Complexity Science for Evolving Software Ecosystems

T. V. Gopal
Department of Computer Science and Engineering
College of Engineering
Anna University
Chennai - 600 025, India

e-mail: gopal@annauniv.edu

***Abstract*—**The innovation necessary to create economic growth, drive societal change and address challenges related to profitable growth relies on technologies that are software – centric. The competitive environment and market dynamics are totally different. This evolving software ecosystem is still struggling with the growing pains that the current economic conditions present both as a catalyst for change and an opportunity to mature. Software ecosystem is typically a set of projects and products that co-evolve within the same organizational, social and technical concerns. Economic principles govern the choices more than the technology. Promoting Human Talent in Software, Creating Innovative Capacity and Shaping the future Internet and Mobile platforms are the core challenges in seizing the emerging opportunities. The production of systems with specific demands on Reliability, Availability, Maintenance, and Performance [RAMP] is one of the greatest challenges of software engineering at all phases of the development cycle. RAMP requirements for the ecosystem are left unspecified, specified at a later stage, or at best vaguely specified. Also, often times either it is difficult or prohibitively expensive to test for some of the RAMP specifications such as maintainability, reliability, and high availability. The difficulties multiply rapidly due to the absence of a clear set of rules, design principle or practices for the RAMP specifications. Acceptable Software Systems are not adequate. Even wrong Software Works. The concepts of **complexity** and **chaos** are becoming quite frequent in the evolving software ecosystems. This paper positions the emerging **Complexity Science** as a viable method.

***Keywords*—**Complexity Science, Cybernetics, Dynamical Systems, Software Aesthetics, Systems Thinking, Web Intelligence

I. INTRODUCTION

"We have seen the enemy and he is us" aptly summarizes the deliberations at the 1968 NATO Conference on Software Engineering.

Some of the basic lessons learnt are given below:

1. The hardware environment of a software system is not a constraint, but rather a primary driving force of software architecture and design.
2. At a high level, every software system can be modeled with four types of design elements: data structures and primitive operations, external (hardware) interfaces, system algorithms, and data flow and sequence of actions.
3. None of the design languages and modeling tools currently in use is adequate for developing and representing an entire software system.

"Engineering, medicine, business, architecture and painting are concerned not with the necessary but with the contingent - not with how things are but with how they might be - in short, with design."

- Herbert Simon, 1996

The lack of first principles of software development results in "the software crisis." The NATO Conference suggested some initial principles given below.

- Software artifacts are "machines"
  - Deterministic, cause and effect, formally describable
- People are potential "machines"
  - After installation of proper formalisms (education/training)
  - With appropriate management discipline
- Systems are Cartesian
- Model = machine
  - Possible to define a formal syntax and grammar capable of unambiguous description of the implemented machine.

Software began to be seen as Art, Science, Discipline and Psychology. Over the past five decades, Software Engineering which began with

the above principles ushered in many more concepts, principles, tools and techniques.

Some Characteristics of Large Software are given below.

- High degree of Uncertainty due to the dynamic nature of the parameters.
- Computing done by a Network of Computers, Sensors and other Gadgets with mobility whose behavior is difficult to predict.
- It is usually not practical to test the system under realistic conditions before deploying.
- Human intervention in debugging and modification while the Software is in use is prohibitively expensive.
- Stiff deadlines and stringent Quality of Service Parameters.
- There is a very high degree of heterogeneity.
- Compelling need to make assumptions, presumptions based on both "Imagination & Information".

The demand for increased variety of applications resulted in the industry examine new channels for supplying customers and new approaches to designing applications based on their core products. The dominant technology in many modern technical products is software. Software often provides the cohesiveness, control, and functionality that enable products to deliver solutions to customers. Software also provides the flexibility needed to workaround limitations or problems encountered when integrating other items into the system. Software is easy to change but extremely difficult to change correctly. The industry took to models based on collaboration between business competitors for developing Software with **Robustness, Productivity and Niche Creation as the crux.** Multi-stakeholders became imperative. Developing Large Software gave way to building services and a set of interacting components. Systems thinking is necessary for composing **Adaptive Software Systems**.

System can be **abstract or concrete**; **elementary or composite**; **linear or nonlinear**; **simple or complicated**; **complex or chaotic**. Complex systems are highly composite ones, built up from very large numbers of mutually interacting subsystems whose repeated interactions result in rich, collective behavior that feeds back into the behavior of the individual parts. Chaotic systems can have very few interacting subsystems, but they interact in such a way as to produce very intricate dynamics. Software systems consist primarily of a set of rules about behavior and also include the mechanism necessary to follow those rules as the system responds to states of the world. The production of such software systems with specific demands on Reliability, Availability, Maintenance, and Performance [RAMP] is one of the greatest challenges of software engineering at all phases of the development cycle.

A complex adaptive system is an ensemble of independent components that interact to create an ecosystem. The interactions are defined by the exchange of information and the actions of the components are based on some system of internal rules. These systems self-organize in nonlinear ways to produce emergent results that exhibit characteristics of both order and chaos. They evolve over time.

"Process" and "Frameworks for Quality" greatly enhanced the visibility of many core challenges. However, the domain knowledge remains a serious hurdle as software systems became pervasive all too soon.

1.1 Fringes of Science

In the philosophy of science, the question of where to properly draw a boundary between science and non-science, when the objective actually is objectivity, is called the demarcation problem. Compounding this issue is that proponents of some fringe theories use both proper scientific evidence and outlandish claims to support their arguments.

Software Development has been replete with the usage of Metaphors to connote the context. The following metaphors have found use in practice.

- **Diaphor** – poetry
  – "what if … was like …"
- **Epiphor** – prose
  – "an atom can be visualized as a small planetary system with electrons (planets) orbiting a central nucleus (sun), the orbits corresponding to …"

- **Lexical usage**
  – "Both the human brain and the electronic computer are instances of physical symbol systems, hence the brain IS a kind of computer."

- **Paraphor**
  – When a metaphor becomes a paradigm
    - Kuhn's notion of paradigm
    - Lexical use of metaphor
  – Metaphor becomes part of cultural perspective
  – Values associated with paraphor become the values of the culture
    - 'Rational' is good – emotional is bad
    - Control is essential – chaos is evil
    - Efficiency is a virtue – waste is a vice

The process based on these approaches does not manage bounds, directs, nudges and confines. It relies on emergent order rather than an imposed order. The components are not based on tasks and are highly interactive and unpredictable. Unfortunately, it needs a major error to bring out the functioning of the software. Unacceptable risks are not recognized. There are unnecessary components developed and some required one not developed. The success rate tends to be dismal.

Cyber – Physical Systems [1,2,3] is a quest for a consistently applied software systems engineering approach to build and deliver the "new order" of software-dependent systems based on Complexity Science.

II. COMPLEXITY

Complex is a special attribute given to many kinds of systems. It is used often, somewhat incorrectly, as a synonym of difficult. Difficult is an object which, with adequate computational power, can be predictable deterministically or stochastically. Complex is an object which is not predictable because of logical impossibility or because its predictability would require computational power far beyond any physical feasibility. Complexity, usually, is in reference to some observing system, it is subjective, and thus it is observed irreducible complexity.

Human systems are affected by several sources of complexity and may be put into three classes given below.

1. **Systems belonging to the first class are not predictable at all.** This class of systems has two types of complexity given below.
   a. **Logical Complexity:** directly deriving from self-reference, Gödel's incompleteness theorems
   b. **Relational Complexity:** resulting in a sort of indeterminacy principle occurring in social systems.

2. **Systems belonging to the second class are predictable only through an infinite computational capacity.** This class of systems has three types of complexity given below.
   a. **Gnosiological Complexity:** consists of the variety of possible perceptions
   b. **Semiotic Complexity:** represents the infinite possible interpretations of signs and facts
   c. **Chaotic Complexity:** characterizes phenomena of nonlinear dynamic systems.

3. **Systems belonging to the third class are predictable only through a trans-computational capacity.** Computational complexity that coincides with the mathematical concept of intractability belongs to this class. What appeared as complexity in the computer program was to a considerable extent complexity of the environment to which the program was seeking to adapt its behavior. Formal systems are merely complicated.

Presently only a limited complexity of systems in third class is studied and used in developing evolving software systems. The "essential" software engineering problems mentioned below have strained the domain knowledge of Computer Science and Engineering.

- **Changeability:** adapting themselves to unanticipated changes
- **Conformity:** working out everything the computer needs to "know"
  - Devoid of intuition, commonsense reasoning
- **Complexity:** integrating multiple already- complex programs
- **Invisibility:** communicating their likely behavior to humans

"Web Intelligence" and related technologies support the software systems through the Information Dependent functionalities such as:

- Integration of Change
- Unclear Goals and Objectives
- Internal and External Communications
- Knowledge Management
- Working as Teams
- Learning Technology
- Work Processes and Flows
- Customer Needs
- Developing Managerial Skills
- Managing Competition and Market Forces

This paper proposes Software Aesthetics and Cybernetics as two solutions for building the Cyber – Physical Systems founded on the Complexity Science.

III. SOFTWARE AESTHETICS

The Nature of Software [5] is as follows.

- Software is utilitarian.
- Software is mutable.
- Software is hidden.
- Software is beautiful.
- Software is complex.
- Software is insidious.
- Software is slavish.
- Software is logical.
- Software is abstract.
- Software is mediatory.
- Software is buggy.
- Software is transformable.
- Software is dictatorial.
- Software is pervasive.

There is no objective reality. All reality considered is socially constructed. Hence Software development is also a moral and social process within the ambit of an exploration of the constructed reality.

Aesthetics is an integral part of society, our interaction with hardware, and of software representation. Increasing pervasiveness of computing results in a corresponding concern for design aesthetics. All designs need strong aesthetic foundations, and a requirement to balance form and function. Software can be represented within the virtuality continuum facilitating effective, efficient and joyous interaction.

The goal of software aesthetics is not simply to create casual, ambient, peripheral, or otherwise fun objects. Instead, the goal is to explore the full range of interaction potential between software in the periphery and in-depth software analysis.

There is a basic difference between engineering problem solving and actually creating beyond what the problem calls for. It is interesting that people with design experience are able to see the nuances of what makes something appropriate in design. Software Aesthetics begins with the subjective or qualitative aspects such as:

- Source code that is more clearly organized
- More effective unit tests
- Better documentation
- More efficient
- More robust
- More usable user interface
- More attractive user interface
- More accessible
- More internationalized
- Presenting a unified whole
- Make users feel good
- Inspire confidence

These aspects need to be blended with the Software Metrics that are quantitative and specific to a chosen framework or a Maturity Model.

Complexity Science specifies aspects of aesthetics as:

- Cooperation
- Appropriate form
- System minimality - It is as small as it can be.
- Component singularity
- Functional locality
- Readability
- Simplicity

Formal proofs need an Unconventional Computing Model [6, 7].

IV. CYBERNETICS

Complexity Science warrants:

- Systems Thinking
- Creativity
- Open Systems
- Incrementalism
- Testing with External Intelligence
- Excellence
- Anticipating Needs
- Managing Resistance to Change
- Challenging Everything
- Risk Taking

A good formalism such as Cybernetics gives a space for complexity science to define the Cyber – Physical System in between the Comprising and Comfort Zone of Experimentation and Chaos or Anarchy. Unlike several engineering disciplines that have clear and well grounded rules specifications for experimentation, software engineering makes "Grounding Practice" as difficult as deciding on practice. Hence, formalism such as Cybernetics is necessary to outline pertinent verification and validation of evolving software ecosystems.

Cybernetics is an interdisciplinary approach for exploring regulatory systems - their structures, constraints, and possibilities **Cybernetics** includes the study of feedback, black boxes and derived concepts such as communication and control in living organisms, machines and organizations including self-organization. It is underpinned by the notion of **circularity** and feedback between a system and its environment.

Cybernetics is not merely a thrust for optimization on time and space. It is about the principle of circularity that factors the human paradox to a maximum possible extent.

The language of a deterministic world view applied to computing gradually changes to systemic view unpredictability of the interactions among the large number of components. Figure 1 [3] below is the basis for the evolving software ecosystem that functions on the emerging complexity science.

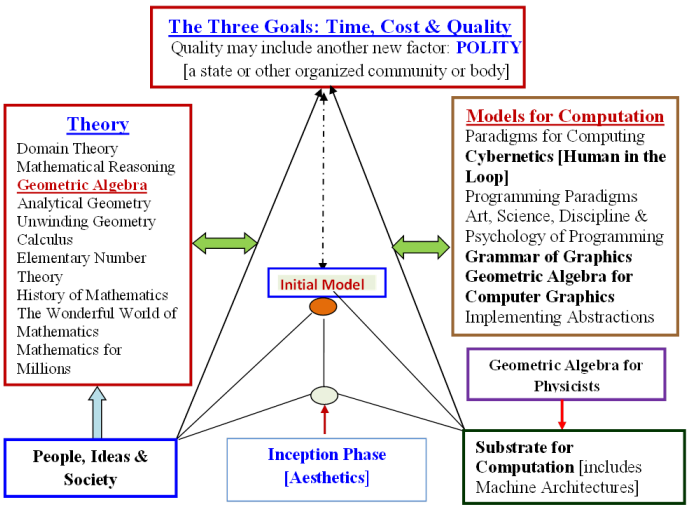

Fig 1. Proposed Block Schematic for Software Behavior in Cyber – Physical Systems

V. CONCLUSIONS

Chaos theory seeks an understanding of simple systems that may change in a sudden, unexpected, or irregular way. **Complexity** theory focuses on **complex** systems involving numerous interacting parts, which often give rise to unexpected order. Complexity Science for evolving Software Ecosystems is based on:

- Interconnected and interdependent elements and dimensions
- Feedback processes promote and inhibit change within systems
- System characteristics and behaviors emerge from simple rules of interaction
- Nonlinearity
- Sensitivity to initial conditions
- Phase space – the 'space of the possible'

- Attractors, chaos and the 'edge of chaos'
- Adaptive agents
- Self-organization
- Co-evolution

This paper presents Software Aesthetics and Cybernetics as two solutions that make Complexity Science viable for evolving software ecosystems.

REFERENCES

1. **Gopal T V**, The Physics of Evolving Complex Software Systems, *International Journal of Engineering Issues* [IJEI], Vol. 2015, no. 1, pp. 33-39.
2. **Gopal T V**, Modeling Cyber - Physical Systems for Engineering Complex Software, *International Journal of Engineering Issues* [IJEI], Vol. 2015, no. 2, pp. 73-78.
3. **Gopal T V**, Engineering Software Behavior in Cyber – Physical Systems, *International Journal of Engineering Issues* [IJEI], Vol. 2016, no. 1, pp. 44-52.
4. **Gopal T V,** Engineering Logic for Cyber – Physical Systems*, International Journal of Engineering Issues*, Vol. 2016, no. 3,pp. 112-120.
5. **Gopal T.V**. Beautiful code – circularity and anti-foundation axiom, *Int. J. Computational Systems Engineering,* Vol. 2, No. 3, 2016, pp.148–154.
6. **Gopal T.V.** Communicating and Negotiating Proof Events in the Cyber – Physical Systems, *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 7, Issue 3, March 2017, pp 236-242.
7. **Reuben Hersh** (Editor), *18 Unconventional Essays on the Nature of Mathematics*, Springer; 2006.