J Syst Sci Complex (2020) 33: 1755–1784

A Review on Complex System Engineering

PARREND Pierre · COLLET Pierre

DOI: 10.1007/s11424-020-8275-0
Received: 19 September 2018 / Revised: 25 January 2019

Abstract Complexity is commonly summarized as 'the actions of the whole are more than the sum of the actions of the parts'. Understanding how the coherence emerges from these natural and artificial systems provides a radical shift in the process of thought, and brings huge promises for controlling and fostering this emergence. The authors define the term 'Complex System Engineering' to denote this approach, which aims at transferring the radical insights from Complex System Science to the pragmatic world of engineering, especially in the Computing System Engineering domain. A theoretical framework for Complex System Engineering is built by the morphogenetic engineering framework, which identifies a graduation of models, in growing order of generative power. The implementation of Complex System Engineering requires a portfolio of operational solutions: The authors therefore provide a classification of Complex System application approaches to answer this challenge and support the emergence of Complex System Engineers capable of addressing the issues of an ever more connected world.

Keywords Complex networks, complex software engineering, complex systems, design structure matrix, emergent engineering, evolutionary algorithms, organically grown architectures.

1 Introduction

Complexity first denoted the property of objects with many interconnected parts[1]. The term is now used to express the behaviour of systems which coherence is drawn from parts interacting locally, such as financial markets, rainforest ecosystems, living organisms, the Internet: Emergence. It is commonly summarized as 'the actions of the whole are more than the sum of the actions of the parts'. Understanding how coherence emerges from these natural and artificial systems provides a radical shift in the process of thought, and brings huge promises for controlling and fostering this emergence: In an ever more connected word, being able to control the dynamics of multi-scale systems is a critical skill for success.

However, as stated by Edgar Morin, complexity 'cannot be reduced to a simple idea', it 'cannot be reduced to a law of complexity'[2]. It is therefore hopeless to expect the expression

PARREND Pierre
ECAM Strasbourg-Europe, 2, *Rue de Madrid, Schiltigheim, France.* Email: pierre.parrend@ecam-strasbourg.eu.
COLLET Pierre
University of Strasbourg, Strasbourg, France. Email: pierre.collet@unistra.fr.
◇ *This paper was recommended for publication by Editor DI Zengru.*

of simple laws, or a restricted body of knowledge, of complexity. Our research effort in the application of complexity to engineering problems thus draws on the properties of Complex Systems, aims at identifying common methodological patterns, and strive to expose pragmatic solutions both from the scientific literature and in our own contributions.

In this paper, we highlight the research challenges we focus on in this work in Section 2. Subsection 2.2 explicits the way science and engineering interact and complement themselves. Section 3 presents the main contributions of the field of complex system engineering, and Section 4 introduces significant applications.

2 Research Challenges

2.1 Engineering and Complex Systems

The field of Complex System Engineering emerges from breakthroughs in the community of morphogenesis[3], embryogenesis[4] and artificial life[5], which enable to model and emulate the emergence of heavily interdependent processes such as life. The ability to understand and reproduce these processes provides the basis for building artificial systems with high levels of emergence and interdependencies between their parts.

The research challenge we seek to address is thus:

How to exploit the results of Complex System science for understanding and building artificial complex systems?

Our approach is to explicit the different types of complex system models and their application scope to make them actionable for the design of original artificial complex systems.

2.2 Engineering and the 'Science of the Artificial'

Since we aim at building artificial complex systems, a pre-requisite for our work is to define engineering as a science. A beautiful definition is given by Herbert Simon in 'The Sciences of the Artificial'[6]: A strong distinction is to be made between the Science of the Artificial, i.e., the Science of Engineering and Engineering Sciences. The Science of the Artificial intends to provide a generic analytic framework to investigate artificial systems. Engineering sciences define a normative framework for building such systems efficiently.

The research objects of the science of the artificial hold four major characteristics[6]:

1) Artificial things are synthesized by human being;

2) They may imitate appearances of natural ones while lacking the reality of the latter;

3) They can be described in term of functions, goals or adaptations;

4) They are often discussed in term of imperative as well as descriptive.

Herbert Simon also places a lot of emphasis on the distinction between the Science of the Artificial, which aims at understanding artificial constructs, and the 'Science of Design', i.e., the conception of these 'artificial things'. This 'Science of Design' builds the basis of Engineering science, as the set of natural laws and technical tools necessary to build and operate artificial things from the material, mechanical, electronics, computing or engineering domains.

Whereas we aim at transferring knowledge and tools from Complexity Science to Engineering

Science, the discussion on the 'science of the artificial', the object of interest, specific scientific methods, and observed properties of the objects under investigation, build a major aspect for understanding the background of our contributions.

3 The Models of Complex System Engineering

This section illustrates processes for engineering complex systems through three points of view: the architecture, the model, and the process. Architecture design for Complex System engineering is highlighted by the example of research on artificial cognition and the quest for reproducing the brain functions[7, 8], which emerge from the interconnection of individual neurons. The model types and metrics significant for their evaluation are then detailed[9, 10]. A classification of processes for building artificial complex systems is presented, using the reference model of organically grown architectures[11].

3.1 Architecture Design and Complex System Engineering: The Brain Model Example

The question of the capability of machines to act like humans was popularized as early as 1950, when Turing published his famous 'imitation game'[12] which defines an investigation protocol for evaluating whether a machine can lure a human and make it impossible for him to decide between two interlocutors which one is the human one.

The quest for 'intelligent' machines is pervasive to the history of computer science and expands far beyond the scope of this work. However, the breakthroughs achieved since the 2000's are significant of the challenges of building nature-inspired, artificial systems. Two opposite approaches coexists: The in-silico simulation of natural systems, and the reproduction of system features and properties, with possibly significant deviations or simplifications from the original model. In the case of brain modelling, these two approaches are large brain simulations[13], on the one hand, and biologically inspired cognitive architectures[14], on the other. IBM Blue Brain project is representative of the former[15], and the cat's model of Modha, also from an IBM Team, of the latter[7]. The two projects are significantly different: Markram intends to reproduce and understand the behaviour of a real brain, whereas Modha focuses on the brain as organised complexity and on the reproduction of the organ's features. Other projects, such as the CAM-Brain project[16] using FPGA technology, deploy a similar feature-oriented approach.

The operational approach of Modha is typical of the approach we explore for designing and building artificial complex systems: Our goal is not to perform breakthrough in natural sciences, but to contribute to the engineering domain. Brain-like architectures typically based on the hypothesis that 'the computational building blocks of the brain (neurons and synapses) can be described by relatively compact, functional, phenomenological mathematical models, and that their communication can be summarized in binary, asynchronous messages (spikes)'[7]. This approach considers that 'the behaviour of the brain apparently emerges via non-random, correlated interactions between individual functional units, a key characteristic of organized complexity. Such complex systems are often more amenable to computer modelling and simulation than to closed-form analysis and often resist piecemeal decomposition'. This strategy

is considered in Modha's approach to have brought breakthrough contributions and impressive results.

Another functional model of the brain is the Wang's model[8]. Although it provides fewer evaluations of the actual performance of the system, it proposes a complete conceptual framework for cognition. It defines the interactions with the outer world, the buffer, short-term and long-term memory processes, and defines two processing types, the subconscious life function (NI-OS: Natural Intelligence — Operating System) and the conscious life functions (NI-App: Natural Intelligence – Applications), as depicted in Figure 1.

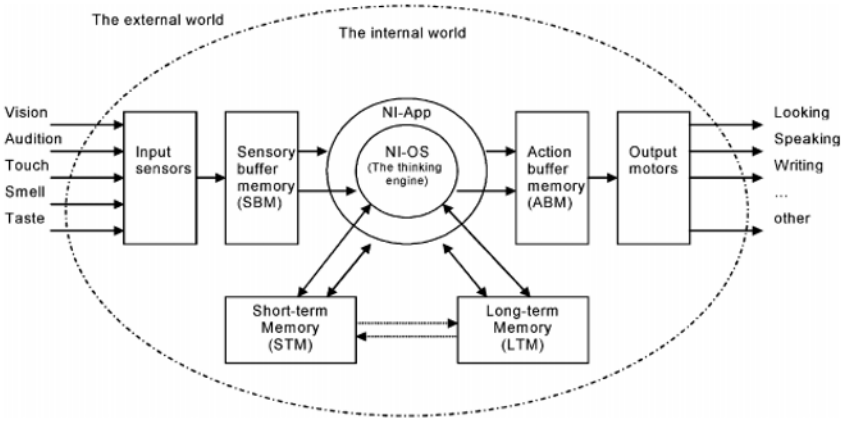

Figure 1 A functional model of the brain[8]

Figure 2 shows the the Layered Reference Model of the Brain (LRBM)[8], which specifies the different, layered functions of natural intelligence at a smaller scale than the functional model: Sensation, memory, perception, action, meta-cognitive functions, and higher cognitive functions.

The challenges of modelling the brain are well representative of the research issues we face in the domain of complex system engineering: The choice between copying the natural interactions or reproducing natural features, the control of emergent interactions between system parts, and multi-layer modelling. It nonetheless introduces a significant simplification compared to other systems, since a brain model is per definition a centralised system, whereas emergence is also a key property of distributed, heterogeneous systems. We focus here on the macro and architectural challenges, and voluntarily neglect model-internal representations.

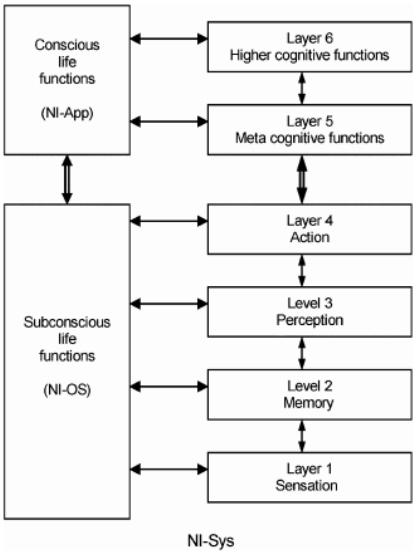

Figure 2 The layered reference model of the brain (LRBM)[8]

3.2 Modelling Complexity

The second challenge of Complex System engineering is the actual modelling approach used to represent the system and to make it operational. One needs to master the stability of complex engineering systems, and to avoid collapse which may emerge from inadequate optimality[17]: In engineering, complexity is often viewed as a source of unsustainability[18]. Its sources can be manifold: Design and product development complexity, manufacturing complexity, business and market complexity[19].

Models of complex systems pertain to two families of systems, matching two very different sets of properties: Self Organised Criticality (SOC), and Highly optimized tolerance (HOT)[20]. The properties of Self Organised Criticality characterize systems which are homogeneous, self-similar, robust, like algorithms based on artificial ants. Highly Optimized Tolerance (HOT) occurs in structured, heterogeneous, self-dissimilar, robust yet fragile systems[9, 20]. The brain model is representative of the HOT property.

One important issue while modelling complex systems is the ability to keep the control over the complexity, in particular by tracking the 'complexity level' of the system. Very few solutions exist for this: The statistical complexity is one of them. It has been introduced in the context of manufacturing processes. Statistical complexity C_{mu} is defined as the Shannon entropy over the distribution of causal states. $P(S_i)$ denotes the probability of the machine being in state S_i[10]:

$$C_{mu} = \sum_{S_i \in S} P(S_i) * \log_2(P(S_i)). \tag{1}$$

Statistical complexity C_{mu} is the average amount of historical memory stored in the process, in bits. In a complex process, more information about the past is stored internally either

through an explicit memory or through the state of system elements such as their location, speed, etc. Prediction therefore requires more information and is, in turn, more difficult. Further investigation would be required to evaluate the relevance of adapting this metric to more complex processes, as to identify the suitable solutions according to the value of the metric.

Examples of modelling approaches for engineering complex systems are provided in Section 4, for illustrating the three representative modelling levels: Network models such as graph and network theory, Collaboration models such as complex adaptative systems and system dynamics, as well as environment model such as stochastic processes.

3.3 Organically Grown Architectures

Although it is currently more geared toward the theoretical evaluation of natural processes, in particular morphogenesis and embryogenesis, the field of Complex System Engineering is actually already defined in the community[11]. The processes proposed by Doursat, if they perfectly match artificial life and organically grown architectures, appear to be actually quite well suited for designing operational artificial systems.

The significant contribution of Complex System Engineering as applied to embryogenesis in particular is the focus brought on emergent engineering, i.e., the construction of systems by themselves. This approach provides a radical shift for exploiting controlled self-evolving systems[21], and introduces very promising perspectives for engineering processes of technical artefacts, which do not pretend to compete with life processes what regards complexity and emergence levels.

This radical shift also highlights the strong limitations of current engineering approaches:

- Traditional engineering requires a system to be well defined;
- Traditional engineering requires a system's performance to be specified;
- Traditional engineering considers complex systems' emergence as an undesirable 'threat';
- Traditional engineering approaches distributed systems design in a top-down centralised manner.

Emergent engineering, on the contrary, requires to significantly redefine usual target properties of the engineering product and processes (see [21] for details):

- Optimality and performance;
- Utility;
- Performance metrics;
- Evolution v.s. evolvability;
- Robustness.

Ideation, design, production, maintenance are thus likely to undergo major mutations under such a paradigm. For instance, embryomorphic engineering supports the creation of decentralized and autonomous systems[22]. Its application to the generation of swarms such as autonomous robots, or flying or swimming drones, opens broad perspectives.

Complex system engineering approaches are defined as pertaining to four major categories[11], which have originally been elicited for morphogenetic engineering:

- Category I: Constructing (or Assembling, Fitting): A small number of mobile agents or components attach to each other or assemble blocks to build a precise 'stick-figure' structure.
- Category II: Coalescing (or Synchronizing, Swarming): A great number of mobile agents flock and make together dense clusters, whose contours adopt certain shapes.
- Category III: Developing (or Growing, Aggregating): The system expands from a single initial agent or group by division or aggregation, forming biological-like patterns or organisms.
- Category IV: Generating (or Rewriting, Inserting): The system expands by successive transformations of components in the 3D space, based on a grammar of 'rewrite' rules.

Figure 3 shows the design-evolution continuum between engineering and biology: Traditional engineering follows the principle of 'intelligent design' (ID), with clearly understood and explicitly defined system architecture and internal models or behaviours. The second step, 'intelligent meta-design' (IMD), consist in generative engineering where constraints and target properties are explicit, but the system building is delegated to a machine or an autonomous system. The third step, 'evolutionary meta-design' (EMD), let the system be generated and evolve over generations, while constraints and target properties are still set. The ultimate step, 'undesigned evolution' (UE), consists is letting system evolve over generation while no longer using explicit constraint and properties, but exploiting feedback from their environment to lead to an evolutionary 'reinforcement learning'.

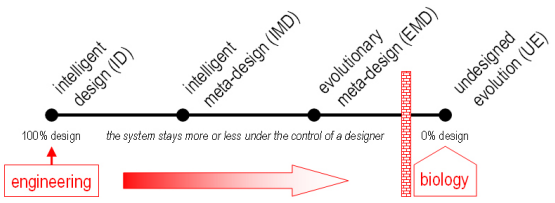

Figure 3 The design-evolution continuum[11]

The field of organically grown architectures thus paves the way towards programmable complex systems, with applications essentially limited to robotics and organ models so far[23]. Programmable architectures already exist[24]. Their patterns for developmental modularity and variability[25] build a critical block for their successful implementation.

From intelligent design to undesigned evolution, organically grown architectures open a broad field of endeavour for the nascent field of Complex System engineering, and provides a framework for evaluating the maturity category of complex system approaches: I) Constructing, II) coalescing, III) developing and IV) generating.

4 Applications

In the wide landscape of Complex System tools which can be applied to engineering, three modelling approaches emerge as pragmatic and powerful solutions: Network model (graph and network theory), collaboration model (complex adaptive systems, system dynamics), environment model (stochastic processes). We now illustrate these models through various success stories of Complex System engineering: Risk analysis of utility infrastructure as Complex Networks, risk management in evolving complex manufactured systems such as helicopters through Design Structure Matrices, System Dynamics for understanding accidents, and bio-inspired stochastic computation. We conclude this section by introducing a recent breakthrough in evolutionary engineering: Complex software engineering, which leverages the properties of artificial evolution for bringing software quality to the next level.

4.1 Utility Infrastructures as Complex Networks

Complex System have been popularized in the engineering domain by the success of network modelling of utility infrastructures and of their weaknesses. One major contribution in the domain is the work of Gorman[26], who have modelled the US optical fibre network, as well as other critical networks such as the gas network. The theoretical foundations for this research was laid by Barabási under the term 'complex networks'[27].

At the example of the US Internet connections, Gorman illustrates that most physical networks are vulnerable to very localised failures such as a line cut by an excavator, which in the earlier years of Internet already caused major overload in the nationwide network, or to domino effects and major regional blackouts following the shutdown of a single plant on the electrical grid. Figure 4 shows an example of such a network: The US optical fibre network[26], which scale-free properties make it more efficient, but challenges its robustness. This study also had a major impact towards the awareness of the policy makers with regard to the quantity of information which can be extracted through data available openly, since it was only performed based on non-confidential data. It can thus be considered as a key moment in the Open Data movement.

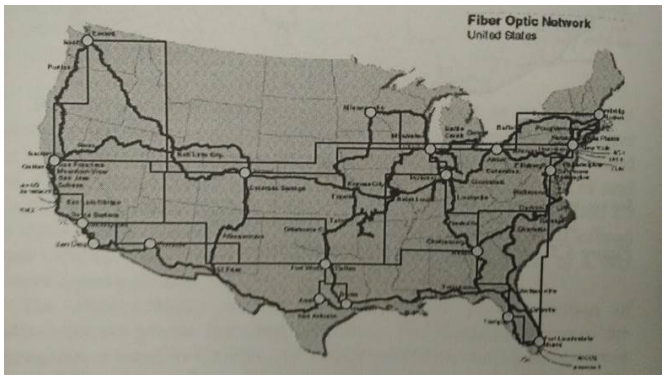

Figure 4 An example of scale-free network: The US optical fiber network[26]

Understanding the topology of networks at different scales thus proves to be of core relevance for the analysis of the interconnections of technical or geographical graphs: It help to understand that a dense interconnection does not abolish the distances or ensure per se a great robustness, but that a new set of metrics is needed to characterize, understand, and maintain the networks. These metrics are in particular:

Centrality: The importance of the vertices in a graph and its variations:

Degree centrality: The number of links incident upon a node.

Closeness: Average length of the shortest path between the node and all other nodes.

Betweenness: The number of times a node acts as a bridge along the shortest path between two other nodes.

Clustering coefficient: The degree to which nodes in a graph tend to cluster together.

Diameter: The maximal distance between any pair of its nodes.

Networks are also characterised by their topologies, which can in particular match following models:

random

Erdós-Rényi (ER) model: With low clustering coefficient and degree distribution converging towards a Poisson distribution[28],

Watts-Strogatz model: With small-world properties such as short average path lengths and high clustering[29],

Barabási-Albert (BA) model: With scale-free properties such as power-law degree distributions leading to the presence of hub, with unusually high degree of connectivity with other nodes[30].

Figure 5 introduces a summary of the difference between random and scale-free networks[27]. Figure 6 illustrates the robustness of a network when facing node removal and Figure 7 the typical topologies of complex networks[31]: Random, small-world and regular.

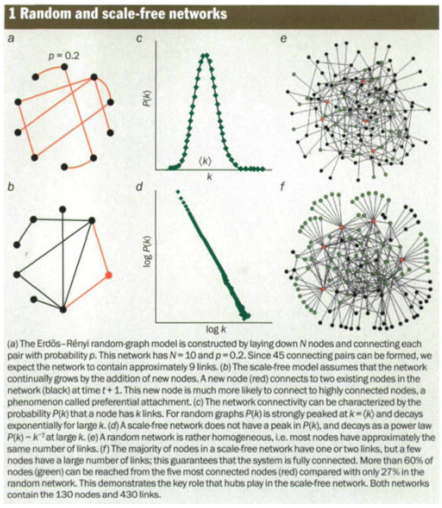

Figure 5 Random and scale-free networks[27]

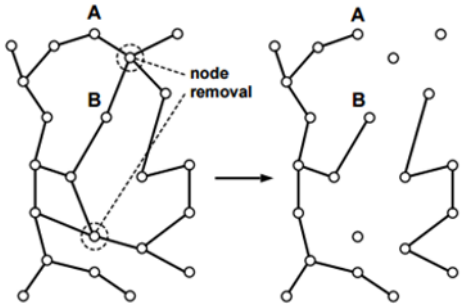

Figure 6 Robustness of a network when facing node removal[31]

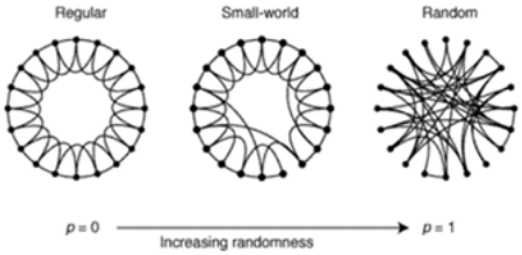

Figure 7 Topologies of complex networks[31]

Similar approaches have been applied to the study of interdependencies in financial networks[32],

as well as subsequent crisis: Knowing the single points of failure, the propagation effects, the probability of occurrence of given events in multi-scale networks is necessary for efficiently ensuring their robustness. Complex networks certainly constitute the better known tool of complex systems outside the Complex System Science community.

4.2 Keeping the Control over Structural Complexity

Another domain where complexity poses a major practical challenge is the manufacturing of highly technological products, where numerous mechanical and electronic parts interact. In this context, the product behaviour itself does not necessarily show emergent properties, but its development time[33], reliability[34], design[35] as well as its robustness to change propagation[36, 37] exhibit a high coupling behaviour. The main drivers of complexity in manufacturing are product enablers, process enablers, market forces and social and environmental pressure, as shown in Figure 8. Product enablers are the manufacturing technology, the product structure, the control system and software, the product complexity and the customer requirements. Process enablers are the global supply chain, human cognitive ergonomics, planning and scheduling, manufacturing system responsiveness and design tools and methodologies. Market forces imply complexity through global competition, turbulence, variety, short delivery and zero defect objectives, among others. Social and environmental pressure mainly comes from standards and government legislation.

So as to solve these challenges, Design Structure Matrix* (DSM)[38, 39] have been proposed, and have evolved since their inception as a research object of their own. Design Structure Matrices represent the dependencies between individual components of systems, which can be interactions based on spatial adjacency, energy flow exchange, information flow exchange or material exchange[38]. They can treat both structural dependencies[38] and time-based dependencies[33] and reflect their propagation through several dependency levels.

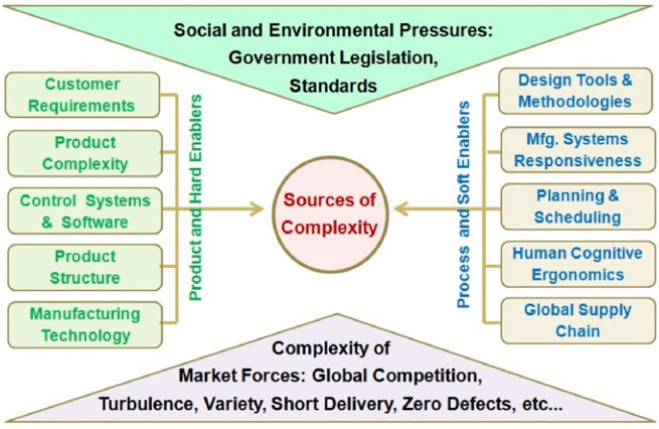

Figure 8 The drivers of manufacturing complexity

*http://www.dsmweb.org/.

The process of generation of the DSM of a given system[40] entails three steps: 1) The creation of the spaghetti graph of the system to be analysed, 2) the extraction of the base DSM out of this spaghetti graph, and 3) the reordering of this DSM. Reordering aims either at simplification or at identification of significant subsystems. Figure 9 shows the spaghetti graph for an example system. Figure 10 shows the base DSM for the example. Figure 11 shows the related partitioned DSM. The reordering of the DSM underlines the presence of three different types of system component relationships: 1) Parallel — Two components have dependencies to and from the same third parties components, but not between themselves; 2) Sequential — One component depends on one another; and 3) coupled — Two components depend on each other[33].

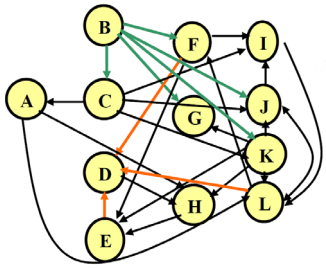

Figure 9 Step 1 of DSM generation: Spaghetti graph[40]

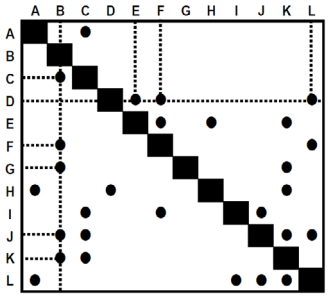

Figure 10 Step 2 of DSM generation: Base DSM[40]

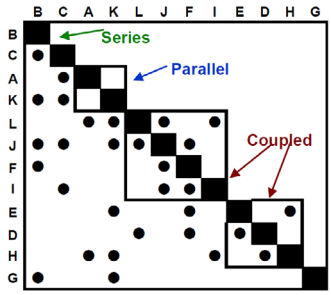

Figure 11 Step 3 of DSM generation: Partitioned DSM[40]

The four operations which can be applied to DSMs for reordering are the following ones[40]:

Partitioning: Manipulating the DSM rows and columns such that the new DSM arrangement does not contain any feedback marks, thus, transforming the DSM into a lower triangular form.

Tearing: The process of choosing the set of feedback marks that if removed from the matrix (and then the matrix is re-partitioned) will render the matrix lower triangular.

Banding: The addition of alternating light and dark bands to a DSM to show independent (i.e., parallel or concurrent) activities or system elements.

Clustering: Finding subsets of DSM elements (i.e., clusters or modules) that are mutually exclusive or minimally interacting[38].

Domain Structure Matrices have been extended to support the heterogeneity of products and processes: Domain mapping matrices[41] support the mapping between two domains, for instance tasks and persons, and multiple domain matrices[42] enable the analysis of multiple domains having multiple elements.

When composite properties build the subject of analysis, Design Structure Matrices can be combined[34]: For instance, risks analysis is performed through the combination of probability and impact matrices. Figure 12 shows the GKN Westland Helicopters EH101, which evolution is taken as a case study for the evaluation of DSMs as a change propagation management tool. Figure 13 shows an excerpt of the DSMs used for the risk evaluation of this change management process: Combined likelihood and combined impact of changes in complex manufactured products enable the computation of combined risks of the planned changes. Each technical interdependency inside the system is identified; the probability of propagation of a flaw from any given component to the next is quantified, as well as the importance of the impact of such a flaw. Recursivity then enables to measure the indirect risks of flow propagation from a given component to a remote one inside the target system.

Figure 12 The GKN westland helicopters EH101[34]

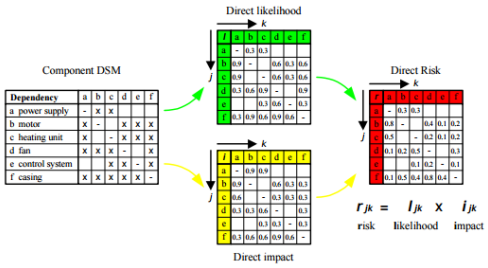

Figure 13 Use of a design structure matrix for change propagation management[34]

4.3 Understanding Accidents

The analysis of emergent properties of complex artificial or natural environments is the subject of the System Dynamics[43] field. In System Dynamics, ecosystems or organisations are modelled through the positive of negative feedback its elements — Or rather its key indicators — Exercise over each other. Accident analysis is a field which is well understood. The STAMP: Systems-Theoretic Accident modelling and Processes[44] provides a comprehensive illustration of the capability of this approach. Its goal is resilience in safety-critical systems[45]. It is based on three core concepts: Constraints, hierarchical levels of control, and process models. It emphasizes the distinction between development and operations. STAMP is based on the risk management framework by Rasmussen[46] which takes hazard source characteristics as well as the full socio-technical system and environmental stressors into account for the analysis: Government, regulators and associations, companies, management, staff and work.

The models of System Dynamic are built by three main types of feedback loops between their elements: The reinforcing loop, shown in Figure 14, the balancing loop, shown in Figure 15, and the balancing loop with a delay, shown in Figure 16[47]. The models can be one-scale model, where individual metrics interact with one another, and multi-scale models, where submodels interact with one another. Tools like VENSIM† or others support the approach.

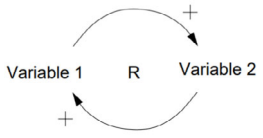

Figure 14 System dynamics: Reinforcing loop[47]

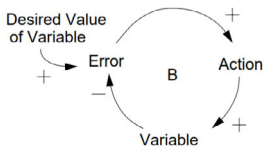

Figure 15 System dynamics: Balancing loop[47]

†http://vensim.com/.

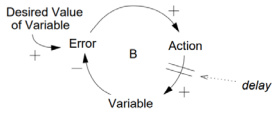

Figure 16 System dynamics: Balancing loop with a delay[47]

Figure 17 shows an excerpt of the model for the multi-scale risk management analysis of Shuttle Columbia loss, performed at NASA[48]. Considered risk submodels are launch rate, perceived success by the administration, system safety resource allocation, shuttle ageing and maintenance, system safety knowledge, skills and staffing, incident learning and corrective action, as well as system safety effort and efficacy. The risk level, which quantification is the target of this investigation, is a model variable for itself.

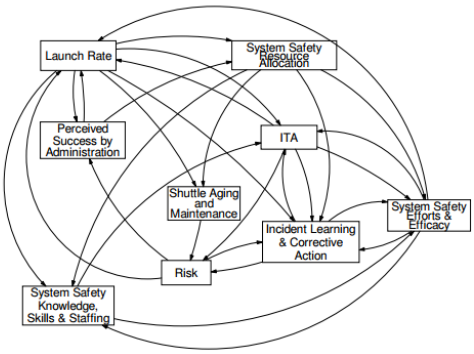

Figure 17 The multi-scale risk management analysis of Shuttle Columbia loss[48]

The modelling using System Dynamics for real-world systems is performed in six steps[43]. This integrated approach from analysis to deployment implies a very thorough validation of field tests and data on which the conclusion are drawn[49].

- Description of the system;
- Conversion of the description to level and rate equations;
- Simulation of the model;
- Design of alternative policies and structures;
- Educate and debate;
- Implement changes in policies and structures.

The evolution of the key indicators over time build the output of the model. They provides significant insights over the behaviour of the system, as well as over potential countermeasures against observed flaws. For instance, Figure 18 shows perceived priorities w.r.t. performance and safety at NASA during the preparation process of spatial flights[47]. Figure 19 depicts the relative impact of fixing symptoms only v.s. fixing systemic factors.

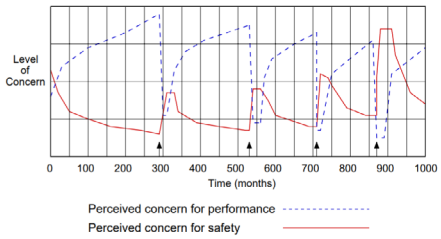

Figure 18 Perceived priorities: Performance v.s. safety[47]

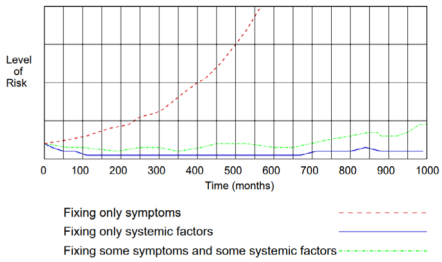

Figure 19 Impact of fixing symptoms only v.s. fixing systemic factors[47]

In cases where only indicators are available from the environment under analysis, System Dynamics provides an efficient framework for modelling the retroactive loops between the various system and environment indicators. A thorough evaluation of the model w.r.t. actual system state enables to evaluate hypotheses and deploy them in the wild. It therefore builds another very pragmatic approach for Complex System modelling applied to the engineering of real products or organisations.

4.4 Bio-Inspired Stochastic Computation

In the field of Computer Science, complexity is tightly bound with Complexity Theory and problems causing combinatorial explosion if one searches to find solutions through systematic exploration, like the Traveling Salesman Problem (TSP). These problems typically pertain to the NP-hard problem category, and although they are not recent, require a radical shift w.r.t. other algorithm types. The idea of exploiting evolution in the computing domain is almost as old as the transistor itself — The first computing efforts on evolution date back to 1954[50]. The theory of this new insight came in the mid-70's as a bio-inspired, stochastic computation approach: Evolutionary algorithms[51]. Actual applications came more than one decade later[52], and the beginning of the XXI's century brought the first significant technological successes of evolutionary approaches, such as the generation of highly focused satellite antennas for the NASA[53]. This dissemination goes along the creation of complementary bio-inspired, stochastic algorithms, such as Ant-Colony Optimisation (ACO) or Particle Swarm Optimisation (PSO).

Evolutionary Algorithms (EAs)

are bio-inspired algorithms which rely on Darwinian theory of evolution[54]. They come in four main flavours[55]:

Genetic algorithms where the solutions are typically represented as strings or binary, and evolve through recombination and mutation[51, 56]. These algorithms are typically used for optimisation problems[57], machine learning[58], or classifiers[59].

Evolution strategies where the potential solutions are represented by numerical vectors[60–62]. The mutation process is in most cases self-adaptive. They have applications in particular for parameter optimisation[62, 63].

Genetic programming considers full-fledged computer programs as the artefacts under mutation, rather than individual solutions to formal problems[64]. The fitness of the program is evaluation w.r.t. a predefined task.

Evolutionary programming is similar to Genetic programming, but the program structure itself is fixed, while only its numerical parameters are allowed to evolve[65, 66]. Often, Finite State Machines are used as predictors.

The individual solutions can thus be either strings, binary, numeric values, parametrized programs, or full-fledge programs. Hybrid approaches are frequent. For instance, so-called memetic algorithms[67] combine a population-based genetic algorithm with specific meta-heuristics for local searches. This approach is in particular efficient for multi-objective problem solving. Genetic algorithms can also be combined with differential evolution[68] for improving local search[69].

Multi-objective problems are not resolved through a single individual, which would be fitter than the others, but through a set of individuals providing a balanced trade-off between the objectives[70, 71]. This set of better fitting individuals, called 'non-dominated', can be visualised with a so-call Pareto front. Figure 20 depicts the various topologies of Pareto fronts for multi-objective problems solved using Evolutionary Algorithms. A whole family of dedicated multi-objectives evolutionary algorithms (MOEAs) addresses the extraction of pareto-optimal solutions through evolutionary algorithms: NSGA2[72], NSGA3[73], ASREA[74], PAES[75] and ESPEA[76].

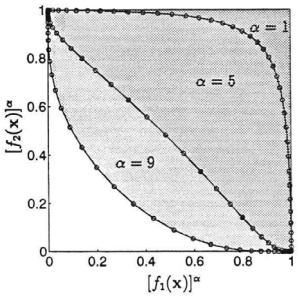

Figure 20 The various topologies of Pareto fronts for multi-objective problems solved using Evolutionary Algorithms[70]

Figure 21 shows the generic process for Evolutionary Algorithms[55, 58], which is are made up of four main steps:

Initialisation The population of potential solutions is created, mostly in a random way, as shown in Figure 22.

Evaluation The current population, either the initial population or the evolved population, is evaluated w.r.t. the fitness function. The fitness function is a numerical value which enables to rank the quality of each individual. A better fitness is typically represented through a smaller fitness value. The fitness function of the population is usually bound with a stop criteria, together with other algorithmic metrics such as the number of iterations: if a sufficiently effective solution is found, or if the search is too long, the algorithm stops and returns the current population as output.

Selection The individuals which are considered the fitter for serving as basis for the creation of the next generation are kept in the system. Often, the best individuals are kept, but sometimes heterogeneous solutions enable to foster exploration of the search space and reduce the risk of sticking to local optima.

Evolution Three main strategies for evolution exist:

Haploïd reproduction i.e., reproduction from a single cell. It is typically implemented through the mutation operator shown in Figure 23 and applied to a single parameter of one given candidate solution.

N-ploïd reproduction i.e., reproduction from several cells. It supports evolution from several existing candidate solutions, as shown in Figure 24. It is for instance typical of Particle Swarm Optimisation (PSO)[77].

Sexual reproduction typically from two cells. It relies on two individuals to create a third one which pertains to the new generation, through the mutation and cross-over operators, as shown in Figure 25.

The two principal operators for evolution are:

Mutation that consists in altering the value of one single individual. It can be performed at random, near the current value, as in Particle Swarm Optimisation (PSO)[77] or Harmony search strategy[78], or consider values of solutions with better evaluation, as in differential evolution[68].

Cross-over that consists in the exchange of one or several parameters (the 'chromosomes') between a pair of individuals of the potential solution set.

Throughout the evolutionary computation process, the evolution step is performed in two phases:

Exploration during the first iterations, consists in generating candidate solutions over the whole search space, as shown in Figure 26.

Exploitation during later iterations, consists in generating candidate solutions based on efficient existing solutions to perform local search, as shown in Figure 27.

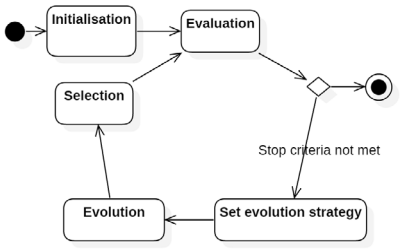

Figure 21 The generic process for Evolutionary Algorithms

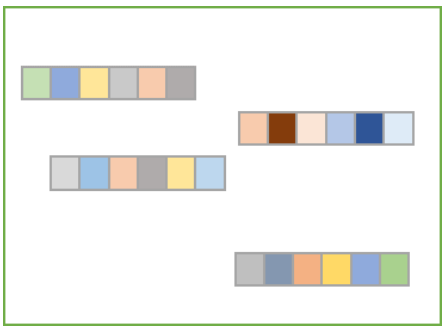

Figure 22 Evolutionary algorithms: The initialisation step

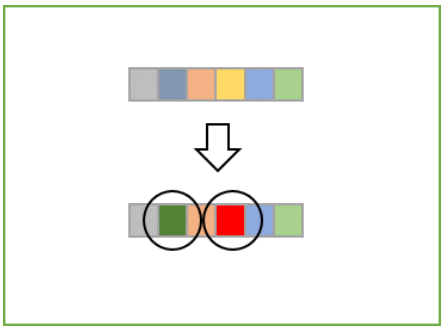

Figure 23 Evolutionary algorithms: The haploïd reproduction strategy for the evolution step

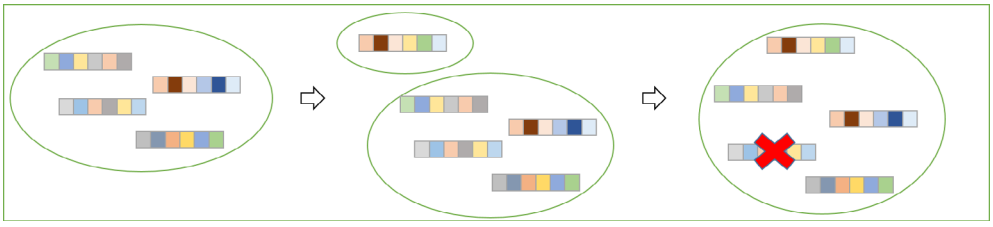

Figure 24 Evolutionary algorithms: The N-ploïde reproduction strategy for the evolution step

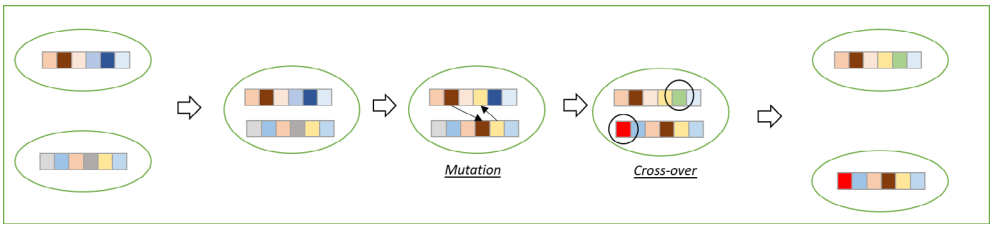

Figure 25 Evolutionary algorithms: The sexual reproduction strategy for the evolution step

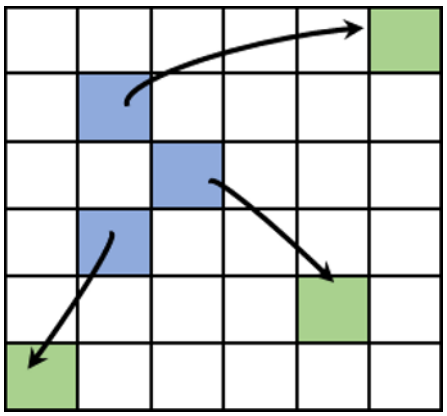

Figure 26 Evolutionary algorithms: The exploration phase of the evolution step

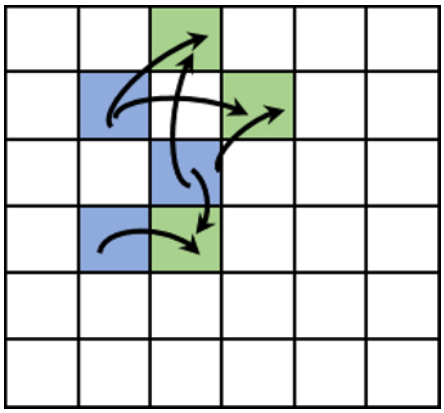

Figure 27 Evolutionary algorithms: The exploitation phase of the evolution step

Each particular Evolutionary Algorithm uses a combination of this overall process[79]. The status of the population being evolved can be of two types: In the Pittsburgh-strategy for Evolutionary Algorithms[80], the population is built of individual candidates, and the problem solution is built out of a single individual of the population, as shown in Figure 28; in the Parisian-strategy for Evolutionary Algorithms, the population is built of partial candidates, and the problem solution is built out of several individuals of the population, as shown in Figure 29. This distinction is in particular exploited in Learning Classifier Systems (LCS)[55, 81].

Pittsburgh-LCS[80, 82, 83] defines a population of classifier-sets, the problem solution is built out of one individual of the population. Michigan-LCS[79, 84, 85] define a population built of individual classifiers, and the problem solution is built out of several individuals of the population. The Parisian approach[86] consists in the generalisation of the Michigan approach to generic Evolutionary Algorithms[87]. It has been in particular evaluated for Iterated Function Systems (IFS), which build systems of functions for generating fractals.

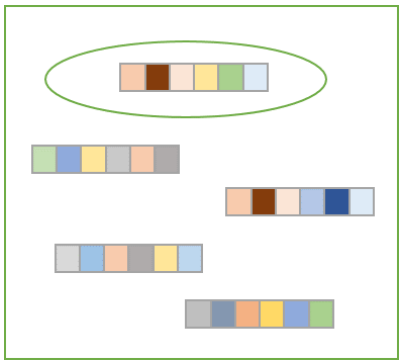

Figure 28 Evolutionary algorithms: Pittsburgh-strategy in the selection step

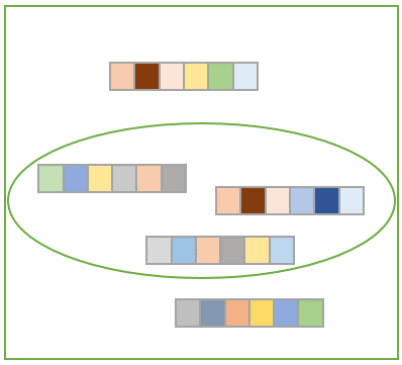

Figure 29 Evolutionary algorithms: Parisian-strategy for the selection step

The efficiency of Evolutionary Algorithm depends on the balance between the phases of exploration, where the overall landscape is discovered, and exploitation, where local search is performed[88]. This balance can be improved by the island model, where local islands exchange individuals from time to time to reduce the risk of being trapped in local optima[89, 90]. The island model provides a theoretical basis for distribution of EAs, either on CPU[91] or on General Purpose Graphical Programming Units (GPGPU)[92]. Such architectures exhibit interesting properties such as the capacity to obtain supralinear performance w.r.t. the actual processing power[93].

4.4.1 Alternative Stochastic Algorithms

In order to exploit bio-inspired stochastic computing in domains where evolutionary algorithms do not outperform, several complementary approaches have been developed. One of

the most famous is Ant-Colony Optimisation (ACO)[94], which are based on work on the Argentina ant 'Iridomyrnex Humilis' by Deneubourg[95, 96]. Whereas EAs intend to solve complex problems through a population based approach, ACOs exploit the emergent properties of collaborating unitary agents, which interactions enable to identify efficient paths in dynamic environments through the deposit of pheromones: This is stygmergy. Figure 30 shows 'Iridomyrnex Humilis' ants finding a short path through two bridges between nest and food[97]. Photos are shot 4 and 8 minutes after the bridge is set, respectively.

Other approaches are Particle Swarm Optimisation (PSO)[77] for local search strategies, Harmony Search which combines a genetic algorithm with a local search strategy[78], or the Fly algorithm[98] for identifying moving objects in video processing. The exhaustive presentation of these algorithms goes far beyond the scope of this contribution. We will therefore not elaborate further on them. As a conclusion, bio-inspired stochastic algorithms introduce non-deterministic optimisation for complex problems, and provides mutation and cross-over mechanisms able to solve complex problems as well as to generate original knowledge. They therefore lay a solid basis for studying complex problems in the Computer Science field.

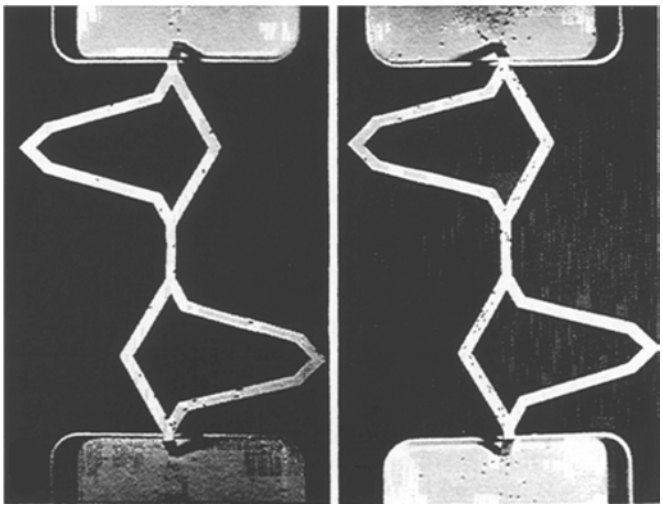

Figure 30 'Iridomyrnex Humilis' ant finding a short path through two bridges between nest and food[97]

4.5 Complex Software Engineering

One of the most recent breakthroughs in the application of Complex System science to engineering is the Genetic Improvement (GI) of programs[99], originally known as Probabilistic incremental program evolution (PIPE)[100]. This approach is a major milestone towards the automation of programming, and thus opens great perspectives for the evolution of the job of software developers.

Genetic Improvement is defined as the process of using computational search techniques to improve existing software in particular in terms of correctness (bug fixing), execution time, or power consumption[101]. It consists in the automated evolution of software programs, under the

form of code, abstract code representation, or even binary packages. One of the key findings of GI is that, unintuitively, software is very robust to mutations: In a reference experiment, Langdon shows that more than 60 % of the mutations of a given test program have actually no impact on the behaviour of the program whatsoever[102]. Figure 31 illustrates the results of this experiment on random changes in code lines: 4400 out of 7079 mutations, on the StereoCamera kernel on an NVidia GeForce FT 730 graphic card, i.e., 62%, have not impact on the execution. 41 mutations, i.e., 0, 6%, of the mutations conserve the program functionalities while actually improving its performances. Other 38 % of mutations negatively alter functionality or performance.

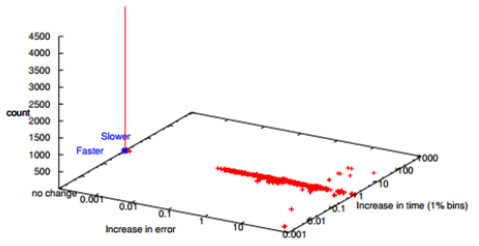

Fig. 2. Impact of all (7079) one change mutations on StereoCamera kernel [16] running on an nVidia GeForce GT 730 graphics card. Nine mutants failed to compile due to a bug in the nVidia CUDA 6.0.1 compiler (fixed in CUDA 7.0). 16 caused infinite loops, 318 others failed at run-time. 4400 (62%) mutants do not change the output at all ("no change"), indeed at least 41 of them are faster (×).

Figure 31 The impact of random changes in code lines[102]

One of the operational challenge for such approaches is of course that the test cases, which serve as reference for the evaluation of the fitness function of the program, must cover all explicit as well as implicit requirements of the software: Otherwise, new bugs or inconsistencies can appear unnoticed. A complete coverage is far from being achieved in most software projects so far, so the application of Genetic Improvement implies a radical shift in the quality of the software engineering process.

GenProg[103] is one of the reference framework for Genetic Improvement. It proved capable of solving bugs which are very tedious to correct manually, such as segmentation faults, for programs of length of several millions lines of code. GenProg represents a program as a pair of:

- An abstract syntax tree (AST) that includes all of the statements in the program;
- A weighted path consisting of a list of program statements, each associated with a weight based on that statement's occurrence in the execution traces of various test cases.

The update operator of the genetic algorithm alters the path. This alteration is reflected back in the program AST. The process iterates until the fitness function confirms that the test cases execute without error. The potential drawbacks of this approach are usually more than compensated by the rapid generation of robust software patches, but need nonetheless to be taken into account: High costs are bound with repairs that actually degrade the functionality, in case of insufficient code coverage by tests, or with the application of repairs following false

positives generated by intrusion detection systems, i.e., which actually do not match an actual vulnerability or instability in the program.

The issue of code coverage poses a major challenge for the wide dissemination of GI. Therefore, some authors propose an iterative process including regular interactions with the user, so as to complete the test case set: Figure 32 shows a three-stage evolutionary approach for evolutionary software repair by Schulte[104], applied to firmware and drivers. The exploitation of the expert feedback also enables a radical shift in the type of programs which can be corrected: rather than considering the program code only, or an abstract representation of the code as AST, it becomes possible to repair program binaries, with little to no preliminary knowledge about the program. First, the vulnerable executable binary is extracted from the firmware. Next, the program is mutated in order to find versions of the executable which solves the bugs or vulnerabilities. Then, the user interactively adds test cases that covers the functionalities broken by the current mutation. The process iterates until an actual program repair is found.

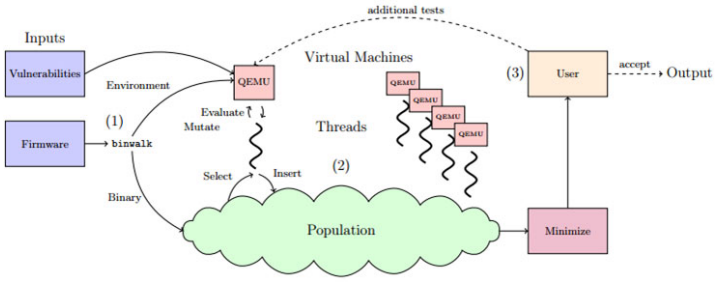

Figure 1: Three-stage automated evolutionary repair technique: (1) the vulnerable binary is extracted from the vendor-supplied firmware; (2) EC techniques are used to find versions of the binary which resolve the target vulnerabilities; (3) the user interactively adds test cases protecting broken functionality and returns to step 2 until an acceptable repair is found.

Figure 32 An approach for evolutionary software repair[102]

Fine-tuning the Genetic Improvement process requires to define specific metrics for the underlying genetic algorithms[101]: The fitness functions is specific to the objective of the GI process, and needs to take into account the evaluation of the functional stability of the program. For bug fixing it is inherently discrete, whereas it becomes continuous when the objective is to improve execution time or to reduce memory consumption. The edit distance measures the number of mutations between two valid programs, and thus evaluates the depth of the performed alterations.

Genetic Improvement paves the way to program synthesis: For instance, evolutionary software sketching[105] enables to quicken the generation of full programs by the machine: Instead of generating a comprehensive program, a first sketch is created by the developers, and serves as basis for the synthesis of missing parts. This approach provides competitive performances w.r.t. full synthesis. The proposed approach lays on Satisfiability Modulo Theories (SMT)[106, 107], a form of the constraint satisfaction problem.

5 Conclusions and Perspectives

Complex System approaches, in particular Evolutionary Algorithms, provide a very promising way to explore the search space of highly complex artefacts like software programs. They are likely to take even more significance as the fields of Genetics Improvement, meant for program optimisation and repair, and Program Synthesis, which aims at generating full applications, will gain in maturity. These evolutions will highly probably bring major disruption in the IT and development industry, and job market, in the coming decades.

Acknowledgements

We thank the CSTB team at ICube laboratory, René Doursat from the Manchester University for valuable exchanges on the subject of morphogenetic engineering and Claudia Eckert from the Open University in London for her pedagogical work on Design Structure Matrices.

References

[1] Holland J, *Complexity: A Very Short Introduction*, Very Short Introductions, OUP Oxford, 2014.

[2] Morin E, *Introduction à la pensée complexe*, Le Seuil, 2015.

[3] Bourgine P and Lesne A, *Morphogenesis: Origins of Patterns and Shapes*, Springer Science & Business Media, 2010.

[4] Zanella C, Campana M, Rizzi B, et al., Cells segmentation from 3-d confocal images of early zebrafish embryogenesis, *IEEE Transactions on Image Processing*, 2010, **19**(3): 770–781.

[5] Bogunia-Kubik K and Sugisaka M, From molecular biology to nanotechnology and nanomedicine, *Biosystems*, 2002, **65**(2): 123–138.

[6] Simon H A, *The Sciences of the Artificial*, MIT Press, 1996.

[7] Modha D S, Ananthanarayanan R, Esser S K, et al., Cognitive computing, *Communications of the ACM*, 2011, **54**(8): 62–71.

[8] Wang Y X, Wang Y, Patel S, et al., A layered reference model of the brain (lrmb), *IEEE Transactions on Systems, Man, and Cybernetics, Part C* (*Applications and Reviews*), 2006, **36**(2): 124–133, 2006.

[9] Carlson J M and Doyle J, Complexity and robustness, *Proceedings of the National Academy of Sciences*, 2002, **99**(suppl 1): 2538–2545.

[10] van Eijnatten F, Putnik G, and Sluga A, Chaordic systems thinking for novelty in contemporary manufacturing, *CIRP Annals-Manufacturing Technology*, 2007, **56**(1): 447–450.

[11] Doursat R, Sayama H, and Michel O, A review of morphogenetic engineering, *Natural Computing*, 2013, **12**(4): 517–535.

[12] Turing A M, Computing machinery and intelligence, *Mind*, 1950, **59**(236): 433–460.

[13] De Garis H, Shuo C, Goertzel B, et al., A world survey of artificial brain projects, part I: Large-scale brain simulations, *Neurocomputing*, 2010, **74**(1): 3–29.

[14] Goertzel B, Lian R, Arel I, et al., A world survey of artificial brain projects, part II: Biologically inspired cognitive architectures, *Neurocomputing*, 2010, **74**(1): 30–49.

[15] Markram H, The blue brain project, *Nature Reviews Neuroscience*, 2006, **7**(2): 153–160.
[16] De Garis H, Korkin M, Gers F, et al., Building an artificial brain using an fpga based cam-brain machine, *Applied Mathematics and Computation*, 2000, **111**(2): 163–192.
[17] Fisk D, Engineering complexity, *Interdisciplinary Science Reviews*, 2004, **29**(2): 151–161.
[18] Fisk D and Kerherve J, Complexity as a cause of unsustainability, *Ecological Complexity*, 2006, **3**(4): 336–343.
[19] ElMaraghy W, ElMaraghy H, Tomiyama T, et al., Complexity in engineering design and manufacturing, *CIRP Annals-Manufacturing Technology*, 2012, **61**(2): 793–814.
[20] Carlson J M and Doyle J, Highly optimized tolerance: Robustness and design in complex systems, *Physical Review Letters*, 2000, **84**(11): 2529.
[21] Ulieru M and Doursat R, Emergent engineering: A radical paradigm shift, *International Journal of Autonomous and Adaptive Communications Systems*, 2010, **4**(1): 39–60.
[22] Doursat R, Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering, *Organic Computing*, Springer, 2009, 167–199.
[23] Doursat R, Sayama H, and Michel O, *Morphogenetic Engineering: Toward Programmable Complex Systems*, Springer, New York, 2012.
[24] Doursat R, Programmable architectures that are complex and self-organized-from morphogenesis to engineering, *ALIFE*, 2008, 181–188.
[25] Doursat R, Facilitating evolutionary innovation by developmental modularity and variability, *Proceedings of the* 11*th Annual conference on Genetic and evolutionary computation*, ACM, 2009, 683–690.
[26] Gorman S P, *Networks, Security and Complexity: The Role of Public Policy in Critical Infrastructure Protection*, Edward Elgar Publishing, 2005.
[27] Barabási A L, The physics of the web, *Physics World*, 2001, **14**(7): 33.
[28] Erdos P and Rényi A, Publicationes mathematicae 6, *On Random Graphs*, 1959, **1**: 290–297.
[29] Watts D J and Strogatz S H, Collective dynamics of'small-world'networks, *Nature*, 1998, **393**(6684): 440.
[30] Albert R and Barabási A L, Statistical mechanics of complex networks, *Reviews of Modern Physics*, 2002, **74**(1): 47.
[31] Barabási A L, Linked: The new science of networks, 2003.
[32] Roukny T, Bersini H, Pirotte H, et al., Default cascades in complex networks: Topology and systemic risk, *Scientific Reports*, 2013, **3**: 2759.
[33] Carrascosa M, Eppinger S D, and Whitney D E, Using the design structure matrix to estimate product development time, *Proceedings of the ASME Design Engineering Technical Conferences* (*Design Automation Conference*), 1998, 1–10.
[34] Eckert C M, Keller R, Earl C, et al., Supporting change processes in design: Complexity, prediction and reliability, *Reliability Engineering & System Safety*, 2006, **91**(12): 1521–1534.
[35] Maurer M S, Structural awareness in complex product design, PhD Thesis, Universität München, October, 2007.
[36] Clarkson P J, Simons C, and Eckert C, Predicting change propagation in complex design, *Journal of Mechanical Design* (*Transactions of the ASME*), 2004, **126**(5): 788–797.
[37] Giffin M, de Weck O, Bounova G, et al., Change propagation analysis in complex technical systems, *Journal of Mechanical Design*, 2009, **131**(8): 081001.
[38] Pimmler T U and Eppinger S D, Integration analysis of product decompositions, *ASME De-*

sign Theory and Methodology Conference, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1994.

[39] Browning T R, Applying the design structure matrix to system decomposition and integration problems: A review and new directions, *IEEE Transactions on Engineering Management*, 2001, **48**(3): 292–306.

[40] Yassine A, An introduction to modeling and analyzing complex product development processes using the design structure matrix (dsm) method, *Urbana*, 2004, **51**(9): 1–17.

[41] Danilovic M and Sandkull B, The use of dependence structure matrix and domain mapping matrix in managing uncertainty in multiple project situations, *International Journal of Project Management*, 2005, **23**(3): 193–203.

[42] Maurer M and Lindemann U, Structural awareness in complex product design–the multiple-domain matrix, *DSM* 2007*: Proceedings of the* 9*th International DSM Conference, Munich, Germany*, 2007, 87–97.

[43] Forrester J W, System dynamics, systems thinking, and soft or, *System Dynamics Review*, 1994, **10**(2–3): 245–256.

[44] Leveson N, A new accident model for engineering safer systems, *Safety Science*, 2004, **42**(4): 237–270.

[45] Leveson N, Daouk M, Dulac N, et al., A systems theoretic approach to safety engineering, *Dept. of Aeronautics and Astronautics, Massachusetts Inst. of Technology*, Cambridge, 2003.

[46] Rasmussen J, Risk management in a dynamic society: A modelling problem, *Safety Science*, 1997, **27**(2): 183–213.

[47] Leveson N, Dulac N, and Zipkin D, N. Dulac Engineering resilience into safety-critical systems, *Resilience Engineering — Concepts and Precepts*, Ashgate Aldershot, 2006, 95–123.

[48] Dulac N, A framework for dynamic safety and risk management modeling in complex engineering systems, PhD Thesis, Citeseer, June 2007.

[49] Barlas Y, Formal aspects of model validity and validation in system dynamics, *System Dynamics Review*, 1996, **12**(3): 183–210.

[50] Barricelli N A, et al., Esempi numerici di processi di evoluzione, *Methodos*, 1954, **6**(21–22): 45–68.

[51] Holland J H, Genetic algorithms and the optimal allocation of trials, *SIAM Journal on Computing*, 1973, **2**(2): 88–105.

[52] De Jong K A, Are genetic algorithms function optimizers?, *PPSN*, 1992, **2**(1): 3–14.

[53] Lohn J D, Linden D S, Hornby G S, et al., Evolutionary design of an x-band antenna for nasa's space technology 5 mission, *Antennas and Propagation Society International Symposium, IEEE*, 2004, **3**: 2313–2316.

[54] Darwin C, *The Origin of Species by Means of Natural Election, Or the Preservation of Favored Races in the Struggle for Life*, AL Burt., 1859.

[55] Back T, Hammel U, and Schwefel H P, Evolutionary computation: Comments on the history and current state, *IEEE Transactions on Evolutionary Computation*, 1997, **1**(1): 3–17.

[56] Holland J H, Genetic algorithms, *Scientific American*, 1992, **267**(1): 66–72.

[57] Deb K, Agrawal S, Pratap A, et al., A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, *International Conference on Parallel Problem Solving From Nature*, 2000, 849–858.

[58] Goldberg D E and Holland J H, Genetic algorithms and machine learning, *Machine Learning*, 1988, **3**(2): 95–99.

[59] Booker L B, Goldberg D E, and Holland J H, Classifier systems and genetic algorithms, *Artificial Intelligence*, 1989, **40**(1–3): 235–282.

[60] Eigen M, Ingo rechenberg evolutionsstrategie optimierung technischer systeme nach prinzipien der biologishen evolution, *mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt*, 1973, **45**: 46–47.

[61] Schwefel H P, *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie*, Birkhäuser, Basel Switzerland, 1977.

[62] Schwefel H P, Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of organic evolution, *Annals of Operations Research*, 1984, **1**(2): 165–167.

[63] Bäck T and Schwefel H P, An overview of evolutionary algorithms for parameter optimization, *Evolutionary Computation*, 1993, **1**(1): 1–23.

[64] Koza J R, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[65] Fogel L J, Owens A J, and Walsh M J, *Artificial Intelligence Through Simulated Evolution*, John Wiley, 1966.

[66] Fogel L J, Evolutionary programming in perspective: The top-down view, *Computational Intelligence: Imitating Life*, 1994.

[67] Moscato P, et al., On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Caltech Concurrent Computation Program, C3P Report*, 1989, **826**: 1989.

[68] Storn R and Price K, Differential evolution — A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization*, 1997, **11**(4): 341–359.

[69] Zaharie D and Micota F, Revisiting the analysis of population variance in differential evolution algorithms, *IEEE Congress Eonvolutionary Computation* (*CEC*), 2017, 1811–1818.

[70] Fonseca C M and Fleming P J, An overview of evolutionary algorithms in multiobjective optimization, *Evolutionary Computation*, 1995, **3**(1): 1–16.

[71] Zitzler E and Thiele L, Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach, *IEEE Transactions on Evolutionary Computation*, 1999, **3**(4): 257–271.

[72] Deb K, Pratap A, Agarwal S, et al., A fast and elitist multiobjective genetic algorithm: Nsga-ii, *IEEE transactions on Evolutionary Computation*, 2002, **6**(2): 182–197.

[73] Deb K and Jain H, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints, *IEEE Trans. Evolutionary Computation*, 2014, **18**(4): 577–601.

[74] Sharma D and Collet P, An archived-based stochastic ranking evolutionary algorithm (asrea) for multi-objective optimization, *Proceedings of the* 12*th Annual Conference on Genetic and Evolutionary Computation*, 2010, 479–486.

[75] Knowles J D and Corne D W, Approximating the nondominated front using the pareto archived evolution strategy, *Evolutionary Computation*, 2000, **8**(2): 149–172.

[76] Collet P and Schoenauer M, Guide: Unifying evolutionary engines through a graphical user interface, *International Conference on Artificial Evolution* (*Evolution Artificielle*), Springer, 2003, 203–215.

[77] Eberhart R and Kennedy J, A new optimizer using particle swarm theory, *IEEE Proceedings of*

the Sixth International Symposium on Micro Machine and Human Science, 1995, 39–43.

[78] Geem Z W, Kim J H, and Loganathan G, A new heuristic optimization algorithm: Harmony search, *Simulation*, 2001, **76**(2): 60–68.

[79] Holland J H, Adaptation in natural and artificial systems: An introductory analysis with application to biology, control, and artificial intelligence, *Ann Arbor*, MI: University of Michigan Press, 1975.

[80] Dejong K, An analysis of the behaviour of a class of genetic adaptive systems, PhD Thesis, Dept. of Computer and Communication Sciences, University of Michigan, *Ann Arbor*, 1975.

[81] Bull L, Learning classifier systems: A brief introduction, *Applications of Learning Classifier Systems*, 2004, 1–12.

[82] Smith S F, Flexible learning of problem solving heuristics through adaptive search, *IJCAI*, 1983, **83**: 422–425.

[83] Bacardit J and Garrell J M, Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system, *Genetic and Evolutionary Computation Conference*, 2003, 1818–1831.

[84] Goldberg D E, Computer-aided gas pipeline operation using genetic algorithms and rule learning, PhD Thesis, University of Michigan, January, 1983.

[85] Holland J H, Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems, *Machine Learning: An Arti-Ficial Intelligence Approach*, 1986, 593–623.

[86] Collet P, Lutton E, Raynal F, et al., Polar IFS+ Individual genetic programming = efficient IFS inverse problem solving, *Genetic Programming and Evolvable Machines*, 2000, **1**(4): 339–361.

[87] Hutchinson J E, Fractals and self similarity, *Indiana University Mathematics Journal*, 1981, **30**(5): 713–747.

[88] Črepinšek M, Liu S H, and Mernik M, Exploration and exploitation in evolutionary algorithms: A survey, *ACM Computing Surveys* (*CSUR*), 2013, **45**(3): 35.

[89] Martin W, Lienig J, and Cohoon J P, C6. 3 island (migration) models: Evolutionary algorithms based on punctuated equilibria, Seiten C, 1997.

[90] Melab N, Talbi E G, et al., Gpu-based island model for evolutionary algorithms, *Proceedings of the* 12*th annual conference on Genetic and Evolutionary Computation*, ACM, 2010, 1089–1096.

[91] Arenas M G, Collet P, Eiben A E, et al., A framework for distributed evolutionary algorithms, *International Conference on Parallel Problem Solving from Nature*, 2002, 665–675.

[92] Maitre O, Baumes L A, Lachiche N, et al., Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea, *Proceedings of the* 11*th Annual Conference on Genetic and Evolutionary Computation*, 2009, 1403–1410.

[93] Krüger F, Baumes L, and Collet P, Exploiting clusters of gpu machines with the easea platform, *Artificial Evolution* 2011 (*Evolution Artificielle* 2011), 2011.

[94] Dorigo M, Maniezzo V, and Colorni A, Ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics, Part B* (*Cybernetics*), 1996, **26**(1): 29–41.

[95] Deneubourg J L and Goss S, Collective patterns and decision-making, *Ethology Ecology & Evolution*, 1989, **1**(4): 295–311.

[96] Deneubourg J L, Aron S, Goss S, et al., The self-organizing exploratory pattern of the argentine ant, *Journal of Insect Behavior*, 1990, **3**(2): 159–168, 1990.

[97] Goss S, Aron S, Deneubourg J L, et al., Self-organized shortcuts in the argentine ant, *Naturwissenschaften*, 1989, **76**(12): 579–581.

[98] Louchet J, Guyon M, Lesot M J, et al., Dynamic flies: A new pattern recognition tool applied to stereo sequence processing, *Pattern Recognition Letters*, 2002, **23**(1): 335–345.

[99] Langdon W B, Genetic improvement of programs, 2014 16*th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (*SYNASC*)*,* 2014, 14–19.

[100] Salustowicz R and Schmidhuber J, Probabilistic incremental program evolution, *Evolutionary Computation*, 1997, **5**(2): 123–141.

[101] Haraldsson S O, Woodward J R, Brownlee A E, et al., Exploring fitness and edit distance of mutated python programs, *European Conference on Genetic Programming*, 2017, 19–34.

[102] Langdon W B and Petke J, Software is not fragile, *First Complex Systems Digital Campus World E-Conference* 2015, 2017, 203–211.

[103] Le Goues C, Nguyen T, Forrest S, et al., Genprog: A generic method for automatic software repair, *IEEE Transactions on Software Engineering*, 2012, **38**(1): 54–72.

[104] Schulte E M, Weimer W, and Forrest S, Repairing cots router firmware without access to source code or test suites: A case study in evolutionary software repair, *Proceedings of the Companion Publication of the* 2015 *Annual Conference on Genetic and Evolutionary Computation*, ACM, 2015, 847–854.

[105] Błądek I and Krawiec K, Evolutionary program sketching, *European Conference on Genetic Programming*, Springer, New York, 2017, 3–18.

[106] Ranise S and Tinelli C, Satisfiability modulo theories, *Trends and Controversies-IEEE Intelligent Systems Magazine*, 2006, **21**(6): 71–81.

[107] Barrett C W, Sebastiani R, Seshia S A, et al., Satisfiability modulo theories, *Handbook of Satisfiability*, 2009, **185**: 825–885.