Position Papers of the 18th Conference on Computer Science and Intelligence Systems pp. 119–126
DOI: 10.15439/2023F2898
ISSN 2300-5963 ACSIS, Vol. 36

Towards a Definition of Complex Software System

Jan Žižka
0009-0007-6483-0037
Faculty of Informatics
Masaryk University
Brno, Czech Republic
Botanická 68a, Brno, 60200
Email: jzi@mail.muni.cz

Bruno Rossi
0000-0002-8659-1520
Faculty of Informatics
Masaryk University
Brno, Czech Republic
Botanická 68a, Brno, 60200
Email: brossi@mail.muni.cz

Tomáš Pitner
0000-0002-2933-2290
Faculty of Informatics
Masaryk University
Brno, Czech Republic
Botanická 68a, Brno, 60200
Email: pitner@muni.cz

***Abstract*—Complex Systems were identified and studied in different fields, such as physics, biology, and economics. These systems exhibit exciting properties such as self-organization, robust order, and emergence. In recent years, software systems displaying behaviors associated with Complex Systems are starting to appear, and these behaviors are showing previously unknown potential (e.g., GPT-based applications). Yet, there is no commonly shared definition of a Complex Software System that can serve as a key reference for academia to support research in the area. In this paper, we adopt the theory-to-research strategy to extract properties of Complex Systems from research in other fields, mapping them to software systems to create a formal definition of a Complex Software System. We support the evolution of the properties through future validation, and we provide examples of the application of the definition. Overall, the definition will allow for a more precise, consistent, and rigorous frame of reference for conducting scientific research on software systems.**

***Index Terms*—Software System, Complex System Theory, Complex Software System**

I. Introduction

COMPLEX Systems manifest multifaceted dependencies and interrelationships with other systems and environments, making them difficult, if not impossible, to model in their entirety [1], [2]. Complex Systems show properties that make them peculiar, such as the property of *emergent behavior*, i.e., behavior deriving from the different parts of a system that cannot easily be determined or forecasted when components are observed in isolation.

Complex Systems theory focuses on understanding and explaining the behavior of Complex Systems formed by interacting components [1]. The theory provides a general framework and a set of methodologies to study the emergent properties and dynamics embedded in Complex Systems. However, there is no agreed precise definition of the term as different authors might have other points of view [3]. Complex Systems have been studied in various fields [4]–[6], for example, in social sciences by exploring the complex interactions of individuals in cities.

The successes of software systems in the past years based on, for example, Neural Networks, such as systems developed by DeepMind [7], or OpenAI [8], bring a new range of software systems to wide attention. The appearance of applications such as AlphaFold or ChatGPT and others in a very short time suggests that many more such systems will rise soon.

The exciting behaviors of these new software systems, such as self-organization and emergence, cannot be explained by inspecting the software implementation they are based on. This range of software systems has specific behaviors correlating with the behaviors of Complex Systems as defined by the Complex Systems theory.

While in Computer Science, complexity is studied in different contexts, such as code complexity and the complexity of algorithms [9], this paper focuses on complexity in the context of *software systems*. Also, many software systems are *socio-technical* systems where humans are part of the system rather than only forming its environment. Our study is interested in pure technical systems where humans are not part of the system but may build the system's environment.

Our work aims to provide a clear definition of a Complex Software System (CSS) based on the *theory-to-research* strategy [10], [11], providing a frame of reference about the properties of such systems in relation to what is postulated by Complex Systems' theory [1].

As the field of Complex Systems is evolving [2], we suggest a framework that will also allow the evolution of the definition and terms. The precise definitions will allow more straightforward and unambiguous communication within academia and will be able to connect to existing and future real-world Complex Software Systems. The definition will also provide boundaries for new research fields with a degree of focus cleared of possible ambiguity due to the lack of definitions.

To summarize, we have the following contribution in this article:

- Setting up a framework under which the definition of a Complex Software System is created;
- Defining a Complex Software System based on reference to general Complex Systems theory;
- Listing examples of the use of such a definition;
- Based on the definition and proposed use, list potential future research directions;

The article is structured as follows. In Section II, we provide basic definitions that are commonly adopted in the context

Thematic track: Software Engineering for Cyber-Physical Systems

of software systems when discussing Complex Software Systems, such as System of Systems (SoS), Software Ecosystems (SECO), and Complex Adaptive Systems (CAS). The purpose of the need for a precise definition is discussed in Section III. Section IV discusses the method for creating the definition of a Complex Software System. We select several postulates in Section V to form an initial base for defining a Complex Software System. Section VI provides examples of using such a definition. Future research directions are discussed in Section VII, and conclusions are presented in Section VIII.

II. Basic Definitions

In Software Engineering, several commonly used terms and definitions of software systems discuss how software systems and components can be combined and aggregated. This section lists some of the main definitions and examines their relationship to Complex Software Systems.

System of Systems (SoS) is a collection of independent interacting systems [12]. An SoS has several key properties [13]:

- Operational Independence. Any system part of an SoS is self-standing and can operate even if the whole SoS is disaggregated.
- Managerial Independence. Every single system in an SoS is self-governing.
- Geographic Distribution. SoS are often distributed over geographic regions.
- Evolutionary Development. The existence and development of SoS are under constant change.
- Emergent Behaviour. *"Through the collaboration between the systems in an SoS, a synergism is reached in which the system behavior fulfills a purpose that cannot be achieved by, or attributed to, any of the individual systems."* [13]

The systems which are part of an SoS may also be Complex Systems, or the SoS as a whole may form a Complex System – however, the definition of an SoS does not imply that such a system is a Complex System.

Software Ecosystems (SECO) are *"defined as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with relationships among them"* [14]. A SECO may be composed of Complex Software Systems and is a type of SoS. SECOs are typically socio-technical systems [15], which exhibit Complex System behaviors. In a SECO, introducing new elements can potentially have disruptive effects. SECOs features [16], [17] for example contain and provide:

- Inherited characteristics of natural ecosystems like predation, parasitism, mutualism, commensalism, symbiosis, and amensalism.
- Architectural concepts like interface stability, evolution management, security, and reliability.
- Open source development model.
- Platform for negotiating requirements aligning needs with solutions, components, and portfolios.
- Capability for process innovation.
- Controlled central part for the core of the technology.

Complex Adaptive Systems (CAS) *"are systems that have a large number of components, often called agents, that interact and adapt or learn"* [18], [19]. The field of CAS focuses on the adaptive behavior of Complex Systems.

Software CAS refers to software systems that exhibit emergent behavior and self-organization, similar to Complex Adaptive Systems found in nature. These systems can adapt and evolve based on their interactions with the environment through feedback loops. They involve multiple interacting components or agents that collectively exhibit behavior that cannot be easily predicted from the behavior of individual components [18].

A software project may also be considered to be CAS, as suggested by [20].

A subset of Software CAS are Software Self-Adaptive Systems (SSAS) that focus on the ability of a software system to autonomously adapt and modify the behavior or configuration in response to changing conditions or requirements [21], [22]. SSAS have built-in mechanisms that monitor the system's state, analyze environmental changes, and take actions to maintain or improve system properties at runtime [21].

Software CAS [18], [19]:

- Typically operate far from equilibrium.
- Undergo revisions and improvements.
- Do not conform to classic, equilibrium-based mathematical approaches.
- Continuously adapt through recombination of the building blocks.

We summarize the main characteristics of SoS, SECO, and CAS in Table I.

III. Why there is a need for a definition of a Complex Software System?

We need the definition of a Complex Software System for several reasons. Below we discuss benefits, which are the motivators for the research presented in this article.

Clarity and precision: ensure that the meaning of the term Complex Software System is unambiguous.

Consistency: avoid that a software Complex System is defined differently by different researchers or in other contexts, and ensure that the term's meaning remains consistent over time.

Rigor: provide a framework for scientific research. Scientific definitions are necessary for the development of clear, precise, and consistent scientific concepts and for the advancement of scientific research.

Once the definition has been developed, it can be used in various ways. For example for:

Hypothesis testing: support the development of hypotheses about properties of a Complex Software System and its behaviors so that they can be tested through experiments and observations. For example, empirical methods can be used to verify if a software system fulfills the necessary conditions for forming a Complex Software System.

TABLE I
SoS, SECO, CAS Concepts

	System of Systems (SoS)	Software Ecosystems (SECO)	Complex Adaptive Systems (CAS)
Definition	Collection of independent interacting systems [12]	Collection of software components, applications, and services [14]	Collection of components (agents) that interact and evolve [18], [19]
Focus	Collection of interacting systems	Software and services relationships	Software agents interaction
Emergent Behavior	As systems get larger, emergent behavior is more probable [23]	Limited emergence, the introduction of new elements might have disrupting effects	Emergence in terms of adaptive behavior and self-organization
Examples	Smart Grids Systems	Android Ecosystem	Robotic Swarm

Classification: allow classification or categorization of Complex Software Systems based on their properties. For example, software systems use different paradigms, such as Neural Networks or multi-agent architectures. This can be used to create classifications based on system boundaries, technology, or the form of their implementation.

Comparison: a known Complex Software Systems can be compared based on the definition with other systems to find similar or differing properties. This can serve as grounds for expanding the definition or driving the creation of similar software systems.

Theory development: develop new theories or models.

Definitions are essential for academic research, providing a clear and precise framework for developing hypotheses, conducting experiments, and developing theories. Definitions allow researchers to communicate effectively and to build upon each other's work.

IV. Methodology to Build the Definition

To define a Complex Software System, we adopt the *theory-to-research* strategy, in which there is a continuous cycle between theory and empirical validation [10], [11]:

- we extract from the literature (e.g., [1], [2]) common properties of Complex Systems as have been studied in the different fields;
- as there is no full agreement on all properties in the context of the theory of Complex Systems [3], we discuss the most appropriate in the context of software systems, both according to our view of software systems implementation and deployment and with the aid of further related works [4], [6], [18], [24];
- we map each of the properties to a set of identified *necessary*, *sufficient* and *representative conditions* to the context of software systems;
- we provide an initial application of the definition to showcase the main benefits;

Ladyman [2] defines a Complex System based on reviewing attempts in the literature to characterize a Complex System and compiles a set of necessary conditions to represent complexity. In their work, authors provide conditions that are qualitative and which may not be sufficient for complexity, but they set a basis for a way how complexity can be defined. We suggest using the same method for defining a Complex Software System.

We base the method of writing a formal definition of a Complex Software System on a list of properties of three types that are *necessary*, *sufficient*, and *representative* conditions written in natural language.

The set of properties is the definition of a Complex Software System.

The properties will be assembled in the form of graphical frames followed by a commentary. Different types of properties will be color-coded for clarity in the following way, where *"n"* is an ordered number and *"keyword"* is a word abbreviating the property:

Property n - *"keyword"* - Necessary

A property that is a necessary condition but not sufficient to define a Complex Software System. Any Complex Software System must be fulfilling all properties that are necessary conditions. But fulfilling all such conditions doesn't imply that the Software System is Complex Software System.

Property n - *"keyword"* - Sufficient

A property that is the sufficient condition to define a Complex Software System. If a software system fulfils any single property that is sufficient condition then such a system is a Complex Software System.

Property n - *"keyword"* - Representative

A property describing a typical feature of a Complex System is a representative property. Such property is neither necessary nor sufficient but does describe a commonality among Complex Software Systems.

The nomenclature allows referring specific property such as Pn-N *"keyword"* for a necessary condition property, Pn-S *"keyword"* for sufficient condition property and Pn-R *"keyword"* for a representative property.

The numbering of properties is sequential across all the types. The intention and expectation is that the type of the property may change based on future validation and research and keeping the numbering intact will allow for unambiguous referencing.

V. Initial Definition of a Complex Software System

A Complex Software System may be defined by a set of properties which may be viewed as *necessary*, *sufficient*, and *representative* for a system to exhibit Complex System behaviors. Such properties are generic for any Complex System and are also described in existing publications such as [1], [2]. In this section, we will summarize the basic properties of Complex Systems and put those in the context of software systems, creating a base for the definition of a Complex Software System.

Property 1 - *"components"* - Necessary

A Complex Software System is composed of many components.

All definitions of systems complexity [1], [2], [4], [6] require the system to have many components. In the context of Complex Systems, the word *"many"* refers to the term's qualitative rather than quantitative nature. It would, therefore, be incorrect to attempt to quantify it. For example, a system composed of two Complex Systems with manifold interactions and fulfilling other necessary properties is a Complex System, as well as a system composed of millions of components of a similar type, maybe a Complex System. This property comes directly from the definition of the word "system" [25], [26]. Software systems are typically composed of components. This property is a necessary condition but not sufficient for a software system to exhibit complexity. In software, a component may describe different entities based on view or perspective. It can represent a code module, software package, process, or service. From the perspective of a Complex Software System, only a subset of such representations can serve as components in a Complex Software System as they must possess further attributes discussed in the following paragraphs.

Property 2 - *"communication"* Necessary

The components of a Complex Software System have means of intercommunication.

Communication is an essential condition for a Complex Software System. As Ladyman [2] explains: *"Without interaction, a system merely forms a "soup" of particles which necessarily are independent and have no means of forming patterns, of establishing order."*. Communication through messaging shared data, and interfaces is fundamental in software systems. However, this property is not a sufficient condition for a Complex Software System, as many software systems communicate but lack other necessary properties.

Property 3 - *"similarity"* - Representative

The components in a Complex Software System are similar.

Based on Ladyman [2]: *"For interactions to happen and for pattern and coherence to develop, the elements have to be not only many but also similar in nature."*. From the software systems perspective, for example, a system based on front-end, business logic middle-ware, and back-end database components may not form a Complex Software System. This has fascinating implications for software systems, which may be considered complex. However, this condition is not sufficient to determine a Complex Software System. This property may be necessary, but such a statement cannot be demonstrated and proven with the current knowledge and it is a question if a Software System with dis-similar components may still form a Complex Software System or if the system boundaries would exclude such dis-similar components into system's environment rather than being part of the system itself. It also remains to be defined what precisely *similar* means in the context of software components, and similarity needs to be inspected along with heterogeneity and homogeneity.

Property 4 - *"interaction-change"* - Necessary

The strength of components interactions in a Complex Software System is dynamic and changes over time.

"Most interactions are mediated through some sort of exchange process between nodes (components). In that sense, interaction strength is often related to the quantity of objects exchanged." [1]. The interactions among components have to change over time for a Complex Software System to evolve and create a self-organized clustered structure [1]. The resulting network topology contains information about the nodes' and links' dynamics and formation (Chapter 4.5) [1]. This is a familiar property in software systems studies, for example, in the field of dynamic or adaptive networks [27], [28]. This property is necessary for a Complex Software System from which self-organization and clustering emerge.

Property 5 - *"states"* - Necessary

Components of a Complex Software System are characterized by states.

Complex Software Systems are systems that evolve. An algorithmic description of evolution (Chapter 5) [1] is based on the fact that the system has states, and the evolution forms a path through states from time t to time $t+1$. Therefore the existence of states is necessary to create a Complex Software System. The notion of states in software systems is among the basic concepts of any information systems [29]. However, the existence of states is not a sufficient condition for Complex Software Systems.

Property 6 - *"co-evolution"* - Representative

The intercommunication and states of components in a Complex Software System are not independent but co-evolve.

As discussed in (Chapter 1.5) [2] *"Complex systems are characterized by the fact that states and interactions are often not independent but evolve together by mutually influencing each other; states and interactions co-evolve."*. The Complex Systems are characterized by co-evolutionary dynamics (Chapter 4.8) [1]. From a software systems perspective, this can be represented, for example, by adaptive network models [28], which are known to exhibit such co-evolutionary dynamics [30], [31]. The co-evolutionary algorithms may also be used to solve complex software problems [32].

Property 7 - *"context-awareness"* - Representative

A Complex Software System is context-aware.

As shown by Thurner [1], the Complex Systems are often represented by multi-layer networks and *"... for any dynamic process happening on a given network layer, the other layers represent the 'context' ..."*. In other words, such context defines how components on different layers may be influenced. This property is typical for Complex Systems to co-evolve through context dependency and awareness.

Property 8 - *"algorithmicism"* - Representative

A Complex Software System is algorithmic.

Based on Thurner [1], the *"... (Complex Systems) algorithmic nature is a direct consequence of the discrete interactions between interaction networks and states."*. This fits software systems that are naturally algorithmic [29].

The "algorithmicism" may need to be replaced with "intelligence" or "cognition" based on symbolic or sub-symbolic approaches [33], which might be more appropriate from a software systems perspective, when comparing this property to general Complex Systems theory.

Property 9 - *"path-dependency"* - Representative

Complex Software System processes are path-dependent and non-ergodic.

"The Complex Systems are typically governed by path-dependent processes." (Section 2.5) [1]. The process, in a general theory of complex systems, refers to stochastic processes [34]. This further means that processes in complex systems are inherently non-Markovian. It can also be shown that Complex Systems are non-ergodic (for in-depth discussion, see [1]). From the software systems perspective, this means that software system to exhibit such Complex System properties, they must change their boundary conditions as the system evolves.

Property 10 - *"disorder"* - Necessary

A Complex Software System is disordered and out-of-equilibrium.

Ladyman [2] argues that *"...complex systems are precisely those whose order emerges from a disorder rather than being built into them."*. Also, it can be noted that Complex Systems are generally out-of-equilibrium [1], which drives interesting challenges to the concepts of entropy. Although it can be shown [1], [2] that Complex Systems exhibit such properties, it is not obvious how to apply those to software systems.

It must be noted that disorder doesn't imply instability, which might sometimes be associated with the term. In the sense presented by the property, the disorder is related to entropy. For example, the use of GA (Genetic Algorithms) result is seemingly disordered systems when the system is inspected.

Property 11 - *"robust-order"* - Necessary

A Complex Software System exhibits robust order.

The concept of robust order is derived from system disorder. As shown by Ladyman [2] *"... a system consisting of many similar components (elements) which are communicating (interacting) in a disordered way has the potential of forming patterns or structures"*. This refers to self-organization and emergence property. From a software system perspective, this indicates that a Complex Software System shall be composed of similar and at least initially disordered components. This might seem to contradict with P10-N *"disorder"* but *"... although the elements continue to interact in a disordered way, the overall patterns and structures are preserved. A macroscopic level arises out of microscopic interaction, and it is stable"* [2] which Ladyman defines it as a robust order and continues that *"t(T)his kind of robust order is a further necessary condition for a system to be complex"*. Therefore disorder and robust order may co-exist. One example of software systems showing such a property are Artificial Neural Networks (ANN), where initially, input weights of neurons may be initialized with random values and, through learning, such initial disorder forms patterns, structures, or clusters. Also, when examined on a neuron level, ANN will still be disordered.

From a software systems perspective, it will be interesting to study also further run-time uncertainties concerning robust order property.

Property 12 - *"memory"* - Necessary

A Complex Software System has memory.

From Holland [18]: *"A system remembers through the persistence of internal structure"*, Ladyman [2] infer that *"Memory is a straightforward corollary of robust order."*. And Thurner [1] notes that the *"Complex systems often have memory. Information about the past can be stored in nodes (components), if they have a memory, or in the network structure of the various layers."* In such a sense, *memory* refers to the internal self-organized structure of the system. The difference between *memory* and *states* defined by P5-N *"states"* is that *states* represent the system at a specific point in time, but they do not represent history-dependent dynamics stored in the systems *memory*. This property might have various interpretations in software systems, such as a path through imitation-learning [35] or system audit trails. This interesting property might also have yet unknown interpretations in software systems.

Property 13 - *"SoS sufficiency"* - Sufficient

A System of Complex Software Systems is a Complex Software System.

As an intuitive analogy to properties of a Complex Software System – as in Ackoff [12] – it may be possible to show that a system of Complex Software Systems forms a Complex Software System.

This might have exciting implications in practice as once a Complex Software System is created and exists, a new Complex Software System may be formed by creating a system of such systems (SoS).

As the software does not require any material or physical manipulation and software systems can be created relatively quickly, this allows the possible rapid advancement of software-based systems exhibiting Complex System behaviors.

VI. Application of the Definition

A. Unambiguous communication within academia

"Complex Software System" is a widely used term in academia and industry. It refers to a wide range of software systems and viewpoints with a generic notion of a system's complexity. The definition presented in this paper attempts to provide a concrete reference that can be utilized throughout academic discussions to facilitate a common understanding of the term and properties of such a system. Also, the proposed definition framework is intended to extend and refine the definition to support further Complex Software Systems theory development.

The definition of a Complex Software System may be referenced as a whole, or specific properties may be the focus of empirical and other research when studying the properties of software systems. Having a definition of a Complex Software System will bring clarity through academic discussions.

B. Complex Software System categorization

Software systems are open systems [36] with external interactions. The boundary of the system defines what belongs to the system itself and what its surroundings are. The edge of the system may be used for categorization. Many software systems nowadays are socio-technical systems [37] where people are part of the system rather than creating the surroundings and interacting with the system only through the system boundary.

The software systems also interact with humans or are part of machine-to-machine interactions. The software system boundaries can be used as one of the aspects of categorizing types of software systems. Most importantly, this categorization has an essential perspective from Complex Software Systems theory. Most of the socio-technical systems are Complex Systems [37], and the involvement of humans fulfills the necessary conditions presented in Section V.

For example, if we consider the Internet as a Complex Software System, it can be viewed as a socio-technical system. In which case, it fulfills the P4-N "interaction-change" property. The changes are done by human developers, companies, and communities, which interconnect services throughout the Internet. If the boundary of the Internet as a system excludes human actors, the P4-N "interaction-change" might not hold.

The presented properties applied to different boundaries of a software systems can, in this way, provide mechanisms to create a categorization and demonstrate which boundary is allowing the creation of a Complex System and which is not, as they are not fulfilling the necessary conditions defined by the presented properties.

C. Complex Software System modeling

To dive into understanding Complex Software Systems, it will be required to have a model to analyze the properties' effects, how the *necessary* and *sufficient* conditions may be fulfilled or violated, and how *representative* properties may help define a Complex Software System. This can be achieved, for example, by studying existing Complex Systems, as it is done in other fields. However, the challenge is that we might not have access to such systems and, based on the boundary categorization (Section VI-B), some categories of Complex Software Systems might not even exist, for example, pure technical Complex Software Systems, where humans are outside the system boundary. The categorization based on modeling may follow, for example, FTG+PM framework [38], which aims at the categorization of complex cyber-physical systems.

The model can be designed and developed to study Complex Software Systems based on the presented definition and specification of necessary conditions for such a system to exist. Commonality and variability analysis will be required to create such models.

The model will allow experiments to evaluate the assumptions placed by the properties of Complex Software Systems. Understanding underlying principles might show how such software systems may be constructed. The models may also be utilized directly during the process of creation of a complex software system, as, for example, suggested by [39] with SDD (Simulation Driven Development) to tackle inherent system complexity. Modeling ASA (Adaptive Software Architectures)

[40] might be another way to direct the creation of Complex Software Systems.

With models based on the defined Complex System properties, it may be, for example, demonstrated that P13-S "SoS sufficiency" is a *sufficient* condition.

Creating models of different categories of Complex Software Systems is another research path we will follow.

VII. Future research

Creation of the definition of a Complex Software System and a framework for describing the definition will open doors for several research areas:

- Search for and refining Complex Software System properties;
- Exploring categories of Complex Software Systems;
- Creation of Complex Software System models;
- Search for underlying principles of Complex Software Systems;

The framework discussed in Section IV provides means of extending and refining the definition based on future research in the field of Complex Software Systems. The properties may be updated or expanded as new information becomes available. This will require empirical validation of hypotheses and possibly rejecting null hypotheses posed by the definition. The validation is expected to trigger a refinement cycle for the theory, as defined by theory-to-research strategy [10]. The validation may be based on case studies of existing software systems or experimental research based on simulations and modeling.

The list of the initial 13 properties harvested from studies of Complex Systems in other fields may not always have a precise mapping to software systems. Some properties that define necessary conditions will require further research to understand what they can indicate in the context of software systems. Especially P7-R "context-awareness", P9-R "path-dependency", P10-N "disorder", P12-N "memory" or P13-S "SoS sufficiency".

The categorization, discussed in Section VI-B, based on the definition, might provide additional research topics while helping to uncover new underlying general principles of Complex Software Systems. Inspecting academic discussions, aided by systematic literature reviews, surveys, or questionnaires, will facilitate such categorization.

Our research aims at technical systems, excluding socio-technical systems. This constraint, however, might prove to be challenging to isolate, and it is clear that the cyber-physical and socio-technical systems will be encountered during further research where the discrete and continuous aspects of Software Systems co-exist. This will require clearly defining and distinguishing the system boundaries to tackle encountered involvement of non-technical aspects.

System Complexity is viewed as beneficial property of Software Systems exhibiting interesting properties, therefore it is not in the scope of our research to suggest means of easing or reducing the complexity.

Several Software Systems might be considered complex, such as modern operating systems with their constant struggle against cyber attacks. These systems might not fulfill the criteria for complexity based on the proposed definition. In future research, it will be required to categorize such systems and avoid potential confusion on the term as there are different perspectives through which complexity can be viewed.

The proposed definition suggests one sufficient condition P13-S "SoS sufficiency." In future research, we will identify other potential sufficient conditions to extend the definition. However, this task will be challenging.

VIII. Conclusions

Complex Systems were identified and studied in different fields, such as physics, biology, and economics. These systems exhibit properties such as self-organization, robust order, and emergence. In recent years, software systems started to display behaviors associated with Complex Systems, showing previously unknown potential (e.g., GPT-based applications). However, a commonly shared definition of a Complex Software System is not yet available.

For this reason, in this paper, we have presented a definition of a Complex Software System that can serve as a reference for academia to support future research. The definition is a set of 13 initial *necessary*, *representative*, and *sufficient* conditions for a software system to exhibit Complex System behaviors. The properties were selected from Complex Systems research in other fields and mapped to software systems. We suggested allowing for evolution and refinement of the properties, as the definition can be refined by evaluating the studied properties using empirical methods. We have also provided examples of the use of the definition and discussed further research directions in the area of Complex Software Systems.

An unambiguous definition of a Complex Software System is a stepping stone toward understanding its underlying principles.

Acknowledgement

The work was supported from ERDF/ESF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

References

[1] S. Thurner, R. Hanel, and P. Klimek, *Introduction to the theory of complex systems*. Oxford University Press, 2018.
[2] J. Ladyman, J. Lambert, and K. Wiesner, "What is a complex system?" *European Journal for Philosophy of Science*, vol. 3, no. 1, pp. 33–67, Jan 2013. doi: 10.1007/s13194-012-0056-8
[3] H. Ledford, "Language: Disputed definitions," *Nature*, vol. 455, no. 7216, pp. 1023–1028, Oct 2008. doi: 10.1038/4551023a
[4] M. Mitchell, *Complexity: A guided tour*. Oxford university press, 2009.
[5] G. J. Klir and H. A. Simon, *The architecture of complexity*. Boston, MA: Springer US, 1991, pp. 457–476.
[6] M. M. Waldrop, *Complexity: The emerging science at the edge of order and chaos*. Simon and Schuster, 1993.
[7] [Online]. Available: https://www.deepmind.com/
[8] [Online]. Available: https://openai.com/
[9] J. Van Leeuwen, *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Cambridge, MA, USA: Mit Press, 1991. ISBN 0444880712

[10] R. A. Swanson and T. J. Chermack, *Theory building in applied disciplines*. Berrett-Koehler Publishers, 2013.

[11] P. D. Reynolds, *Primer in theory construction: An A&B classics edition*. Routledge, 2015.

[12] R. L. Ackoff, "Towards a system of systems concepts," *Management science*, vol. 17, no. 11, pp. 661–671, 1971. doi: 10.1287/mnsc.17.11.661

[13] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: basic concepts, model-based techniques, and research directions," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, pp. 1–41, 2015. doi: 10.1145/2794381

[14] D. G. Messerschmitt, C. Szyperski *et al.*, *Software ecosystem: understanding an indispensable technology and industry*. MIT press Cambridge, 2003, vol. 1.

[15] T. Lima, R. P. dos Santos, and C. Werner, "A survey on socio-technical resources for software ecosystems," in *Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems*, 2015. doi: 10.1145/2857218.2857230 pp. 72–79.

[16] J. Joshua, D. Alao, S. Okolie, and O. Awodele, "Software ecosystem: Features, benefits and challenges," *International Journal of Advanced Computer Science and Applications*, vol. 4, no. 8, 2013. doi: 10.14569/IJACSA.2013.040833

[17] D. Lettner, F. Angerer, H. Prähofer, and P. Grünbacher, "A case study on software ecosystem characteristics in industrial automation software," in *Proceedings of the 2014 International Conference on Software and System Process*, ser. ICSSP 2014. New York, NY, USA: Association for Computing Machinery, 2014. doi: 10.1145/2600821.2600826. ISBN 9781450327541 pp. 40–49.

[18] J. H. Holland, "Complex adaptive systems," *Daedalus*, vol. 121, no. 1, pp. 17–30, 1992.

[19] ——, "Studying complex adaptive systems," *Journal of systems science and complexity*, vol. 19, pp. 1–8, 2006. doi: 10.1007/s11424-006-0001-z

[20] A. B. Myburgh, "Situational software engineering complex adaptive responses of software development teams," *2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014*, p. 841 – 850, 2014. doi: 10.15439/2014F196

[21] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Springer, 2013. doi: 10.1007/978-3-642-02161-9_1 pp. 1–32.

[22] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267–7279, 2013. doi: 10.1016/j.eswa.2013.07.033

[23] J. S. Osmundson, T. V. Huynh, and G. O. Langford, "Emergent behavior in systems of systems," in *INCOSE International Symposium*, vol. 18, no. 1. Wiley Online Library, 2008. doi: 10.1002/j.2334-5837.2008.tb00900.x pp. 1557–1568.

[24] J. M. Ottino, "Complex systems," *American Institute of Chemical Engineers. AIChE Journal*, vol. 49, no. 2, p. 292, 2003. doi: 10.1002/aic.690490202

[25] Merriam-Webster. System. [Online]. Available: https://www.merriam-webster.com/dictionary/system

[26] O. E. Dictionary. system, n. [Online]. Available: https://www.oed.com/view/Entry/196665

[27] F. Kuhn and R. Oshman, "Dynamic networks: models and algorithms," *ACM SIGACT News*, vol. 42, no. 1, pp. 82–96, 2011. doi: 10.1145/1959045.1959064

[28] A. H. Sayed, "Adaptive networks," *Proceedings of the IEEE*, vol. 102, no. 4, pp. 460–497, 2014. doi: 10.1109/JPROC.2014.2306253

[29] I. Sommerville, *Software Engineering*, ser. Always learning. Pearson, 2016. ISBN 9780133943030

[30] D. Antoniades and C. Dovrolis, "Co-evolutionary dynamics in social networks: A case study of twitter," *Computational Social Networks*, vol. 2, no. 1, pp. 1–21, 2015. doi: 10.1109/SITIS.2014.68

[31] M. Farajtabar, M. Gomez-Rodriguez, Y. Wang, S. Li, H. Zha, and L. Song, "Co-evolutionary dynamics of information diffusion and network structure," in *Proceedings of the 24th International Conference on World Wide Web*, 2015. doi: 10.1145/2740908.2744105 pp. 619–620.

[32] P. B. Myszkowski, M. Laszczyk, and D. Kalinowski, "Co-evolutionary algorithm solving multi-skill resource-constrained project scheduling problem," *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017*, p. 75 – 82, 2017. doi: 10.15439/2017F318

[33] E. Ilkou and M. Koutraki, "Symbolic vs sub-symbolic ai methods: Friends or enemies?" *CEUR Workshop Proceedings*, vol. 2699, 2020.

[34] S. M. Ross, *Stochastic processes*. John Wiley & Sons, 1995.

[35] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. doi: 10.1038/s41586-019-1724-z

[36] L. v. Bertalanffy, *General system theory: Foundations, development, applications*. G. Braziller, 1968.

[37] V. Tomic, "A bionic view on complex software systems-and the consequences for digital resilience," Master's thesis, Wien, 2021.

[38] S. Mustafiz, J. Denil, L. Lúcio, and H. Vangheluwe, "The ftg+pm framework for multi-paradigm modelling: An automotive case study," *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling, MPM 2012*, p. 13 – 18, 2012. doi: 10.1145/2508443.2508446

[39] T. Baumann, B. Pfitzinger, and T. Jestadt, "Simulation driven development-validation of requirements in the early design stages of complex systems-the example of the german toll system," *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017*, p. 1127 – 1134, 2017. doi: 10.15439/2017F133

[40] N.-T. Huynh, M.-T. Segarra, and A. Beugnard, "A development process based on variability modeling for building adaptive software architectures," *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016*, p. 1715 – 1718, 2016. doi: 10.15439/2016F170