. In our opinion software engineering is not the only discipline in this quadrant. The challenges imposed by hyper competition create similar characteristics than in software engineering developments. So, the rules of engagement proved effective for one discipline could result useful in the other.

A second line of research (Burton & Obel, 1998), introduced a classification based on four-variable model: equivocality, environmental complexity, uncertainty and hostility. Equivocality is "the existence of multiple and conflicting interpretations", it is a measure of the lack of knowledge or the level of ignorance whether a variable exists in the space. Uncertainty is the lack of knowledge about the likelihood of values for the known variables. Environmental complexity is the number of factors in the environment affecting the organization and their interdependency. Finally, hostility *is "the level of competition and how malevolent the environment is."*

In Table 1, we disregard the fourth variable: hostility. Hostility is a discontinuity of the environment. When it is high, then it overrules other factors. In highly hostility scenarios only a highly centralized organization ("regular army"), or a low-formal-low-complex organization ("guerilla") are the possible alternatives.

Software development scenarios usually correspond to high equivocality, high environmental complexity and high uncertainty scenarios (dark gray in the matrix), which correspond to low

---

[1] Van Genuchten found that 45% of all the causes for delayed software are related to organizational issues.

[2] Capers Jones found that on military software developments the two more common threats are excessive paperwork (90% of the time) and low productivity (85% of the time).

formalization and low organizational complexity, with centralization inverse to the environmental complexity. The recommended organization could be ad hoc or matrix with coordination by integrator or group meeting. The information exchange is rich and abundant. The incentive policy should be based on results.

| Equivocality | Enviromental Complexity | Uncertainty | Formalization | Organizational Complexity | Centraliza-tion |
|---|---|---|---|---|---|
| Low | Low | Low | High | Medium | High |
| Low | Low | High | Medium | High | Medium |
| Low | High | Low | High | Medium | Medium |
| Low | High | High | Medium | High | Low |
| High | Low | Low | Medium | Medium | High |
| High | Low | High | Low | Low | High |
| High | High | Low | Medium | Medium | Low |
| High | High | High | Low | Low | Low |

**Tab1e 1: Burton & Obel classification**

Understanding these organizational characteristics inherent of software projects is required to create a more fitted software process. The application of a quasi-chaotic process keeps the organization in continuous movement with positive effects its internal behavior. The rhythmic change avoids manager's tendency to slow down the process or introduce changes too often. The periodic changes create small amounts of chaos that maintain the organization in the edge.

## 4. The engineering issues

Despite 50 years of progress, the software industry remains immature to meet the demands of an information-age economy. Many researches have treated the problem using different approaches: formal methods, prototyping, software processes, etc. However, the problem remains open today. The third unresolved issue is a set of engineering problems concerning software processes, risk assessment, and reuse.

### *4.1. The software process problem*

Studies have shown that early parts of the system development cycle such as requirements and design specifications are especially prone to error (Luqi, 1989). Problems originating in the early stages often have a lasting influence on the reliability, safety and cost of the system. This effect is particularly notorious in projects involving multiple stakeholders with different points of view. Evolutionary software processes offer an iterative approach to requirement engineering to alleviate the problems of uncertainty, ambiguity and inconsistency inherent in software developments. Experience suggests that building and integrating software by mechanically processable formal models leads to cheaper, faster and more reliable products. Moreover, prototyping can improve the capture of change in requirements and assumptions during the development process. Prototypes are useful to demonstrate system scenarios to the affected parties as a way to: a) collect criticisms and feedback that are sources for new requirements; b) enable early detection of devia-

tions from users' expectations; c) trace the evolution of the requirements; and d) improve the communication and integration of the users and the development personnel.

Despite the unquestionable benefits of evolutionary software processes, we have some concerns. The first concern is that prototyping poses a problem to project planning because of the uncertain number of cycles required to construct the product. Most project management and estimation techniques are based on linear layouts of activities, so they do not fit completely.

Second, evolutionary software processes do not establish the maximum speed of the evolution. If the evolutions occur too fast, without a period of relaxation, it is certain that the process will fall into chaos. On the other hand if the speed is too slow then the productivity could result affected. The correct rhythm for software processes has not been researched and remains on the hands of the project manager.

Third, software processes should be focused on *flexibility* and *extensibility* rather than in *high quality*. This assertion sounds scary. However, we should prioritize the speed of the development over zero defects. Extending the development in order to reach high quality could result in a late delivery of the product, when the opportunity niche has disappeared. This paradigm shift is imposed by the competition on the edge of chaos.

### *4.2. The risk assessment and estimation problems*

Developing software is still a high-risk activity. Despite the advances in technology and tools, little progress has been done in improving the management of software development projects. Part of the problem is misinterpretation of the importance of risk management that is usually viewed as an extra activity layered on the assigned work, or worst, as an outside activity that is not part of the software process (Hall, 1997), (Karolak, 1996).

Software development processes such the hypergraph model for software evolution (Luqi, 1989), or the spiral model (Boehm, 1988), improved the state of the art. However, all of them have a common weakness: risk assessment.

On the software evolution domain, risk assessment has not been addressed as part of the model. In the various enhancements and extensions, the graph model did not include risk assessment steps; hence risk management remains as a human-dependent activity that requires expertise.

On the evaluation of the spiral model, one of the difficulties mentioned by Boehm was: *"Relying on risk-assessment expertise, the spiral model places a great deal of reliance on the ability of software developers to identify and manage sources of project risk."* (Boehm, 1988).

Many researches have addressed the problem of risk assessment following only one perspective. The available tools for risk assessment are guidelines for practices, checklists, taxonomies of risk factors and few metrics. All these methods work fine if a) there is a human educated on risk assessment, and b) he/she has enough experience. Such resources are very scarce and it is difficult to leverage their expertise over large organizations.

The main line of previous research has addressed the problem in parallel with the development process using informal methods. Basically the proposed methodologies are lists of practices and

checklists (SEI, 1996), (Hall, 1997) or scoring techniques (Karolak, 1996) that are dependent on human expertise.

The second weakness on risk assessment is caused by the difficulties in estimate the development tiem. The industry has been using three classes of tools to estimate effort and time that can be applied at different moments during the life cycle, each category being more precise than the previous one but arriving later on the life cycle:

a) Very early estimations. This category includes very crude approximations done during the beginning of the process usually by subjective comparisons using previous projects.

b) Macro models. This category includes Basic COCOMO, COCOMO II (application composition model), Putnam, Function Points, etc. The estimation is done after completing the requirements phase.

c) Micro models. This category includes intermediate and detailed COCOMO, COCOMO II (early design and post-architecture models), and Pert/CPM/Gantt techniques. The estimation is done after the design when it is possible to have a work breakdown structure. The project estimate is the integration of all module estimates.

A detailed discussion of these techniques is outside the scope of this paper; the details can be read in (Albrecht, 1979 and 1983), (Boehm, 1981 and 2000), (Londeix, 1987), (Putnam, 1980, 1992, 1996, and 1997). None of these techniques consider the following characteristics of software projects:

a) Requirement volatility
b) Personnel volatility
c) Time consumed by communications, exceptions and noise in the process. All the methods use size as an input parameter via some kind of derivation from complexity. In many cases the methods to compute such complexities and sizes are questionable (Kitchenham, 1993 and 1997), (Kemerer, 1993).

Recently, NPS developed a formal model for risk identification and assessment for evolutionary software processes that solves the problems of automation, human dependency, and estimation (Nogueira et al. 2000). This research is focused on studying software project risk assessment from a different perspective, viewing risk assessment as the prediction of success of the project given a set of characteristics, a probabilistic model based on Weibull distribution, and learning from each successive cycle on the process.

### *4.3. The reuse problem*

Even if the industry claims for the use of flexible and extensible architectures from which reusable components could be integrated as a way of generating applications, the reality is that the standard does not exist. Different architectures are competing for becoming the de facto standard. Microsoft proposes the Distributed network Architecture (DNA) based on DCOM and ActiveX. Sun and other OMG members propose the Enterprise Computing Platform (ECP) based on IIOP and CORBA. Each alternative presents advantages and disadvantages and it is not easy to forecast the winner.

## 5. The edge of chaos

In the fiction novel "The Lost World", Michael Crichton explains why some species evolve while others become extinct. *"Complex systems tend to locate themselves at a place we call 'the edge of chaos' ... as a place where there is enough innovation to keep alive system vibrant, and enough stability to keep it from collapsing into anarchy... Only at the edge of chaos can a complex system flourish."*

Chaos theory describes a specific range of irregular behaviors in systems that move or change (James7). Chaotic does not mean random. The primary feature distinguishing chaotic from random behavior is the existence of one ore more attractors. Without the existence of such attractors the quasi-chaotic scenarios could not be repeatable. It is important to realize that a chaotic system must be bounded, nonlinear, non-periodic and sensitive to small disturbances and mixing. If a system has all these properties can be driven into chaos.

The edge of chaos is defined as *"a natural state between order and chaos, a grand compromise between structure and surprise"* (Glenn, 1996). The edge of chaos can be visualized as an unstable partially structured state of the universe. It is unstable because it is constantly attracted to the chaos or to the absolute order.

We have the tendency to think that the order is the ideal state of nature. This is could be mistake. Research on Organizational Theory (Stacey, Nonaka, Zimmerman); Management (Stacey, Levy); and Economics (Arthur) support the theory that operation away from equilibrium generates creativity, self-organization processes and increasing returns (Roos, 1996). Absolute order means the absence of variability, which could be an advantage under unpredictable environments.

Change occurs when there is some structure so that the change can be organized, but not so rigid that it cannot occur. Too much chaos, on the other hand, can make impossible the coordination and coherence. Lack of structure does not always mean disorder. Let illustrate this idea with an example. We can agree that there is little structure in a flock of migratory ducks in a lake. However, few minutes after they start flying some order appear and the flock creates a V shape formation. This self-organized behavior occurs because a loose form of structure exists. Experiments with intelligent agents governed by three rules (a) try to maintain a minimum distance from the other objects in the environment, including other agents; b) try to match the speed of other agents in the vicinity; and c) try to move toward the perceived center of mass of the agents in the vicinity), show the same behavior. Independently of the starting position of the agents, they always end up in a flock. Even if an obstacle disturbs the formation, the pseudo-order is recovered some time later. This self-organized behavior emerges despite the absence of leadership and without an explicit order to form a flock.

A more interesting example is the behavior of software development teams. A recent article (Cusumano, 1997), describes the strategies of Microsoft to manage large teams as small teams. Dr. Cusumano says "*What Microsoft tries to do is allow many small teams and individuals enough freedom to work in parallel yet still function as one large team, so they can build large-scale products relatively quickly and cheaply. **The teams adhere to a few rigid rules that enforce a high degree of coordination and communication***."

This is an exact description of the emerging behavior in a complex adaptive system. It is self-adaptive because the agents realize the adjustment to the environment, and it is emergent because

it arises from the system and can only be partly predicted. As in the example of the ducks, few rules of interaction between the agents (in this case people) generate a performing behavior. The three rigid rules at Microsoft are: a) developers integrate their work daily forcing the synchronization and testing of the work; b) developers responsible for bugs must fix them immediately, and are responsible for the next day integration; and c) milestone stabilization is sacred.

Complex adaptive systems, as the one just described, are made up with multiple interacting agents. The emergence of the complex behavior requires some conditions. The first condition is the existence of more than one agent. A second condition is that agents must be sufficiently different to each other that their behavior is not exactly the same in all cases. When agents behave exactly the same way exhibit predictable, not complex, behavior. Finally, a third condition is required. Complex adaptive behavior only occurs in the edge of chaos.

## 6. Some of the risks of being in the edge of chaos

Limiting the structure in organizations can be useful in situations when innovation is critical or when is required to revitalize bureaucracies. However, if the structure is debilitated beyond a certain minimum, it can conduct to an undesired state. Some traits can alert the eminence of such anarchic situation known as the "chaos trap" (Brown & Eisenhardt, 1999): a) emerging of a rule-breaking culture, b) missing deadlines and unclear responsibilities and goals, and c) random communication flows.

On the other hand focusing in hierarchy and disciplined processes, emphasis on schedules, planning and job descriptions may conduct to a steady inert bureaucracy. Organizations in such state react too late failing to capture shifting strategic opportunities. This is the case of a "bureaucratic trap", where there are also some observable warning traits: a) rule-following culture, b) rigid structure, tight processes and job definitions, and c) formal communication as the only channel.

The alternative is "surfing" the edge of chaos avoiding both attractors. That requires limited structure combined with intense interaction between the agents, giving enough flexibility to develop surprising and adaptive behavior. Organizations in this state are characterized by having an adaptive culture. People expect and anticipate changes. A second characteristic is that the few key existing structures are never violated. Finally, real time communication is required throughout the entire organization.

Being in the edge of the chaos implies an unstable position. Some perturbations can cause the rupture of this delicate equilibrium and the fall into one of the two steady states. A potential perturbation factor is the organizational collaboration style. Too much collaboration can disturb the performance of each agent and consequently, the whole system is affected. On the other hand, too little collaboration destroys the advantage of acting organized and leads to paralysis.

Another sources of perturbation are the tendency to be tight to the past and cultural idiosyncrasy, or by contrary, to loose the link with the past. In one case, the change becomes impossible. In the other case, the assets from previous experiences are not capitalized. The equilibrium point is called regeneration. In such unstable state, mutation can occur. Therefore the inherited characteristics that give competitive advantage in a certain scenario can be perpetuated, and new variations are introduced. If too little variation exists, natural selection fails. This process permits that complex adaptive systems change over the time following a Darwinian pattern.

(Kauffman, 1995) introduced the concept of fitness landscape. We can understand this concept observing the behavior of species. In the competition for survival, species attempt to alter their genetic make-up by taking adaptation trying to move to higher "fitness points" where their viability will be enhanced. Species that are not able to reach higher points on their landscapes may be outpaced by competitors who are more successful in doing so. If that occurs the risk of extinction increases. The same principle applies between predator and prey. Each development in the abilities of one species generates an improvement on the abilities of the other. This concept is called co-evolution.

Certain higher fitness points have more value to some species than to others. The contribution a new gene can make to a species' fitness depends on genes the species already has. As more complicated is the genetic pattern (more evolved), the probability of conflict of a new adaptation increases slowing down the speed of variations.

Natural selection is an effective, but not generally efficient way to evolve. The process requires some amount of mutation to avoid the sudden convergence on suboptimal characteristics. Some of the characteristics lost in the past can be reintroduced being useful in the new scenario. Many errors are committed during this blind process. A more efficient way to evolve is by recombination of the pool of genes using genetic algorithms. This technique has been applied to improve the performance of robots, however the idea can be used to improve the competencies of organizations. If too much or too less variation occurs the result always conduct to the failure of the system.

## 7. Application in software engineering

Chaos in software development comes from various sources: a) the intrinsic variable nature of requirements, b) the changes introduced by new technologies, c) the dynamics of the software process, and d) the complex nature of human interaction. These conditions are sufficient for the development of complex adaptive systems where the agents are software developers or parallel collaborative projects.

Software development scenarios usually have high equivocality, high environmental complexity and high uncertainty. The suggested organizational structure to deal with such scenarios (Burton & Obel, 1998) should have low formalization and organizational complexity, centralization inverse to the environmental complexity, and rich and abundant information exchange. The recommended organization should be ad hoc or matrix, with coordination by integrator or group meeting. This organizational style is difficult to achieve when the organizations are large.

A simple solution can be recognized at Microsoft (Cusumano, 1997): a) parallel developments by small teams with continuous synchronization and periodically stabilization, b) software evolution processes where the product acquires new features in increments as the project proceeds rather than at the end of a project, c) testing conducted in parallel as part of the evolution process, and d) focus creativity by evolving features and "fixing" resources. Cusumano observed that small development teams were more productive because: a) fewer people on a team have better communication and consistency of ideas than large teams, and b) in research, engineering and intellectual work individual productivity has big variance. Software development requires teamwork, more specifically organized work. So we require understanding the dynamics of organizations as artifi-

cial social entities that exist to achieve a specific purpose, in this case to develop software. Such organizations are made up of individuals who accomplish diverse desegregate activities that require coordination and consequently information exchange.

A shift from the traditional long-term development organizations is required. Virtual teams created as temporary dynamic project-oriented structures, with a composition of skills matching exactly the objectives could improve the current performances. Such virtual organizations are not exposed to bureaucratic loads and do not require to absorb the cost of permanent staff (Senegupta & Jones, 1999).

Larger developments could be achieved by parallel projects loosely coupled sharing a common architecture such CORBA or DCOM. This paradigm enables the possibility of managing large developing organizations as if they were small. In such scenarios, the benefits of complex adaptive systems will occur at two levels. At the micro level, inside each small project, the agents are individuals. Second, at the macro level where the agents are the small projects.

## 8. Conclusion

Complex adaptive systems appear as the most attractive way to deal with changing environments. Besides some indicators introduced by (Brown & Eisenhardt, 1999), the academic research is not mature enough to assert a methodology for competition on the edge. Some enterprises like Microsoft and Intel seem to have discovered and applied this form of strategy since many years ago, but little information have permeated.

We propose a drastic change in the software processes using the benefits of programming in the small to programming in the large. More even, we state the quality-driven paradigm should be revised, and that the objective should be shorter delivery times, flexibility and expansibility.

Despite the obvious differences in terms of hostility, we found several similarities between war and software development scenarios. A depth research is required to evaluate the applicability of this theory to different fields in which uncertainty is a key factor peace keeping operations, joint $C^4I$, and irregular warfare.

## References


(Abdel-Hamid, 1997) Abdel-Hamid, T. Lessons Learned from Modeling the Dynamics of Software Development. Edited by Kemerer, C. McGraw Hill 1997.

(Albrecht, 1979) Albrecht, A. Measuring Application Development Productivity. Proceedings IBM. October 1979.

(Albrecht, 1983) Albrecht, A. and Gaffney, J. Software Function Source Lines of Code and Development Effort Prediction. IEEE Transactions Software Engineering, SE-9, 1983.

(Bach, 1994) Bach, J. The Immaturity of the CMM. American Programmer, September 1994.

(Boehm, 1981) Boehm, B. Software Engineering Economics. Prentice Hall, 1981.

(Boehm, 1988) Boehm, B. A Spiral Model of Software Development and Enhancement. Computer. May, 1988.

(Boehm, 2000) Boehm, B., Madachy R., Selby, R. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. http://sunset.usc.edu/COCOMOII/cocomo.html

(Brown & Eisenhardt, 1999) Brown, S. and Eisenhardt, K. Competing on the Edge. Strategy as Structured Chaos. Harvard Business School Press, 1999.

(Burton & Obel, 1998) Burton, R. and Obel, B. Strategic Organizational Diagnosis ans Design. Developing Theory for Application. Second Edition. Kluwer Academic Publishers, 1998.

(Cusumano, 1997) Cusumano, Michael How Microsoft Makes Large Teams Work Like Small Teams. Sloan Management Review. Fall, 1997.

(Dooley, 1994) Dooley, K. and Flor, R. "Success and Failure in Total Quality Management Initiatives", Proceeding of the Chaos Network, Denver, 1994.

(Hall, 1997) Hall, E. Managing Risk. Methods for Software Systems Development. Addison Wesley, 1997.

(Humphrey, 1990) Humphrey, Watts. Managing the Software Process. Addison-Wesley, 1990.

(James, 1996) James, G. E. Chaos Theory. The Essentials for Military Applications. Naval War College. The Newport Papers, 1996.

(Karolak, 1996) Karolak, D. Software Engineering Management. IEEE Computer Society Press, 1996.

(Kauffman, 1995) Kauffman, Stuart. At Home in the Universe. Oxford University Press, 1995.

(Kemerer, 1993) Kemerer, C. Reliability of Function Points Measurements: A Field Experiment. Communications of ACM, Vol 36 No 2. 1993.

(Kitchenham, 1993) Kitchenham, B., Kansala, K. Inter-item Correlations among Function Points. First International Software metrics Symposium. IEEE Computer Society Press. 1993.

(Kitchenham, 1997) Kitchenham, B., Linkman, S. Estimates, Uncertainty, and Risk. IEEE Software. May-June, 1997.

(Londeix, 1987) Londeix, B. Cost Estimation for Software Development. Addison-Wesley, 1987.

| | |
|---|---|
| (Luqi, 1989) | Luqi. Software Evolution Through Rapid Prototyping. IEEE Computer. May, 1989. |
| (Nogueira et al., 2000) | Nogueira, J.C., Luqi, and Berzins, V. A Formal Risk Assessment Model for Software Evolution. Paper submitted to SEKE 2000. |
| (Porter, 1980) | Porter, Michael. Competitive Strategy. Free Press, 1980. |
| (Putnam, 1980) | Putnam, L. Software Cost Estimating and Life-cycle Control: Getting the Software Numbers. IEEE Computer Society Press. 1980. |
| (Putnam, 1992) | Putnam, L. and Myers, W. Measures for Excellence. Reliable Software On Time Within Budget. Yourdon Press, 1992. |
| (Putnam, 1996) | Putnam, L. and Myers, W. Executive Briefing. Controlling Software Development. IEEE Computer Society Press. 1996. |
| (Putnam, 1997) | Putnam, L. and Myers, W. Industrial Strength Software. Effective Management Using Measurement. IEEE Computer Society Press, 1997. |
| (Roos, 1996) | Roos, Johan. The Poised Organization: Navigating Effectively on Knowledge Landscapes, 1996. http://www.imd.ch/fac/roos/paper_po.html |
| (Santosus, 1998) | Santosus, Megan. Simple, Yet Complex. Business Management CIO Enterprise Magazine. April 15, 1998. |
| (SEI, 1996) | Software Engineering Institute. Software Risk Management. Technical Report CMU/SEI-96-TR-012. June, 1996. |
| (Senegupta & Jones, 1999) | Sengupta, K. and Jones Carl R. Creating Structures for Network-Centric Warefare: Perspectives from Organizational Theory. Command & Control Researc & Technology Symposium. CCRP 1999. Naval War College, 1999. |
| (von Bertalanfy, 1976) | von Bertalanfy, L. General System Theory: Foundations, Development, Applications. Braziller, 1976. |