Proceedings of the
International Conference on Complex Systems
Nashua, NH, 21–26 Sept. 1997.

Y. Bar-Yam (ed.), New England Complex Systems Institute (1997) .
Also on-line in the *InterJournal*.

Assessing Software Organizations from a Complex Systems Perspective

Javier Dolado*
Dept. of Computer Languages and Systems, Univ. of the Basque Country, Spain
Alvaro Moreno
Dept. of Logic and Philosophy of Science, Univ. of the Basque Country, Spain

Nowadays, the words "software process improvement" are buzzwords in the software management activities. Assessments of the status of the software process in organizations are made by using models such as CMM, SPICE or ISO 9000. We see some uses of those models as an example of the technological (or mechanical) paradigm that is trying to pervade most activities of software construction.

We can consider some types of software products as the final outcome of human design processes. Thus, to use mechanical procedures as patterns to follow may not be the apropriate way for dealing with emergent structures. Looking at the biological area, a morphogenetic metaphor allows us to introduce the concepts of deducible emergence and hyperstructures. Thus, software processes could take many appearances, and improvement could be evaluated with respect to what extent processes allow intermediate structures to evolve and reach the final form.

1 Introduction

Software is an element that can be considered complex from different perspectives, and the complexity that arises in software development has been acknowledged for some time. The phrase "software crisis" tries to reflect many of the problems that software development organizations face in their projects. One of the factors that has been blamed for the source of the obstacles in achieving a controlled project is the software process. However, the ideal normative process for development activities is still unknown. In this work we aim to link some of the concepts of the complex systems area to the management of software.

Basically the question that is posed here is whether it is sound to have a mechanistic view of software development given that the social and human aspects are so significant. The view of software creation as a sort of form generation conveys an 'organic perception' of software houses. This suggests that the evaluation of the excellence of an organization has to take into consideration not only the 'manufacturing' processes, but also other less structured, but effective, processes.

2 The Software Process and its Evaluation

A *software process* can be defined as a set of activites, methods, practices and transformations that people employ to develop and maintain software ant the associated products. Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and made effective. There are some methods that aim to assess the state of a software production organization, one of them being the well-known SEI's Capability Maturity Model – CMM– [16]. Other trends are the ISO 9000 [12] guidelines and the new SPICE model.

2.1 Different Evaluations of the Process

Every evaluation of a software process is made under a theoretical framework, whether or not it is explicitly defined. The framework provides the basis for the specific rules to be followed in the assessment. These may be one those previously listed or one of other more general guidelines like Total Quality Management, etc. From the practical point of view, the evaluation takes the general form of a review of the extent to which the areas, aspects or issues are fulfilled in a verification list. The two best known methods (CMM and ISO 9000) provide separate approaches to the evaluation, that of CMM being more flexible. As an equivalence can be established between those models, and since the purpose of this paper is not to dig into the technicalities of each model, we will refer to some aspects of the CMM as it is carried out in the actual practice.

Schematically, this model establishes a 5 level scale
1. Initial, sometimes called "chaotic" (the process is "ad hoc")
2. Repeatable, in which the process depends on individuals
3. Defined, the process is defined and institutionalized
4. Managed, the process is quantitatively measured
5. Optimizing, in which the improvement is fed back to process.

Thus, a software organization is better rated as a higher point is attained. Each of the levels is composed of several *key process areas*. Improvement is achieved by taking action in the activities involved in the key process areas. Reaching the highest level implies that the organization is mature with respect to the software engineering process and that it is able to manage effectively many aspects of the software construction.

While trying not to be simplistic, and acknowledging that technology is an important part in software, we view this

*Work supported by UPV-EHU under project 141.226 - EA 114/96

model as rooted in the technological paradigm of improvement by standardizing repeatable procedures. Several facts can contradict this way of assessment and could lead us to think of that scale as a very artificial evaluation. There are software houses that could be rated at level 1 and, despite this, they are able to produce good software. Although it seems that some economic benefits are obtained as higher point in the scale is attained [8], it is not clear that a level 3 software house (or labeled as ISO 9000) could guarantee a good product.

2.2 Some Facts: Feedback and Emergent Patterns

One thing that has to be made clear here is that software is not a unique type of engineered object, but that there are many types of software. Thus Lehman [10] establishes two broad categories:
a) S-type software is required to be correct in the full mathematical sense with respect to a fixed specification;
b) E-type programs model real world applications and are frequently subjected to system evolution. The development of these systems constitutes a complex feedback learning system. Data collected in projects such as the IBM OS-360 shows trends in evolution that support the existence of feedback and evolution in the software process.

Also, confronting the idea of a pure technological view we can hear categorical phrases such as "*Software development is a predominantly social activity*" [4]. Cain and Coplien take a sociological view of the software process and empirically study the underlying (self–organizing) structures that are created among the members of the organizations of software development.

Several of their discoveries are worth noting here:
a) there was a strong focus on process documentation in the organization, but what was documented was often distant from what was seen in actual practice;
b) in one case 80% of the developers were working on different processes from the official one, because the project's process standard did not capture the stable structure of the process;
c) communication is a key element for obtaining good organizational behavior. Highly productive organizations are cohesive; but more importantly: "Good organizations nurture roles that work directly with product artifacts".

They suggest that the patterns generated within the organization can be used for improving the software process. Another author, King [9], uses CASE tools to support systems emergence in information systems development, differentiating it from software engineering development. The link that is still missing is the relationship of the organizational structures with the series of intermediate structures that they are creating through the development process.

In this section we have observed two important views of software development: the first one embodies the characteristics of complex feedback systems and the second one highlights the communication aspects of an organization. The next section introduces a paradigm that can easily accomodate these facts.

2.3 Design: Moving away from the Technological Paradigm

In a recent book Blum [2] addresses the software construction problem as an ecological design problem. Blum does not deal with software as a mechanic artifact, but as a product of the design process, the design resultant of an activity. But the design process he proposes is not a technological design, but an *adaptive design*, wherein the design process takes into account changes in the requirements. Design is not a phase of the development of a software system, but the process itself.

The metaphor that Blum proposes is the 'sculpting process', in such a way that the sculptor interacts with the materials, adjusting the goals while the process goes on. The modifications of a part of the design extend inmediately to the other design objects influenced. He presents his evaluations of the ecological design paradigm as a way of moving away from the machine paradigm, but does not assume that it is the unique solution.

It is the concept of *design* that can accommodate the facts stated in section §2.2 and that links to the concepts of the following section §3. Design is a synonym of form, and a design is a transition from concepts and ideas to concrete descriptions [3]. But as Braha and Maimon state, "the concept of form is elusive, abstract, and complex". However, the design process can be characterised and situated within a paradigm. The five major paradigms of design are: analysis–synthesis–evaluation, case–based, cognitive, algorithmic and artificial intelligence. So, human design activities are developed within these frameworks. Design is also an ubiquitous process in nature as we state in the next section.

3 A Metaphor for the Software Process: Morphogenesis

Nature can provide us with examples of what are the results of effective processes that guarantee good products. Looking at the biological area for models we can initially pay attention to insect colonies. Ant colonies, for instance, are able to make some effective and reliable constructions. From the point of view of the process, nature has positioned those systems at the higher level of the scale, but that does not say much about the organization of the elements. The plan and the product is achieved with only a very effective communication among the agents.

But, if we focus on the aspects of design, the model to follow is that of *morphogenesis*. Morphogenesis in nature is also design. The metaphor of morphogenesis is taken here as a framework that allows us to view the process of software development as form generation. This is one of the biological similes that is more appropriate for comparison with software development.

Form is one of the properties that organisms show [7]. The word *phenotype* represents what organisms reveal to the senses and, according to Rosen [14], it is a uniquely biological concept. The generation of a phenotype starts from an initial set of genetic (explicit, informational) instructions that only specify the construction of low level building blocks. The interactions (implicit, dynamical) of those blocks unfold the next organizational levels in the system. As result of the internal interactions among the processes and with the environment a form emerges.

The distinction between phenotype and genotype resembles in some ways the distinctions between specifications and final code in software systems (E-type or real world systems). If in biology little can be said of the genotype by the study of the phenotype, in software systems it is very questionable that it would be possible to recover exactly the original specifications from the final product. More difficult still is the issue with the software process, since the product -code- does not entail much knowledge of the process used to built it.

Taking this perspective, the final product of the software process can be seen as an emergent structure. This emergent outcome is the result of a group of dynamic processes that are induced by a set of initial specifications. These specifications, in turn, are handled by the next level of procedures. Nature through evolution discards those forms that are not adapted to the environment. But there are also other reasons for limiting the variety of forms, like structural or "architectural coherence". If we take this analogy, then the evaluation of an organization not only has to take into account how the process is implemented, but also what the software products (final and intermediate) it manufactures are like. This can help us to understand why software houses that could be rated as inferior from the normative assessments can still survive in the market.

3.1 Emergence of Software Systems

The biological concept of *emergent property* is important since it can be made operational by using some formalized theories. There are several approaches to emergence as explained by Emmeche et al. [6], but almost all of them share the general definition of "creation of new properties", regardless of the substance involved. With the inclusion of the concept of *levels*, more power is introduced in the description of phenomena. The elements of a level give shape to the entities of the emerging level. Several constraining conditions act on the low level to form the next level. Different levels and entities can arise as the combination of primary and secondary emergence. This idea of emergence among levels can be applied to multiple fields.

The concept of emergence that is useful here is the one named *deductional emergence* defined by Baas [1]. Emergence is the creation of new properties in a structure that is the result of the interactions among the agents involved. The conceptual element that allows emergence to occur is the concept of *hyperstructure* which is a multi-level emergent structure. Hyperstructures apart from being the organizative and design principle, allow to incorporate the concept of *evolution*. A hyperstructure is a cumulative structure, not necessarily recursive. Hyperstructures are higher order structures (structures of order n), that are the representation of complexity and of complex designs. Interactions among structures can take the form of communication or information processing.

Baas establishes a basic distinction between deducible emergence and observational emergence. In the first case there is a deductional process that allows us to deduce the new properties. This type of emergence is applied to engineering constructions and to some nonlinear systems and it seems adequate for representing software development. Observational emergence, on the contrary, can not be deduced. The causes for emergence are attributed to two basic circumstances: 1) the existence or nonlinear interactions and 2) the existence of a large number of structures. The second cause is ubiquitous in software, and the first is the object of the work of [10]. So, as occurs in morphogenesis, emergence implies not only upward processes (from the bottom level to the upper), but also downward ones.

It seems that hyperstructures could be the way to represent the organization of hierarchies in various fields. It is argued also that the formation principle for the hyperstructures is the nonlinear interaction among structures, objects and systems, with emergence of new properties at each level. Hyperstructures allow the combination of the processes of evolution, adaptation and recursion. *Category theory* is the vehicle for manipulating those structures. Category theory has already been used in computer science for manipulating some object constructs, and it is a valid tool for analyzing transformational processes.

The previous paragraphs suggest that in the creation of an artifact several distinct types of processes concur, and the interaction among them leads as result to the generation of new properties in the resultant product. It is not difficult to envisage that in the creation of software products a combination of multiple types of processes cooperating until the final product is built. Many unknown relationships are established among the intermediate products and the designers of the software system that are invisible to the normative process (if existing).

3.2 The Human Element as the Substratum for the Creation of Processes

The substratum that allows the creation of E-type software is a human organization. So, the relationships between the groups and the intermediate structures they create should be made explicit in order to analyze the software process and its improvement. The main problem is that the view that each developer has of the system at each time step is not represented or documented, so that the traceability of the combination of representations is laborious. Only part of the emergent structures are recognized externally.

But it is possible to have less "mechanical" software processes. Zhuge et al. [17] have proposed a software process that is a special case of the general human problem-solving process. This perspective involves using analogy and abstraction as essential components of software process models. Curiously, these authors use the category theory as the representation for the objects and its operators within the software process. The cognitive-based problem-solving model is an example that deviates from other classical models and includes the possibility of constructing designs in a cognitive space. This clearly assumes (although not explicitly stated) that emergence of intermediate representations is a mechanism that allows the creation of an engineered product.

These cognitive-based problem-solving models have also other advantages, such as the ability to improve the software process by enhancing the group and developer's cognition. The inclusion of the cognitive-space for the group of developers, is an example of how a mechanism can be developed for group learning which bypasses some other structured processes for developing software. This supports the hypothesis that it is possible to develop software without having a "defined" software process, since abstraction and analogy are working at the cognitive level, interacting with the intermediate products that are being refined until the final solution is reached.

4 Conclusion

The thread of this paper is that there are some facts which speak against the consideration of software development as a purely mechanical matter. Thus, assessment from that point of view may be flawed. The concept of design has a biological interpretation that is the process of morphogenesis. This process is characterized by the creation of emergent structures, and emergence can be theoretically formalized by using hyperstructures. The construction of E-type software systems can be seen as the generation of forms by social bodies, so processes of adaptation, evolution and recursion are pertinent to the evaluation of those systems. More research is needed for the formalization of the software processes using, for instance, category theory.

Some of the recommendations of the technological paradigm that are suggested for improving the software process may achieve progress in the technological part of the process. But as products and human resources are so intertwined, establishing a comparison with a pure mechanical paradigm is difficult to maintain. Some well-known software companies do not follow that path [5]. Having strictly defined processes may go against the self-organization processes that help to improve those very processes. Some specific uses of models such as those mentioned in §2.1 can convey this view, although other interpretations may overcome the limitations of the technological approach. The evaluation of the processes should be made taking final and intermediate products into account, while considering the emergent intermediate system representations that the set of developers have when making decisions.

Finally, there remains the question of what is the exact relationship between process and product. This depends on the paradigm that is used for software developement. In fact, there is a way of building programs –genetic programming– in which the final product does not tell anything about the steps of the process (survival of the fittest). Thus, taking an organic perspective, many of the concepts of the complex system area could be useful for software development.

References

[1] Baas, N.A., "Emergence, Hierarchies and Hyperstructures", *Artificial Life III* (C.G. Langton ed.), Addison-Wesley (1994), 515–537.

[2] Blum, B., *Beyond Programming: To a New Era of Design*, Oxford University Press (1995).

[3] Braha, D., and O. Maimon, " The Design Process: Properties, Paradigms and Structure", *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans* **27(2)** (1997), 146–167.

[4] Cain, B.G., and J.O. Coplien, "Social patterns in productive software development organizations", *Annals of Software Engineering* **2** (1996), 259–286.

[5] Cusumano, Michael A., and Richard W. Selby, "How Microsoft Builds Software", *Comm. of the ACM* **40 (6)** (1997), 53–62.

[6] Emmeche, C., S. Koppe, and F. Stjernfelt, *Emergence and the Ontology of Levels. In Search of the Unexplainable*, Arbejdspapir 11, ISSN 0908-0589, Kobenhavn, (1993).

[7] Goodwin, Brian, *How the leopard changed its spots* , Phoenix Giant (1994).

[8] Herbsleb, James, et al., "Software Quality and the Capability Maturity Model", *Comm. of the ACM* **40, 6** (1997), 30–40.

[9] King, S., "Tool support for systems emergence: a multimedia CASE tool", *Information and Software Technology* **39** (1997), 323–330.

[10] Lehman, M.M., "Feedback in the Software Evolution Process", *Information and Software Technology* **38** (1996), 681–686.

[11] Lumley, T., "Complexity and the 'Learning Organization", *Complexity* **2 (5)** (1997), 14–22.

[12] Paulk, M., "How ISO 9001 Compares with the CMM", *IEEE Software* (January, 1995), 74–83.

[13] Racoon, L.B.S., "The Chaos Model and the Chaos Life Cycle", *ACM Software Engineering Notes* **20 5** (1995), 40–47.

[14] Rosen, Robert, "Mind as Phenotype", *Matter Matters? On the Material Basis of the Cognitive Activity of Mind* (P. Arhem, H. Liljenström, and U. Svedin eds.), Springer (1997), 39–56.

[15] Sanders, J., "Product, Not Process: A Parable ", *IEEE Software* (March, 1997), 6–8.

[16] Software Engineering Institute, *The Capability Maturity model. Guidelines for Improving the Software Process*, Addison-Wesley (1995).

[17] Zhuge, Hai, Jian Ma, and Xiaoqing Shi, "Abstraction and analogy in cognitive space: A software process model", *Information and Software Technology* **39** (1997), 463–468.