Rikkila et al., J Comput Eng Inf Technol 2012, 1:1
http://dx.doi.org/10.4172/jceit.1000e103

Journal of Computer Engineering & Information Technology

Editorial

A SCITECHNOL JOURNAL

The Implications of a Complexity Perspective for Software Engineering Practice and Research

Juha Rikkila[1], Pekka Abrahamsson[1] and Xiaofeng Wang[1*]

[1]Studios for Future Software, Free University of Bozen-Bolzano, Bolzano, Italy

Large software companies that enjoyed success in the past find themselves in increasing difficulties in today's turbulent business environments. Various issues they need to tackle include delays in releasing, feature bloating, slow or no response to changing, often individual, customer needs, and so on. Meanwhile, there are a growing number of small start-ups and "second-product" companies struggling to survive and thrive in increasingly competitive market places, too. For them an accurate definition of future is impossible to achieve. They have to live in the current moment, yet strive towards their continuously evolving vision. This new landscape of software engineering has brought serious challenges to the old approaches of managing software business. It calls for a different perspective on it and new approaches underpinned by such a perspective. In accordance with the new approaches, new research topics emerge which require appropriate research methodologies to make better sense of them.

Criticisms of the old (and not so old) approaches

Much of the past software business success can be contributed to well-planned strategies, well-defined long and short-term goals, well-defined program and project plans, and the prudent execution of them. Similarly, the hierarchical structures of authority for decision-making and for guiding the execution were well established in most organizations [1]. However, Koskela and Howell (2002) argue that the basic logic of software project is incorrect, based on a steam engine like adjustment paradigm. Reinertsen (2009) calls the traditional product development approach "not just little wrong, but wrong to its very core". The old approaches have been built upon Scientific Management [2], which details the management decisions and rules of work and leaves very little room for individual creativity or variation. Individual performance was accurately measured with predefined attributes. People were considered to be rational and become motivated and efficient by getting sufficient financial rewards. Even though the research in organizational behaviors and psychology has advanced hugely from the scientific management era, the old attitude is still strong in many software companies. However, the old approaches to manage software business became ineffective for large companies in turbulent business environments. And it is obvious that they would stifle the startup initiatives and entrepreneur spirits.

***Corresponding author:** Dr. Xiaofeng Wang, Studios for Future Software, Free University of Bozen-Bolzano, Bolzano, Italy, E-mail: xiaofeng.wang@unibz.it

Received: July 24, 2012 **Accepted:** July 26, 2012 **Published:** July 31, 2012

Agile software development movement appeared as a remedy to the old approaches. The emphasis of quickly responding to change and the focus on people's capabilities to achieve it are at the very core of agile methods [3]. After some limited success and rather long overall resistance, various agile practices started to gain ground. In many software organizations the most notable pressure of change has come with the desire to implement an agile approach to development. Initially a team level practice, agile methods have grown to attract larger organizations, and consequently put pressure on them to align management structures and practices with the agile methods they adopt. However, agile initiatives remained a developer and development team level change in spite of the efforts to scale them to organizational wide endeavors. Most of the management structures and practices remained the same in large software companies (Rikkilä 2012). On the other hand, start-ups have often felt that even though agile approaches are valid for their development effort, they fall short when dealing with customers in business terms, or creating products for common market needs. Business and product management agility are not unknown as terms, but often considered the practice of large organizations and corresponding to their needs. Start-ups need typically to deal with much shorter cycles in a much more customer responsive manner [4].

The perspective of Complexity

To understand why the old (and not so old) approaches are not sufficient in the new landscape of software engineering, we need to revisit what we have believed of being true without questioning in the past. To do so, we take on a perspective that shows promises: the Complexity theories. Originated in natural sciences such as physics and biology, the Complexity theories have been increasingly adopted in social sciences and the studies of human systems. We start with an overall picture of Complexity offered by the Cynefin model, as shown in Figure 1.

The Cynefin model separates ordered and unordered domains. An ordered domain is the one where cause and effect relationships are known or at least knowable after analysis. Generally a reductionist approach, that is, breaking a larger whole into pieces, solving the individual pieces and then summing up the results for the whole, is an essential analytical approach in an ordered domain. Causality,

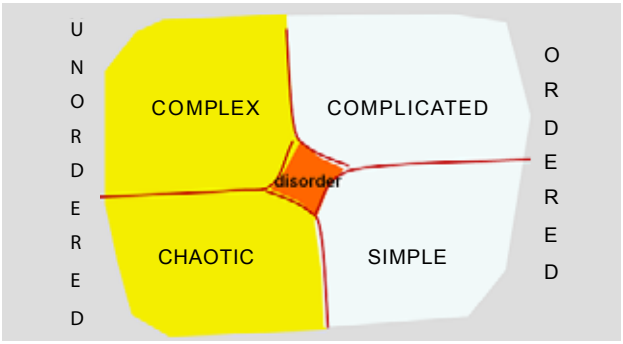

Figure 1: The Cynefin Model (adapted from Snowden & Boone, 2007).

reductionist thinking and predictability enable planning and control. In contrast, in an unordered domain neither causality nor linearity applies. This is the domain that interests us. The model makes a further distinction between complexity and chaos in an unordered domain. While chaos is completely random in behavior and without any expected consequence when acted upon, complex systems have properties that enable meaningful comprehension of and actions upon them. It is these properties and means of influencing that are the basis for radical new thinking of software development and its management in the new landscape.

There is no a unanimous definition of *complex system* or understanding of its properties. Drawing on different references, a condensed definition of a complex system can be an open system consisting of autonomous agents interacting with each other and with the environment. When human systems are concerned, human and non-human agents are heterogeneous, and often have the properties of a complex system themselves. Several properties have significant implications to the new approaches we are seeking.

The edge of chaos: A complex adaptive system is poised at the edge of chaos, where "the components of a system never quite lock into place, and yet never quite dissolve into turbulence, either" [5]. Stacey (2003) names it bounded instability, which means stable and unstable at the same time [6]. When a system operates in this dynamic it displays radical unpredictability over certain time spans and at certain levels of detail. Uncertainty is inevitable. The system shows patterns of behavior. The possibility space of the system's states in the short term can be depicted using fine details, but the path to which the system follows is uncertain and unpredictable in the long run. In addition, although the dynamic at the edge of chaos is required for novelty to emerge, it does not provide a guarantee of survival.

Self-organization: it is the ability of a system to evolve into an organized form without any external force. It is a natural result of nonlinear interaction, not any tendency of individual agents to prefer or seek order [7]. Generally, a self-organized system is dissipative, which means it needs energy to flow into and within it in order to move from one attractor to another. Self-organization only occurs in open systems that import energy from their environments *(Prigogine and Stengers 1985).*

Emergence: it is the appearance of a new feature, structure, or pattern of behavior at the system level which is not previously observed as a part of the system's functional characteristics, without any overall program or design, in a context that may be characterized by chance events. It is a collective phenomenon [6]. New structures, patterns, and properties emerge in a bottom-up way, from the interactions of lower level agents, but cannot be reduced to the characteristics of those individuals. Emergent phenomena seem to have a life of their own with their own rules, laws and possibilities [8]. Emergence is a source of variety.

Coevolution: the system is continuously adapting to its environment, and co-evolving with its environment by reconfiguring itself and changing its characteristics and behavior, creating new subsystems or agents and their configurations as well as new behavior. Sometimes the evolution is gradual, sometimes radical. Particularly system behavior and change is non-linear and not predictable, i.e., sensitive to initial condition and disproportionate to initial triggering effort and succeeding amplifications. On the other hand, a system can be very robust and path dependent so that the change is minor or non-existent even when great effort or energy is consumed for it [9].

From the perspective of Complexity, we argue that in most organizations software development is managed as in an ordered domain. The assumptions underlying the old approaches are that software and software organizations are complicated, therefore knowable, as long as we understand all the factors and causality links involved. However, an increasing amount of studies and empirical evidences demonstrate that what we tackle in software engineering (at least in recent time) are complex or chaotic phenomena, problems that are hard to define, hard to specify cause and effect relationships between them and the solutions. Furthermore, different stakeholders have different opinions on both problems and solutions. No solutions can be claimed complete, and problems keep on evolving. Any attempts to resolve them will reveal further problems. Problems vary over time and relate to other problems, without clear boundaries in between. Every problem is unique; therefore previous solutions will not apply to the next problem.

New approaches to software engineering practice

To operate effectively in a complex or even chaotic domain, different software practices and knowledge acquisition methods are in need. Some agile proponents have already referred to Complexity as the theoretical ground of agile methods when the *Agile Manifesto* was introduced in 2001. Yet an exhaustive interpretation of Complexity and the mapping to software development domain are still missing. Two initiatives are worth the attentions of software practitioners and researchers alike. One is the CALM (Complex, Agile, Lean Mesh-up) driven by Cognitive Edge (http://cognitive-edge.com/), and the other is the Stoos initiative driven by Stoos group (http://www.stoosnetwork.org/). Intensive activities from the two initiatives, together with the research on Complexity and its application in software development domain, create expectations on major advancement in this area.

Even though the new approaches are yet to take forms, we can specify a short list of requirements for them to be effective, drawing upon the insights from the Complexity theories:

- Enable effective operation even when there is a poor or no visibility to the future (efficiency is not measurable).
- Focus on stakeholders' value; the vision of value propositions is clear but evolves continuously.
- Constraints and boundaries for a solution can be set, but solution properties and drives are continuously evolving.
- Effective use of all available capabilities in organizations is enabled (limited capability to build "dream teams").
- Novelty and emergence are of high priority; predictability is secondary; failing is safe and even desirable when more knowledge is needed.
- Knowledge creation and consequently the adaptation of constraints and boundaries are continuous; traditional constraints such as architecture, security and safety requirements, user experience and the like need to be treated as continuously evolving.
- Steering of the solution development is done "at the edge", that is, setting operational constraints and boundaries, influencing attractors, creating and adjusting basis mechanisms (instead of in the middle with controlling requirements and development steps);
- Intensive internal communication is a must for emergence; transparency is vital for keeping direction and speed.

- Communication with the environment is a must to ensure continuous adaptation between stakeholders' needs and their implementation.

The list of requirements would serve as a guidance to evolve new approaches for software engineering practice. These approaches would be different from traditional, even agile, product and project management approaches. We are seeking "unproject" management approaches in which most of the traditional management principles would be reversed.

Implications to software engineering research

A Complexity perspective on software development and organizations also has implications to software engineering research in terms of both what we could study, as suggested by the new approaches, and how we should study them. Till now the focus of science has largely been on the problems that are "known" or "knowable", and the reductionist approach is effective. When faced with problems of uncertainty, complexity or chaos where the problems are "unknown" or "unknowable", how could we possibly research on them?

Among several potential challenges raised by Complexity, two has most significant even profound influence on how we conduct research. The first is the rethinking of causality. According to *Kurtz and Snowden,* in a complex domain where large software organizations reside, cause and effect relationships are only coherent in retrospect and do not repeat themselves, therefore no precise prediction can be made. We can study how patterns emerge through the interaction of agents, but the non-linear interactions "defy categorization or analytic techniques". We can perceive, but not predict, emergent patterns with retrospective coherence only, and we cannot be sure that they will repeat themselves because "the underlying sources of the patterns are not open to inspection (and observation of the system may itself disrupt the patterns)" [10]. In a chaotic domain where most startups strive to survive, there are even no perceivable cause and effect relationships. "There is nothing to analyze".

As a consequence, the ultimate goals of software engineering research (on complex phenomena at least) are not the searching for causalities, striving for predictions and increasing the generalizability of research findings. Rather the focus should be on the understanding of software and software development organizations in their local contexts. The research should target at detecting meaningful patterns without assuming that the same patterns would repeat themselves in similar complex systems. Therefore generalization is neither possible nor relevant. Local insights, local solutions, and local innovations are acceptable and even desirable.

Another Complexity challenge is the changing role played by researchers when studying complex or chaotic phenomena. The perspective of Complexity suggests that one can never achieve a complete view of a complex system. All views are partial. What's more, a researcher is never an outsider of a complex system. Through the act of researching, he becomes a part, another agent, of the system who interacts locally with other agents, and what would emerge from these local interactions are unpredictable. Kurtz and Snowden believe that without action a researcher would never possibly start to make sense of the researched phenomena. Therefore, the line between researcher and "researched" is very much blurred. It seems counterproductive to regard the researcher as an objective observer or "expert", and the researched simply the "targets" or "subjects".

As a result, a researcher needs to take a more humble and realistic stance towards his research, and more participative and action oriented research methods are necessary to achieve joint learning of researchers and software organization involved. Among various empirical software engineering research methods, action research seems particularly an appropriate one. It is "an iterative process involving researchers and practitioners acting together on a particular cycle of activities, including problem diagnosis, action intervention, and reflective learning" [11]. Phelps and Hase demonstrate that there are theoretical and methodological connections between complexity and action research. They advocate that action research "may be an appropriate and powerful vessel" [12] to conduct research where complexity is the paradigm. Actually action research has been under-deployed in software engineering research and more action research is called for [13]. We concur with this call and believe we saw a good reason for it.

To conclude our article, "sometimes, in order to see the future, it is necessary to rewrite the past"[1]. The critique presented against the contemporary management and development practices in software industry helped us to set focus on rising fringe areas of software business where different rules apply. Markets do not seem to stand still, which leads to the investigation of the "unordered" behavior of complex systems and how it has become applicable for managing software development as well. Consistent and proven models do not exist yet, and more research is needed with research methods suitable for studying complex systems.

References

1. Cooper RG, Edgett SJ, Kleinschmidt EJ (2002) Portfolio Management for New Products. (2nd edn).
2. Taylor FW (1911) The Principles of Scientific Management. Harper & Brothers, New York, NY, USA.
3. Highsmith J (2002) Agile Software Development Ecosystems. (1st edn) Addison-Wesley Professional.
4. Ries E (2011) The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business, USA.
5. Waldrop MM (1994) Complexity: The Emerging Science at the Edge of Order and Chaos. Penguin books, London.
6. Stacey RD (2003) Strategic Management and Organisational Dynamics: The Challenge of Complexity. (4th edn) Financial Times, Prentice Hall.
7. Fontana W, Ballati S (1999) Complexity. Complexity 4: 14-16.
8. Choi TY, Dooley KJ, Rungtusanatham M (2001) Supply Networks and Complex Adaptive Systems: Control versus Emergence. J Oper Manag 19: 351-366.
9. Volberda HW, Lewin AY (2003) Co-evolutionary Dynamics Within and Between Firms: From Evolution to Co-evolution. J Manage Stud 40: 2111–2136.
10. Kurtz CF, Snowden DJ (2003) The new dynamics of strategy: Sense-making in a complex and complicated world. IBM Syst J 42: 462-483.
11. Avison D, Lau F, Myers MD, Nielsen PA (1999) Action Research. Communications of the ACM 42: 94-97.
12. Phelps R, Hase S (2002) Complexity and action research: exploring the theoretical and methodological connections. Educational Action Research 10: 507-524.
13. Sjoberg DIK, Dyba T, Jorgensen M (2007) The Future of Empirical Methods in Software Engineering Research. Future of Software Engineering 358-378.

[1]Adapted from the HBR blog network article "If You Don't Like Your Future, Rewrite Your Past" by Rosabeth Moss Kanter on 12.6.2012.