Advances in Complexity Engineering

Regina Frei
Birkbeck, University of London, UK
work@reginafrei.ch

Giovanna Di Marzo Serugendo
University of Geneva, Switzerland
giovanna.dimarzo@unige.ch

September 1, 2010

Abstract

Complexity science has seen increasing interest in the recent years. Many engineers have discovered that traditional methods come to their limits when coping with complex adaptive systems or autonomous agents. To find alternatives, complexity science can be applied to engineering, resulting in a quickly growing field, referred to as *complexity engineering*. Most current efforts come either from scientists who are interested in bio-inspired methods and working in computer science or mobile robots, or they come from the area of systems engineering. This article reviews the definitions of the most important concepts such as emergence and self-organisation from an engineer's perspective, and analyses different types of nature-inspired technology. It provides an survey of the currently existing approaches to complexity engineering. Finally, challenges ahead are indicated.

1 Introduction

Traditional engineering methods cope well with *reducible systems* [1], i.e. those systems which can be decomposed without loss and which are made of parts or sub-systems which are well known, which interact in predefined and well-understood ways and mostly stay the same during the system's life time. For reducible systems, the sum of their parts makes the whole. Systems with emergence, at the contrary, are more (or less?) than the sum of their parts. For instance, swarming birds only follow very simple local rules, but as a whole the swarm exhibits sophisticated dynamic formations. In manufacturing, robotic modules may function perfectly in a certain arrangement, but not in another one. GPS-based mobile services may show good performance at locations where difficulties were expected, and bad performance in areas of good reception because of the interference of other devices.

An increasing number of modern systems do not correspond to the description of a reducible system; their composition as well as the user requirements and the environment dynamically change, and often, their behaviour or some of their characteristics are emergent.

Complexity is omnipresent [2], and there are two main directions of research: (1) complexity as an emerging phenomenon (in natural or engineered systems) to be understood and (2) complexity as an engineering problem to be tackled, mostly by reducing the environmental complexity, or by augmenting the system's capabilities of coping with complexity [3].

Complexity engineering [4] can be considered as a third direction, which currently attracts the attention of an increasing number of researchers: using complexity for engineering - not fighting against it, but using it to the engineer's favour. This is the topic of this article. Under the name of *emergent engineering* [5] argues for the same paradigm change as we suggest with complexity engineering.

Despite a lot of knowledge about complex systems, the application of this knowledge to the engineering domain remains difficult. Efforts are scattered over many scientific and engineering disciplines such as software engineering, social sciences, econ-

omy, physics, chemistry, biotechnology, and others.

Only few of the projects cited in this article have a systematic approach which could be applied to other problems. This lack of general methodologies may have various reasons. Compared to other engineering branches, complexity science is quite recent, and complexity engineering even more so. While researchers observe the typical characteristics of complexity in many different areas, the way of treating them or using them is mostly very individually tailored for the specific system at hand. Furthermore, there is probably a lack of incentives for unifying complexity-related methods, as researchers often rather consider themselves as experts for their area than as complexity engineers.

There is clearly a need for systematic approaches and generally valid methods. The focus of this article is therefore on how to use the findings of complexity science for engineering, with the most prominent ingredients' being self-organisation and emergence.

Complexity related research areas: Figure 1 illustrates the situation of the complexity researcher; many different areas are related and relevant for many different types of complex systems in nature and engineering. A multi-disciplinary approach and the ability to communicate with specialists from many different domains are required. How can this overwhelming richness of concepts be managed? Are there useful principles?

First of all, it is necessary to very well understand the characteristics of the system being studied, or the requirements of the systems being engineered [6]. Second, the key concepts for success have to be identified. Most of them cannot be found in traditional engineering disciplines. Third, the concepts and methods taken from non-engineering domains have to be adapted in order to comply with engineering principles.

The inherent multi-disciplinarity requires researchers able of understanding a broad range of concepts, methods and principles. An example of such multi-disciplinarity is natural computing [7], where natural sciences meet computer science and all kinds of bio-inspired methods are applied to engineering issues. Another example are self-organising assembly systems [8, 9], where agile manufacturing comes together with software engineering and complexity science. It is an area that is used for illustrative examples throughout this article.

1.1 Scope and organisation

The topic of this article is engineering, not the sole study of complex systems. We therefore do not discuss *natural* complex systems, but rather consider how to engineer *artificial* complex systems, and how to use the findings of complexity science. Different complexity disciplines are explained in section 2.

Researchers such as Lucas [10], Wolfram [11], Holland [12, 13, 14, 15], De Wolf [16] and Gershenson [17] have studied the various definitions for complex systems and their characteristics. We therefore only briefly consider complex systems definitions in section 3 and lay the focus on various notions which are important for this article, like self-organisation and emergence. The controversies between emergence, surprise, unpredictability, (non-)determinism and others are discussed as well as the differences between distributed and decentralised control.

The survey in section 4 covers work done under the names of *emergence engineering*, *complexity engineering* and other related terms because they mostly address the same type of system and use similar approaches. Typical application areas and concrete cases of complexity engineering are:

- Systems engineering: systems of systems in health care, military defense and transportation including pedestrians, bikes, cars, buses, trains and planes.
- Mobile robotics: swarms for maintenance and safety.
- Manufacturing automation: agile and evolvable production systems.
- Software engineering: peer-2-peer, multi-agent systems, safety-critical applications.
- Communication systems: persuasive computing.
- Business / finance / economy: prediction and influencing.
- Nanotechnology and biotechnology: cell engineering, nano-robotics for medical applications.

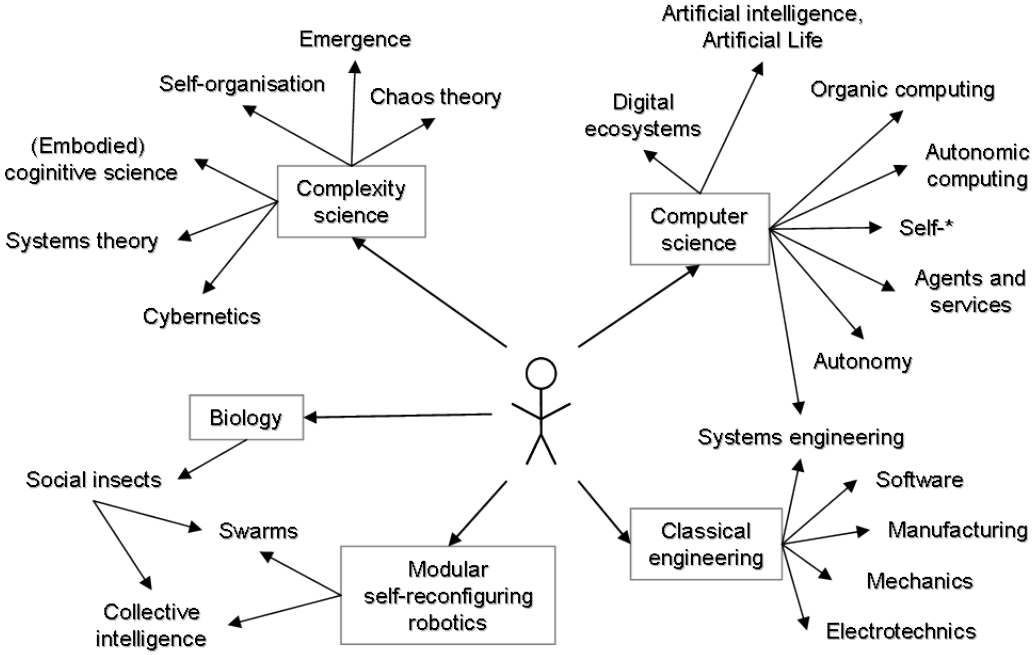

Figure 1: Complexity related research areas

	Individual systems	Systems of systems
Reducible systems	*Classical engineering*	*Classical systems engineering*
Systems with emergence	*Complexity engineering*	*Complex systems engineering*

Table 1: System types and engineering types

In section 5, we reflect on the complexity engineering approach, draw conclusions and give directions for future work.

2 Engineering types

Complex and unconventional systems require different mind-sets than offered by classical engineering. This is why general and complexity-related engineering comes in various flavours. It is important for the reader to understand the different types of engineering, some of which are only currently emerging, and have not been established as proper disciplines yet. This does, however, not reduce their importance and relevance.

Table 1 gives an overview of the engineering types and the systems which they respectively address; these engineering types are discussed in sections 2.2 to 2.5.

2.1 Preliminary discussion

This section addresses relevant aspects and notions which are required to understand the subsequent classification.

2.1.1 Systems of systems

Systems of Systems (SoS) are very large and complex systems [18], composed of complex subsystems. The *entwined* nature of the systems' multiple components

limits the success of a standard divide-and-conquer approach [19]. Classical methodological approaches neglect or are unable to fully capture the sources of emergence and evolvability in distributed networks.

2.1.2 Modularity and its limitations

Modularity is a well-known way to divide a large system into parts which can be individually designed and modified. The modules can then be assembled stepwise, and the system's functionality verified accordingly [20]. This works very well for reducible systems. Modularity is closely related to **reductionism**, which reduces the system to the sum of its parts, and goes contrary to the principle of emergence. Reductionism is only valid if the parts are unrelated (which is rarely the case). Nevertheless, analysing the parts can be helpful: they are easier to understand, and their sum gives an idea of the whole, even if incomplete [21]. Also, modules are useful as building blocks to create systems which may or may not exhibit emergent behaviour.

However, in the case of complex systems, it is wrong to assume that the behaviour of the whole system could be reduced to the sum of its parts [22]. The parts are often strongly dependent on each other and interact in multiple ways. Therefore, one of the challenges of complexity engineering is how to integrate modularity into a framework which can cope with emergent behaviour (if this should be possible). The effects of *composition* arise where many, in themselves often simple, entities interact to form a system, the resulting behaviour is usually not simply the linear sum of the behaviour of the individual components [23].

2.1.3 Trade-off: creative freedom versus specification

One of the challenges in engineering is the trade-off between the system specification[1] by the designer and the creative freedom of the system [4, 24]. More freedom means less control over the system's behaviour.

Engineers need to find ways to delimit the system's behaviour while still allowing it sufficient creative freedom to localise solutions in an adaptive way. Indeed, most systems exhibit certain patterns of behaviour, which is enough to make predictions about system behaviour. Consequently, any state belonging to the delimited patterns are acceptable. In other words: to assure that the system will not show undesired behaviour, the system can be bound to a *virtual box*[2]. Inside this box, the system is free, but it may not leave the box (Figure 2, for further discussion see also [26]). While the behaviour is inside the range specified by the *desired* box, no actions need to be taken. If the system leaves this area and remains inside the *allowed* area, no drastic measures need to be taken, but the system should eventually steer itself back towards the desired area. In case a system should diverge take states which are *possible*, but not allowed, immediate actions are necessary to bring the system back on save grounds. The fact that a system even reached this non-allowed area already means that the working parameters or policies need to be adjusted. The difficulties here are:

- Finding the box or pattern which corresponds to the acceptable system behaviour. This means that the designer has to define the limits of what is acceptable, and then somehow relate one borderline to another.
- Describing the box or pattern in a coherent way. Besides a description in natural language, in most cases also a computer-readable/-understandable version is necessary.
- Agreeing a compromise between the normally acceptable system behaviour and the additional freedom we can concede to the system. For instance, normally a mobile robot may not be allowed to enter a certain area of the shopfloor. Nevertheless, allowing it to do so under cer-

[1] In this section, the term *specification* is not used with its meaning in computer science, but rather its meaning in manufacturing engineering; a specification is a description of the identified requirements.

[2] Another way of expressing this is: 'The behaviour of a complex system will be a combination of pre-set objectives and constraints as defined by the system developer, and adaptive *islands* where the system is allowed to make its own decisions' [25].

tain circumstances may enable the other mobile robots to execute their task in a more efficient way, and thus it may improve the overall system performance. This is why the designer must find a balance between the benefits and potential dangers of crossing the border of acceptable system behaviour.

- Staying inside the box.

To conclude, the trade-off between creative freedom and specification is an important issue, and further investigation is necessary.

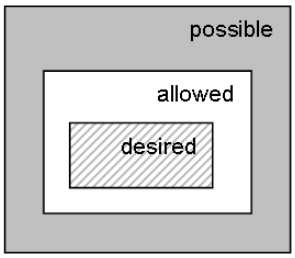

Figure 2: Desired, allowed and possible areas (*boxes*) of system behaviour

2.1.4 Complexity science versus complexity engineering

Complexity science, the study of complex systems, has seen an increasing interest in the last decades, pioneered by the Santa Fé Institute in New Mexico, USA [27]. Ever since, characteristics of complex systems in diverse areas have been thoroughly studied. Only few authors, however, take an engineer's perspective towards using the findings of complexity science for designing systems, even though the term *complexity engineering* appeared already in 1986 in the context of pattern recognition in *cellular automata* [11].

As we use it today, complexity engineering aims at the concrete use of complexity-inspired methods for engineering. 'In complexity science, one looks for underlying and unifying principles among many systems. In complexity engineering, we look into these different systems and their underlying principles from the point of view of application' [4].

Complexity engineering has not been established as a proper discipline yet. Literature about methods or frameworks is still scarce. One of the reasons may be basic misunderstandings over common terms such as emergence, for instance. The seeming contradiction between engineering and emergence may arise because engineers freely move from so-called *predictive* definitions, in which emergence is equated to surprise, towards definitions of *strong emergence* where higher-level patterns can be used as design templates [28]. To overcome such problems, the broader dissemination of clear definitions (see section 3.4) is important.

The techniques and concepts from *complexity science* need to be formalised in order to be usable in engineering [4]. Most complexity research is still in an early stage of development, in the 'trial and error intuitive engineering phase'. So far there are almost no methodologies, no common language and no common body of experience. Only a collection of examples, methods and metaphors for modelling complex, self-organising systems exist. Integrated theoretical foundations are still lacking [29].

This situation is a strong motivation to our endeavour to establish complexity engineering as a broadly known discipline, to deliver useful definitions, and to give an overview of the existing methods and approaches.

2.2 Classical engineering

Classical engineering is what is taught in most common engineering courses at universities, and what most scholarly books transmit. The engineering sciences have centuries old traditions, and in comparison, complexity science is relatively recent. It is still mostly considered as somewhat alternative, and we therefore refer to the 'old' school as *classical* or *traditional*.

Such engineering is essentially applying methods and tools to solve problems using a reductionist approach, be it top-down or bottom-up. This means that 'what you see is what you get', and there is no space to consider concepts like emergence.

Classical engineering, the majority of scientific models as well as our intuitive understanding are based on reductionism or analysis, predictability and

objectivity, determinism, correspondence theory of knowledge and rationality [17]. An example could be the *divide and conquer* strategy of software development strategies and Newtonian mechanics [29]: a problem is cut into its simplest components, and each of them is treated separately. They are described in a complete, objective and deterministic manner. Once each of them is resolved individually, then they are joined and, voilà, the entire problem is solved. Most engineers would indeed make the (often reasonable) hypothesis that the parts interact in some well-known and predefined ways without further influencing each other. Modularity then works at its perfection.

For application examples of classical engineering see Table 2.

2.3 Classical systems engineering

Systems engineering, as described by the *international council on systems engineering (INCOSE)*, is an interdisciplinary approach focusing on all aspects of systems. It considers all phases of a product, from its concept to production, operation and disposal, as well as all the involved parties, such as suppliers, manufacturers and customers. Although systems engineering attempts to consider the entire system instead of only parts of them, it is indeed a classical engineering approach because it ignores emergence and related concepts.

Systems engineering emphasises the importance of managing the whole as well as its parts, of seeing the interconnectedness of decisions, of taking a collective view.

For application examples of classical systems engineering see Table 3.

2.4 Complexity engineering

Complexity engineering is the creation of systems using tools originating from complexity science. The question is not so much in which ways complexity engineering would be better than classical engineering, but rather, in which situations classical engineering comes to its limits, and complexity engineering can help. This is mostly the case with complex systems (discussed in section 3.1): systems which are composed of many interacting components, where the interactions are multiple and changing in time; open systems; systems which have to function in a dynamic environment and strongly interact with it. Complex systems use adaptation, anticipation and robustness to cope with their often unpredictable environment [17], and complexity engineering therefore requires tools which take these issues into account.

Such systems, said to have *emergent functionality* [30], are useful in cases where there is a lot of dependence on the environment and it is difficult or impossible to foresee all possible circumstances in advance. Traditional systems are therefore unlikely to be able to cope with such conditions. Systems with emergent functionality can be seen as a contrast to reducible systems and usually hierarchical functionality; the latter means that a function is not achieved directly by a component or a hierarchical system of components, but indirectly by the interaction of lower-level components among themselves and with the world. Careful design at micro-level leads to behaviours at macro-level which are within the desired range.

Typically, no single entity within the system knows how to solve the entire problem. The knowledge for solving local problems is distributed across the system [17], and together, the entities achieve an emerging global solution. The right interactions need to be carefully engineered into the system, so that the systems self-organising capabilities serve our purpose, i.e. they do satisfy and support the requirements [4].

Complexity engineering will not lead to systems which are unpredictable, non-deterministic or uncontrolled. The output (i.e. certain aspects) may be predicted and controlled - it is how the system arrived to that output that can not be known, complex or not computationally reproducible [4]. However, it remains an open question if the latter is acceptable for all application domains. The system's development cannot be completely separated from the system's operation in the case of a complex system [1].

For application examples of complexity engineering see Table 4.

Engineering area	**Application example**
Software engineering	planning algorithms
Mechanical engineering	construction of a crane
Electrical engineering	hierarchical control of a machine
Production engineering	dedicated assembly station
Biotechnology	fruit fly breeding

Table 2: Application examples of **classical engineering**

Engineering area	**Application example**
Software engineering	large database systems with several subsystems
Mechanical and electrical engineering	cars, trains, ships, air planes
Production engineering	dedicated assembly line
Biotechnology	production of vaccines

Table 3: Application examples of **classical systems engineering**

Engineering area	**Application example**
Software engineering	peer to peer systems, grid
Mechanical and materials engineering	machines made of intelligent' materials, which recognise when parts undergo too much strain
Electrical engineering	traffic control
Production engineering	individual evolvable assembly systems
Biotechnology	tissue engineering, growing organs in the test tube

Table 4: Application examples of **complexity engineering**

2.5 Complex systems engineering

In contrast to classical systems engineering, which treats reducible systems, complex systems engineering will apply the methods from complexity engineering to systems of systems. Complex systems engineering [31] is appropriate to address problems which are continually changing or which require concepts at multiple scales or levels to be fully understood. The notion of *higher* and *lower* scales of conceptualisation give rise to the metaphor of a *ladder of scales*, in contrast to the often-used concept of a *hierarchy of scales*.

Complex systems engineering is typical for cases where systems of systems constantly evolve, where different parts integrate or compositions dissolve at any instant. There is both internal competition and collaboration which stimulates evolution. Specific outcomes of complex-system development cannot be specified in advance. But they can be shaped (i.e. strongly and persistently influenced) [32], for instance by guiding policies as used in MetaSelf [33].

For application examples of complex systems engineering see Table 5.

2.6 Inspiration from nature

Not only complex systems, but also nature in general inspires many researchers and engineers. The following classification attempts to structure this broad field.

Bio-inspiration in technology can take various forms. Each of them has particular goals and strategies, and researchers should be aware of them. The items 1, 2a and 2b on the following list correspond to the three research phases of inspiration by nature described in [6]. The last three items are additional. Table 6 gives an overview of inspirations and applications.

(1) **Using technology to understand natural systems:** Biologists, chemists and physicists have for a long time been using technological tools to help them investigate natural systems and to verify the established models. The palette of such tools includes oscilloscopes, gyroscopes as well as compound pendulums. More recently, computers allowed researchers to run large-scale simulations with thousands of iterations. Even more sophisticated, nowadays researchers use robots to emulate natural systems, and they even succeed in incorporating robotic 'cockroaches' into real cockroach swarms [35, 36].

(2a) **Using ideas from natural systems to make lab experiments and find usable mechanisms:** This refers to the experimental phase of *bionics*. Researchers understood long ago that they can learn from nature and use mechanisms discovered in natural systems to solve engineering problems. However, most mechanisms need to be adapted in order to be usable, and this can only happen through an experimentation phase in the lab. Different versions are often discovered by changing the initial mechanisms, and the researchers can let their creativity play.

(2b) **Using ideas from natural systems to build industrial technology:** The final goal of most bionic developments is using them in real-world applications. This means that they have to comply with industrial standards. It has been achieved for many technologies, such as ultrasound, radar and sonar systems, dolphin-shaped boats, ultrahydrophobic and self-cleaning surfaces based on the Lotus effect, and cat-eye reflectors. Researchers now increasingly approach distributed and autonomous adaptive systems, which are more difficult to build than other bionic applications.

(3) **Using the 'engineering toolbox'[3] on natural systems:** Denominated *biotechnology, biomedical engineering, genetic engineering* or similar, these disciplines use engineering technology on natural substrates such as living cells, bacteria and sometimes higher animals. Researchers grow virus cells in tanks to produce vaccines, they try reproducing epidermic tissue and inner organs or genetically modified animals. Many different technologies are being used to diverse purposes. As a specific example, when a certain gene is implanted and then inherited to future generations, cancerigenous cells can become fluorescent, which facilitates their identification under the microscope[4].

[3] Safe synthetic biology, see `http://www.synbiosafe.eu`

[4] Explanations from the **Biotechnology** lectures (academic year 2004-2005) by Prof. Florian Wurm at the Swiss Federal Institute in Lausanne, Switzerland

Engineering area	Application example
Software engineering	self-organising displays [34]
Electrical engineering	large scale traffic management
Production engineering	evolvable assembly systems including their supply networks and customers
Biotechnology	man-made biological ecosystems
Robotics	open mobile robots coalitions

Table 5: Application examples of **complex systems engineering**

(4) **Using biotechnology methods for software engineering:** Researchers in computer science now often take inspiration from methods used in biotechnology, in particular in cell engineering. Methods which work for living cells supposedly also work for software agents.

(5) **Using ideas from engineering to build new models for understanding natural systems:** Probably the most recently initiated discipline considers architectures and mechanisms used by engineers to create technological systems which have nothing to do with natural systems. Natural scientists then use such ideas to build new models for understanding natural systems [37], in the sense that if engineers have come up with ideas, maybe nature has invented them long ago.

Complexity engineering, as treated in this article, belongs to the class 2a / 2b in the sense that it uses inspiration from nature for engineering.

3 Definitions and terms

Many of the terms discussed in this article are often used with an intuitive understanding in colloquial speech. Also in scientific work, they take varying meanings. The following subsections discuss the definitions of these terms in scientific use.

Agents: By *agent* we refer to an entity which is able to act in a fairly autonomous way, according to norms / rules / policies and in order to achieve a goal. An agent can be something like an ant, a human person, a robot or a software agent. It consists of some kind of brain or computational power and often also has some kind of embodiment.

Systems: As a working definition, a system may be considered as a set of entities (often agents), which interact with each other as well as with the environment, and some infrastructure or passive components / entities.

3.1 Introduction to complexity

Complexity issues have been studied within various contexts, for instance: physical phenomena [38], cellular automata [11, 39], ICT systems [19], supply chain networks [24], management [40], networks [3, 41], modelling [42], the laws of diversity [43], natural disasters such as earthquakes [44] and epidemics [45], adaptation in [12, 13, 14, 15], the dynamics of complex systems [46], chaos theory [47], and engineering aspects [48, 49, 50, 51, 22, 52, 53, 54]. The search of the mechanisms behind emergence and self-organisation has also been approached by many complexity researchers, such as [55, 56, 57, 30].

In some way, many open questions are related to each other, and common characteristics can be identified when investigating, for instance, how often earthquakes of a certain strength happen, why certain neighbourhoods become dangerous, how and why epidemics spread, through how many degrees of separation we are linked to any other person in the world, etc. The study of non-linear systems, dynamic systems, differential equations, non-determinism is intimately related to the nature of intelligence, the creation of structure and organisation, the creation of life, emergence, self-organisation, the micro- and the macro-scale etc. Table 7 places complexity between deterministic and statistical science, in terms of the scope in time and numbers of entities considered.

CALResCo [10] is a valuable source for all kinds

Phase	Inspiration / assisting tools	Application / goal
(1)	Technological tools	Understanding natural systems
(2a)	Natural systems	Lab experimentation
(2b)	Natural systems	Industrial engineering, technology
(3)	Engineering methods	Biotechnology on living substrates
(4)	Biotechnological methods	Software engineering
(5)	Software engineering methods	Building artificial models to understand natural systems

Table 6: Engineering and natural systems

of question concerning complexity science, which searches the laws that apply at all scales, the inherent constraints on visible order. Typical systems may be described as: *'Critically interacting components self-organise to form potentially evolving structures exhibiting a hierarchy of emergent system properties.'* This is a confirmation that the study of complexity science may prove to be useful for agile manufacturing.

The study of complex systems requires a conceptual framework which should include three different perspectives [58]: non-linear dynamics and chaos theory, statistical physics including discrete modelling, and network theory, which is especially useful for understanding the Internet and other communication networks, the structure of natural ecosystems, the spread of diseases and information, the structure of cellular signalling networks, and infrastructure robustness.

Some authors' strategy is to avoid complexity as far as possible, and they use metrics to determine the degree of complexity of a given configuration [59]. Most researchers, however, aim at gaining a better understanding of complexity before dismissing it as assumingly being disturbing or useless.

The assumption that principles and mechanisms which are successful in nature will also work in technology / engineering is not undoubted. Besides the numerous similarities they share [6], there are also important differences between nature and engineering [60]. Namely:

- In nature, there is time and space for failures. In engineering, we must get it right the first time (or at least very soon after a test phase), and we must avoid failures.
- The main goal is (supposedly) only survival of the species. In technology, we have very specific goals.

Taking these differences into consideration is certainly sensible.

3.1.1 Complexity definitions

For instance, industrial assembly systems are complex; there is a plentitude of often conflicting interests and objectives being pursued, and the overall behaviour of the system results from the behaviour of many individual components which mutually and multi-laterally[5] influence each other. Complexity science is an area of research which studies exactly this kind of systems, and is therefore potentially a useful tool for assembly engineers. To understand how complexity might help, it is necessary to understand complexity itself - which is not evident, especially as it comes in many different flavours, depending on both the field of research and the researcher.

Complexity can be defined as 'the name given to the emerging field of research that explores systems in which a great many independent agents are interacting with each other in many ways' [27]. Examples of such systems [21] could be electrons and molecules, which require cohesive and disruptive forces to work the way they do. Instantiations of this principle are ordering and disordering forces, kinetic energy and binding energy, coherence and disruption, transaction cost and administrative cost, etc.

Quite different sounds this definition:

[5] *Multi-lateral* describes a relationship with several peers at the same time.

Short term	Mid term	Long term
1, 2, 3, ... a few	Thousands, ten-thousands, ...	Millions, billions, ...
Deterministic Science	Complexity Theory: anything having interacting parts and undergoing continuous change	Statistical Science, Quantum Theory

Table 7: Complexity

> [Complexity is] 'that property of a language expression which makes it difficult to formulate its overall behaviour, even when given almost complete information about its atomic components and their interrelations' [61].

Various researchers have tried to classify complexity types:

- Random complexity, probabilistic complexity, deterministic chaos, emergent complexity and Newtonian dissipative structures [62].
- *Effective complexity* versus *underlying simplicity with a certain amount of logical depth*, which may also seem complex [23].
- Complexity can also be classified by the following characteristics [63]:
  - Time-related: static or dynamic.
  - Organisational: process-related or structural.
  - Systemic: internal or external.
- The *external complexity* [64] is the amount of input, information, or energy obtained from the environment which the system is capable of handling. The *internal complexity* is the complexity of the input representation which the system receives. Complex systems often increase their external complexity to reduce their internal complexity.

Complexity is characterised by non-linear relationships between parts, openness, feedback loops, emergence, pattern formation, and self-organisation [65]. In linear systems, effect is directly proportional to cause, whereas in non-linear systems, the effect may be any. Non-linearity comes in many flavours, tending to occur when a system's interactions are multiple, ecologically embedded, non-additive, inseparable, heterogeneous, interactive, asynchronous, lagged, or delayed [19].

3.1.2 Complex systems

Complex Systems (CS) can be defined in various ways. Most scientists consider CS as being composed of a large number of relatively simple heterogeneous components, which interact multi-laterally and in changing ways; collective behaviour emerges. The interactions sometimes result in non-linear behaviour, and there are multiple feedback loops. Complex systems often evolve, adapt, and exhibit learning behaviours. They typically exhibit emergence and are often self-organised.

The original Latin word *complexus* signifies *entwined* or *twisted together* [66]. A complex system is thus made of more than one part, and the parts are at the same time distinct and connected. It is therefore inherently difficult to model them. Often, there are circular causal relationships: one part influences the other, which in turn influences the first, and so on.

Complex systems refer to 'a set of systems which share some common behavioural and structural properties', where the meaning of structure can be spatial, temporal or functional [65].

The micro-level interactions between parts of the system may either be independent or coherent, resulting in different collective behaviours [65]:

- Coherent interactions: coordination at micro scale only.
- Independent interactions: random behaviour at micro scale, coordination at macro scale.

- Correlated behaviours: coordination at micro and macro scale.

3.1.3 Complex Adaptive Systems

Systems which emerge over time into a coherent form, and adapt and organise themselves without any singular entity deliberately managing or controlling it, belong to the class of *Complex Adaptive Systems* (CAS) [14]. CAS are many-body systems, composed of numerous elements of varying sophistication, which interact in a multi-directional way to give rise to the systems global behaviour. The system is embedded in a changing environment, with which it exchanges energy and information. Variables mostly change at the same time with others and in non-linear manner, which is the reason why it is so difficult to characterize the system's dynamical behaviour.

CAS often generate 'more of their kind' [23], which means that one CAS may generate another. To characterise them, researchers describe their components, environment, internal interactions and interactions with the environment.

It remains open if there are complex systems which are not adaptive. Some researchers agree, as, depending on its definition, adaptivity may require diversity and natural selection, as shown in ecosystems [65].

3.2 Self-organisation

A well-known definition was suggested by Camazine et al. [57]:

> 'Self-organisation is a process in which patterns at the global system emerges solely from numerous interactions among the lower-level components. Moreover the rules specifying interactions among the system's components are executed using only local information without reference to the global pattern.'

The following definition is a few years more recent:

> Self-organisation is the **dynamical and adaptive mechanism or process** enabling a system to acquire, maintain and change its organisation without explicit external command during its execution time; there is no centralised or hierarchical control. It is essentially a spontaneous, dynamical (re-)organisation of the system structure or composition' [67, 68].

The identification of a boundary of the system is extremely important when deciding if a system is self-organising or not: defining an entity with controlling influence as external disqualifies a system from being self-organised, whereas the situation is different if the entity is considered as being internal.

By some researchers, self-organisation may also be seen as the spontaneous creation of globally coherent pattern out of local interactions [56] (although this is usually considered as the definition of emergence, see section 3.4). This shows how controversial the research area still is.

Preconditions for having self-organisation in engineered systems, based on characteristics discussed in [69, 70, 68, 56], are:

- Autonomous and interacting units.
- No external control; the question of corresponding system boundary definition arises.
- Positive and negative feedback. For instance monetary rewards / punishments for successful collaboration and achievement of tasks respectively contract breaching or failures.
- Fluctuations / variations which lead to the typical far-from-equilibrium state, which is in manufacturing systems given by disturbances and changing production requirements, such as changing volumes and fluctuating part deliveries or equipment down-times.
- Safety measures in case the system should drift towards undesired or harmful behaviour.
- A flat internal architecture, as opposed to a hierarchical one, with dynamically changeable organisation of the interacting agents.

Adaptation means achieving a fit between system and environment; thus every self-organising system adapts to its environment [56].

Mechanisms which lead to self-organisation in engineered systems include stigmergy (known from

social insects, such as ants releasing pheromones in the environment), gossip, trust, collaboration/competition, swarms (as seen in schools of fish or flocks of birds), and chemical reactions. Most of these mechanisms happen according to a set of rules which can be identified. For instance, the entities in swarms respect three principles, such as (1) advance, (2) stay close to your peers, and (3) avoid collisions. Depending on the case and the mechanism, the rules can be more numerous, more complicated, and more complex. For engineering purposes, they may be adapted and extended.

A working definition for self-organisation seen from an engineering perspective is given in section 3.2.2.

3.2.1 Weak and strong self-organisation

When it comes to concrete applications, it makes often sense to differentiate between *weak* and *strong* self-organisation. Not all cases do fully comply with the rules' of strong self-organisation, but still, there is some form of self-organisation.

In the *strong* case, the self-organisation happens without any centralised control, whereas in the *weak* case, there may be some internal (centralised) control or planning [71].

3.2.2 Working definition of self-organisation

After studying the existing definitions in literature as well as an engineering perspective on complexity concepts, we suggest the following *working definition*, based on the research done and experience gained in the scope of our work [8]:

Self-organisation: Systems which self-organise are typically composed of many, at least partially autonomous components. These components have certain characteristics and skills, and have at least one way of communicating with their peers and the environment. The environment dynamically changes and influences the system. The components engage in interactions with their peers; they may collaborate, compete, negotiate, gossip, and establish varying levels of trust between each other. This depends on the mechanism which leads to self-organisation. The components may have individual goals, but also shared or global goals. The system is not under any type of external or central control, although in engineered systems, the self-organisation process happens according to certain rules which were defined by the system designer. These rules may be dynamically changed, even at run-time, and thus allow the designer to influence the system at any time. Self-organisation is scalable, robust, and fault-tolerant, i.e. insensitive to small perturbations and local errors as well as component failure, thanks to redundancy. Self-organising systems exhibit graceful degradation, meaning that there is no total break-down because of minor local errors. Self-organisation is a **dynamic process** in many-body systems and may occur with or without emergence.

3.3 Self-* properties

In literature, diverse interpretations of self-organisation, self-adaptation, self-healing, self-management, self-(re)configuration and emergence can be found. Many of them focus on one single term; only few mention the links between the concepts. For instance, self-adaptation is included in self-managed systems, and self-management is included in self-organisation, according to the classification in [72].

One important differentiation to be made is the direction of the property: self-organisation and self-healing are bottom-up, whereas self-adaptation, self-management and self-healing are top-down, as illustrated in Figure 3.

Besides the differences in the orientation (bottom-up or top-down), most often the name given to the property is a question of the focus: the behaviours can sometimes not even be clearly classified as 'pure organisation' or 'pure healing' etc., and most often self-* properties have an emergent character. As an example, when a system re-organises its internal structure to recover from a failure, is this self-organisation or self-healing? Is self-organisation used for self-healing? Or is it emergence, because the process is based on local rules and produces a new, global result? It depends on the rules which define the behaviour, but an observer may not know them.

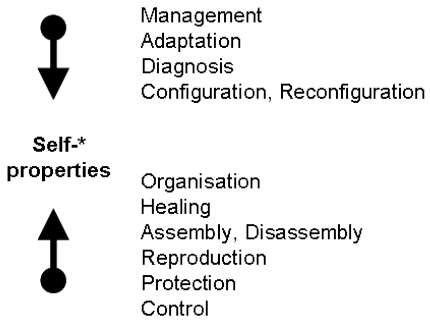

Figure 3: Bottom-up and top-down self-* properties

De Wolf suggested a taxonomy of self-* properties [73] which focuses on *decentralised autonomic computing* and discusses characteristics of self-* properties and implications for their engineering. Among other criteria, the taxonomy considers if a self-* property is achieved on macroscopic or microscopic level, if it is on-going or one-shot, if it is time/history dependent or independent, if it evolves in a continuous or abrupt way, and it is adaptation-related or not. The taxonomy gives examples of mechanisms leading to self-* properties and classifies application examples according to the considered characteristics, but it does not indicate to which kind of self-* property a mechanism or application belongs.

Even though the following classification is general, the following non-exhaustive list of working definitions is influenced by the domain of robotics and artificial intelligence. These working definitions [6] are not conclusive, but they give indications and contribute to a base for further research: they intend to trigger other researchers to reflect about them. The term *self* generally refers to the absence of external control. After a general description, an application to assembly systems follows.

Self-adaptation: a system adjusts itself to changing conditions without major physical modifications. ≫ For instance, in the case of an industrial assembly system [8], when more urgent orders arrive, a robot can increase its working speed.

Self-configuration: a system prepares itself for functioning, including the adjustment of parameters and calibration. ≫ A robot adjusts its movement accuracy to the desired value.

Self-reconfiguration: mostly encompasses self-adaptation and self-configuration, but also some physical change (including software and hardware). ≫ When a conveyor module fails, and there is an alternative conveyor path to reach the affected destination, the modules adapt their behaviour and use the alternative path until the module has recovered from the failure. Alternatively, a new conveyor module is requested from the user and integrated into the existing system.

Self-organisation: a system creates or adapts its own structure to reach a goal. ≫ Modules form coalitions to provide the requested skills.

Self-assembly: sub-systems or modules connect with each other to form the whole. ≫ A robot self-assembles with a gripper which it can autonomously pick up from its toolwarehouse.

Self-disassembly: a system decomposes itself into sub-systems or modules. ≫ A coalition which is not necessary any more disassembles. For instance, a robot will place its gripper back in the toolwarehouse.

Self-diagnosis: modules can find out and state what is wrong with themselves. ≫ A feeder which cannot provide parts will check if there are no ready parts inside, or if there is a blockage, or if there is any other problem preventing normal functioning.

Self-repair / self-healing: a system can treat its problems and maintain or re-establish functionality. ≫ A blocked feeder will restart its software, execute calibration movements, and if still blocked, ask the user for help.

Self-reproduction / self-replication: a system can create a copy of itself. ≫ A module coalition incentivises suitable modules to form the same type of coalition.

Self-protection: a system can protect itself from intruders or attacks. ≫ In case an assembly system was open enough for strangers to gain access to it, for instance over the Internet, it would need to protect itself from harm.

Self-control: the system steers itself. ≫ The modules control their own behaviour, for instance

guided by policies.

Self-management: a system can take care of itself. This may include self-protection, self-healing, self-configuration, self-optimisation, self-adaptation etc. ≫ At production time, the modules maintain themselves as well as their neighbours in good conditions. They manage their multi-lateral interactions, provide the requested services, schedule maintenance etc.

3.4 Emergence

Emergence describes how order appears out of chaos [15]. Both emergence and self-organisation (section 3.2) are concepts which first appeared in physics (phase transitions) and chemistry (molecules and material properties), and were then also observed in other domains, including biology (cells, DNA, brain, etc.), game theory, social science, economics and engineering. A general theory of emergence is still missing [74].

Most systems which exhibit emergence can be modelled in terms of the interaction of agents. Building blocks are combined to form a higher level system. Emergent phenomena are often hierarchical: complex ones are composed of simpler ones [14].

Definitions found in literature include:

Emergence is a bottom-up effect, which generates order from randomness [75]. It results in a self-organised increase of order, in space or time. A global behaviour arises from the interactions of its local parts; cannot be traced back to the individual parts [70]. Desirable and undesirable emergent behaviour in distributed systems results from the non-linear interaction of completely deterministic processes [76]. None of the entities composing the system knows how to achieve the emergent phenomenon [68].

Although controversial, emergence does not only exist in the eye of the observer; it is intrinsic to the system [15]. Novelty does not depend on the experience of the observer, neither. It refers to the new class of words used to describe the global phenomenon, new in the sense of different from those used for the local level description. However, novelty is not the same as surprise, as surprise is related to the preparation of the observer, and novelty is not.

According to Holland [15], for engineered systems, emergence happens according to rules. The designers have to find the level of detail where they can set the rules and therefore control emergence. Notice that also this is a very controversial statement, as for most other researchers, this describes self-organisation, and not emergence.

It is mostly agreed that an emergent property [21]:

1. of a whole is not the sum of the characters of its parts.
2. is of a type which is totally different from the character types of its constituents.
3. is not deducible or predictable from the behaviours of the constituents investigated separately.

A *resultant* is different from an *emergent* [21]: A resultant is closely tied to the material content of the constituents. Linear systems have resultant behaviours and are traceable. The principles of superposition, aggregation and additivity apply. An emergent has a structural aspect, there is novelty and non-additivity. For instance, conductivity is resultant, whereas superconductivity is emergent. Nevertheless, both properties involve the same 'ingredients'. Emergent properties can in principle be predicted by analysing the lower levels; in practice, we are not always capable of doing it [10].

Different forms of emergence [77] exist: *Diachronic:* develops in time. This may happen when new technologies are introduced and they combine with previously existing modules. *Synchronic:* different ways of looking at a given info from one level to another, e.g. emergence of a significant pattern, structure or form from the point of view of a given observer. *Descriptive:* synchronic, but not related just to the observer's conceptualisation and description; *objective* emergence if causal effect on environment. May occur when new system behaviours cause the user to take previously not necessary actions. *Cognitive:* becoming aware of previously ignored knowledge. A system designer or user may experience this.

Some authors consider that not only system characteristics may emerge, but also goals [78] and functionalities [79, 30]. In the context of engineering, this

may be interpreted as systems which can do things they were not made for.

A working definition for emergence seen from an engineering perspective is given in section 3.4.2.

3.4.1 Weak and strong emergence

To bring the classical notions of emergence, discussed before, closer to the reality of engineered systems, two classes of emergence are proposed [16, 80]:

For *strong* emergence, the global level must show further development. There is non-linear dependence of the global functionality on the components and their interactions between themselves and the environment.

Weak emergence means that the local-to-global dependence may be *quasi-linear* - but still, the appearance of the global phenomenon is not self-evident and needs some kind of *inspiration*.

> 'A macrostate is weakly emergent if it can be derived from micro-states and micro-dynamics but only by simulation' [81].

3.4.2 Working definition of emergence

After studying the existing definitions in literature as well as an engineering perspective on complexity concepts, we suggest the following *working definition*, based on the research done and experience gained in the scope of our work [8]:

Emergence: Systems exhibiting emergence most often consist of at least two different levels: the *macro-level*, considering the system as a whole, and the *micro-level*, considering the system from the point of view of the local components. Local components behave according to local rules and based on local knowledge; a representation of the entire system or knowledge about the global system functionality is neither provided by a central authority nor reachable for the components themselves. They communicate, locally interact with each other and exchange information with the environment. From the interaction in this local world emerge global phenomena, which are more than a straight-forward composition of the local components' behaviours and capabilities. Typically, there is a two-way interdependence: not only is the global behaviour dependent on the local parts, but their behaviour is also influenced by the system as a whole. Nobody in the system knows how to achieve the emergent phenomenon, and nobody has complete knowledge of the system or a global observer's perspective. An emergent phenomenon is a **structure or pattern, visible at global level**.

3.5 Chaos

A system may be viewed as *deterministic* if the current state(s) of the system determine its future state(s) in the presence of random noise, environmental inputs and unknown initial conditions; a deterministic dynamic system whose behaviour is hard to predict is called a *chaotic* system [65].

Chaos in common language means confusion or the lack of fixed principles, whereas chaos in mathematics is behaviour according to certain rules [21]. The methods for mathematically describing chaotic behaviour founded by Poincaré and Lorentz bring structure into seemingly random behaviour [27].

Chaos is different from randomness: chaotic systems behave according to strange attractors. This means that under a set of conditions (i.e. within the attractor basin), a system will always move towards a certain state or set of states. To leave them, the system requires a certain energy input (disturbance). In mathematical terms, chaotic systems are deterministic, whereas randomness has no structure at all.

In complexity terms, *entropy* is the tendency of systems to create chaos from order, while *extropy* is the tendency of systems to create order from chaos (that is, emergence) [10].

Chaotic systems have been discovered in domains as diverse as mathematics, physics, biology, chemistry, meteorology, fluid dynamics, astronomy and statistical mechanics and logistics [82]. Also industrial assembly systems exhibit chaotic behaviour:

- cause and effect are not always in a linear relation, as a small perturbation may cause a total system breakdown;

- a successful assembly system will tend towards an attractor which stands for the correctly assembled product, although it may assume different states on the way there;
- assembly system behaviour is bound to certain limits (robots cannot suddenly start doing crazy things), although within the given boundaries, the behaviours may vary (different robots may dynamically take over the insertion of a bolt, according to their availability and performance).

3.5.1 Sensitivity to initial conditions

The *butterfly effect* stands for sensitive dependence on initial conditions. Little causes do not necessarily lead to little effects, and big causes to big effects. Future outcomes are arbitrarily sensitive to tiny changes in conditions [23]. Simple mechanisms may cause considerable complexity, as well as complex sources may lead to simple phenomena. In the same sense, complex systems can give rise to turbulence and coherence at the same time. Brought to a simple formula, we may say: *'In the middle of chaos, there is order. In the middle of order, there is chaos'* [83].

Manufacturing systems often exhibit sensitivity to specific conditions and to disturbances. Certain factors, like energy disruptions or an abnormal increase of temperature and humidity may lead to system breakdown, while others have no significant effect (for instance, the occurrence of extreme noise would disturb human operators, but not bother robots). Some disturbances may have consequences in some cases, but lack any effect in others. For example a robot using optical sensors reacts sensitively to changing light conditions, whereas a robot working with tactile sensors remains unaffected.

3.5.2 Edge-of-chaos

Various terms are being used for the state somewhere between stable order and chaos (see Figure 4), among others: *dynamic order*, *instability in order*, and *self-organised criticality* [44, 45].

Constantly stable equilibrium states would block evolution. Dynamic systems get again and again into states where a little stimulus can trigger a major reaction. This gives the systems energy to evolve and makes new phenomena emerge. 'The edge of chaos is a point between chaos and order when creativity and stability fuse, where living systems are at their most inventive, where there is the highest chance that something distinct and unique will emerge' [84].

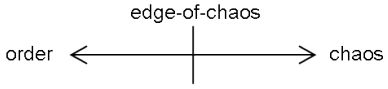

Figure 4: Somewhere between order and chaos

Analogies may be drawn between sand piles, earth quakes, wars, extinctions of species and revolutions [85]. The world organises itself into a critical state at the edge-of-chaos: small events can stay small or grow to enormous importance and have heavy consequences. The *power law* describes such phenomena. It expresses that if we double the energy (or any other quantity being studied), the probability of the phenomenon to appear is half, a quarter, etc. For instance, if the probability of an earthquake of strength x to happen is y, the probability of an earthquake of $2x$ to appear is $\frac{y}{2}$. All events can have the same kind of trigger. There is no fundamental difference between small and big events. No-one knows if the next event falls onto a *finger of instability*, which leads to its propagation, or if it will stay small and not propagate. When building models of such events, it is often possible to greatly simplify. Researchers will find the same power laws in their models as in reality, if the fundamentals of the models are correct. It is at the edge-of-chaos that epidemics do or do not spread [45].

Failures and perturbations in manufacturing systems often follow power laws as well. This is why the systems must be able to cope with frequent small failures as well as with big rare ones.

3.5.3 Phase space / state space

Phase space or *state space* diagrams are used to represent the behaviour of systems, with all the states which are reachable for a system, and the transitions

in-between, as a function of system parameters. The bifurcation diagram shows where the previously uniform behaviour of a system separates into different directions, and possibly diffuses into an unlimited number of different behaviours.

An attractor is a state towards which a system will always tend, as long as it is under a set of initial conditions. The *Lorentz attractor* and other *strange attractors* describe systems which never quite settle into a state, but eternally oscillate within a certain range of states, never taking the same state twice [86]. If we know a system's strange attractor and its dimensionality (the number of dimensions of the corresponding state space), we can make predictions about the systems behaviour, for instance the performance of an automated production line [86].

3.5.4 Noise, perturbations and local maxima

Perturbations are a challenge and a chance at the same time. Systems must cope with perturbations and not let themselves drift away from their normal functioning. However, perturbations can also be helpful: systems which require some kind of optimisation may tend to be stuck in local minima and thus not be able to evolve towards better solutions without the system being disturbed or otherwise stimulated.

The *cybernetic law of requisite variety* by Ashby [43] teaches that the greater the variety of possible perturbations, the greater the variety of controlling actions it needs. This means that a system which is always perturbed in the same way, will always require the same corrective measures. However, if there is a plenitude of different influences on the system, it will need a correspondingly varied set of ways of reacting.

Complexity engineers should try to use perturbations to their benefit. For instance, when a robot fails and other robots resolve to collaborate in an unusual way to cope with the failure of their peer, this new collaboration may be discovered as an efficient way of executing the task, and thus be retained for further use even after the peer's reparation.

3.5.5 Further concepts

The concepts described in this subsection have not been explained yet but are important for the general understanding of complexity science and chaos theory.

Fractals: Inspiration for fractal manufacturing systems [87]. Fractals have a self-similar structure at arbitrarily small scale, meaning that new similar structures appear when zooming in; self-similarity may also be stochastic or approximate.

Attractors basin: Like a river has a watershed basin that drains to it, every attractor has a basin. Of particular interest are the basin boundaries, which are often fractal.

Fitness function/landscape: Organisms must be fit for survival and thus react to the requirements of the ever-changing environment. These requirements can be described by a fitness function. The closer an organism matches the fitness function, the better adapted it is to the current life condition. The criteria for endurance or elimination of new characteristics are most often multiple and form a *fitness landscape*.

Spontaneous order: Complex systems can spontaneously organise themselves into coherent patterns. Conflicting constraints lead to a *rugged* fitness landscape, which means that the fitness parameters do not evolve linearly/smoothly. The ruggedness is determined by the internal organisation of the organism.

Convergence: Happens when a system tends towards the desired state / solution. If the system cannot reach it, no matter how long it runs (it may oscillate endlessly, or tend towards an undesired state, such as a chaotic attractor), the system does not converge. The **speed of convergence** [26] describes how quickly a system reaches the desired state.

Downward causation: Influence of the global / macro system on its components / the micro elements, derived from the constituents' self-organisation. Also an emergent phenomenon can exhibit downward causation, that is, the emergent phenomenon influences the elements which lead to the emergence.

Equilibrium: Self-maintained state of a (partially) isolated system. Equilibria can be stable, metastable, unstable quasi-stable, local / relative or global / absolute.

3.6 Dependability, robustness and similar terms

The terms *robustness*, *dependability*, *resilience*, *redundancy*, *degeneracy* and *graceful degradation* are often used in the same context: they all refer to how a system copes with failures and perturbations.

Dependability is the ability of a system to deliver a service that can justifiably be trusted [88]. For instance, a cash machine must always provide the same service, and we must be sure that nothing else happens when we are requesting a certain amount of cash. Central to this definition is the notion that it is possible to provide a justification for placing trust in a system. In practice this justification often takes the form of a dependability case which may include test evidence, development process arguments and mathematical or formal proof.

The original meaning of **resilience** refers to the maximal elastic deformation of a material. In the context of computer science and robotics [89, 90], resilience means *dependability when facing changes*, or in other words, its ability to maintain dependability while assimilating change without dysfunction. In the case of MetaSelf [33], a key feature for dynamic resilience is the availability of dependability metadata at runtime. For instance, for dynamically attributing a new server, it is necessary to know the dependability values of the servers in question.

Dynamic resilience is a system's capacity to respond dynamically by adaptation in order to maintain an acceptable level of service in the presence of impairments' [90], whereas *predictable dynamic resilience* refers to the capacity to deliver dynamic resilience within bounds that can be predicted at design time. Accordingly, for MetaSelf, *resilience metadata* is information about system components, sufficient to govern decision-making about dynamic reconfiguration. *Resilience policies* serve as guidelines for reconfiguration.

Stability means in manufacturing that a process always delivers the same result, as long as the conditions are within a certain specified range. A system must continuously deliver correctly assembled products and cope with perturbations or failures.

Robustness means that a system does not easily get disturbed in its normal functioning. It can cope with failures, changing conditions and is able to remain usable.

Redundancy means that there are more than one elements with the same functionalities in a system. It is the standard solution of engineers to cope with failures, and it involves structurally identical elements. Redundancy is costly, because the redundant resources remain unused, and therefore redundancy is often avoided as far as possible. Self-organising systems have typically a lot of redundancy, which leads to an inherent robustness against many failures.

Degeneracy [91] is an alternative which can be observed in natural systems such as the brain. In case of a lesion, structurally different brain regions can adapt to take over the tasks of the damaged area. The same can also be achieved in technological systems: for instance a robot may request new coalition partners to form composite skills, which allow them to take over the task of a failing original robot.

With **graceful degradation**, a damaged or perturbed system does not totally break down. It maintains at least part of its functionality, even if with reduced performance.

4 Complexity engineering approaches

This section is organised as follows: Concepts and principles are reported in section 4.1. Section 4.2 de-

tails mechanisms and patterns. Modelling and analysis are the subjects of section 4.3, and section 4.4 considers design approaches. Section 4.5 details architectures. Methods to develop and implement the designed systems are presented in section 4.6. Section 4.7 treats validation and verification. Finally, section 4.8 explains applied approaches.

4.1 Concepts and principles

This section reports a set of concepts and principles which are generally important when creating complex systems. They represent different perspectives which lead to different approaches.

4.1.1 Emergent functionality

Steels [30] defined emergent functionality (EF) as a function which is not achieved directly by a component or a hierarchical system of components, but indirectly by the interaction of more primitive components among themselves and with the environment. Each component's behaviour has side effects, and the sum of these gives rise to the EF. In order to achieve the desired effect, all the components need to be together and operate simultaneously. Systems with emergent functionality are useful when the dependence on the environment is important, and when it is difficult to foresee all possible circumstances in advance. Remark: what Steels called emergent functionality is nowadays often referred to as self-organisation; the arguments have mostly stayed the same.

4.1.2 Synthetic ecosystems

Creating systems based on the concept of synthetic ecosystems [74, 92] or digital ecosystems [93] is very useful for complexity engineering. Systems are considered 'alife' or 'life-like' and the integration of nature-inspired mechanisms follows almost automatically. The behaviour of species (often insects) and their interactions with each other as well as with the available resources serve as models for with multi-agent systems. The following design principles are suggested [74]:

1. Things, not functions: avoid functional decomposition, take real-world units instead.
2. Small agents: prefer many simple agents to a few complicated ones.
3. Diversity, heterogeneity: create agents with differing capabilities and characteristics.
4. Redundancy: the same capabilities should exist more than once, and there should be more than one way to solve a specific problem.
5. Decentralisation: create *proactive* agents and avoid centralised services.
6. Modularity: it should be possible to compose the system's functionalities stepwise, in layers. (Nevertheless, do not forget the limitations of modularity, discussed in section 2.1.2.)
7. Parallelism: solve problems in parallel and allow agents to participate in several coalitions at once.
8. Bottom-up control: local interactions lead to a global result, with no entity executing control from the top.
9. Locality: sensor-motor interaction is local, as well as the interactions between the agents.
10. Indirect communication: as far as possible, abstain from direct agent-to-agent communication. Passing messages through a shared environment allows communication to be decoupled in time.
11. Recursion, self-similarity: re-use successful structures and strategies at various levels.
12. Feedback, reinforcement: take into account the result of earlier actions.
13. Randomisation: introduce a random factor in agent decisions to avoid negative synchronism (e.g. all agents heading for the shortest queue at the same instant).
14. Evolutionary change: prefer gradual and evolutionary change to abrupt and revolutionary change.
15. Information sharing: inform other agents. Learn as individuals or as a society.
16. Forgetting: outdated information must disappear automatically.
17. Multiple goals: include maintenance-goals and achievement-goals. Design the system to be able to pursue various goals at once.

An additional design principle, added by the authors of this article, is the use of positive and negative feedback. Their interplay contributes to the system's convergence, oscillations or divergence.

These design rules summarise the most important principles which should always be applied when designing nature-inspired systems. In some cases, there may be reasons for making exceptions, such as having direct communication between the agents. The designer should be aware of the reasons and know that the choice to make an exception may cause difficulties under certain circumstances.

4.1.3 Distributed autonomic computing

De Wolf and Holvoet suggest that *decentralised autonomic computing (DAC)* can realise autonomic computing in a decentralised way, using emergence. Self-* properties are thus achieved collectively. They propose a taxonomy for self-* properties.

> DAC is achieved when a system is constructed as a group of locally interacting autonomous entities that cooperate to adaptively maintain the desired system-wide behaviour without any external or centralised control [73] .

DAC is achieved mainly through the implementation of collectively achieved self-* properties, which can be classified according to the following taxonomy criteria [73]:

- *'Micro versus macro' or 'local versus global':* Self-* properties can be of microscopic (local, concerning a single agent and its immediate vicinity), or macroscopic (global, concerning several agents / the entire system) scope. The way how *locality* is defined is determining for judging if a property is local or global. Additionally, a self-* property can be macroscopic in one system and microscopic in another: it depends on how it is implemented.
- *Ongoing versus one-shot:* most self-* properties are required over an extended time (e.g. maintaining the system protected from malicious intrusion), but there may also be one-shot properties which are triggered from time to time (e.g. self-reconfiguration after major failures).
- *Time/history dependent versus time/history independent:* behaviour which can be objectively measured at any time is time/history independent. Time/history dependent behaviour needs to be seen in relation to the system's evolution over a certain period (e.g. number of packets delivered per hour).
- *Continuous or smooth evolution:* properties which evolve in a smooth way are rather rare. Most of them jump from one state to another.
- *Adaptation related:* properties which show how well a system adapts to change.
- *Spatial versus non-spatial:* some self-* properties require a spacial structure, while others are not space-related.
- *Resource allocation:* in certain cases the system is required to allocate limited resources to services, or tasks to resources, etc.
- *Group formation:* coalitions or teams may be formed, and also clustering of items or data can be included here.
- *Role-based organisations:* some self-* properties form organisations based on roles and interactions.
- *Self-protection:* some systems need to protect themselves from malicious attacks. This includes defence actions and in certain cases also counter-attack.

4.2 Mechanisms

According to Bar-Yam [52], complex systems should be built with strategies modelled after *biological evolution* or *market economics*. Planning mostly does not work in such systems, and design is often done in parallels *(concurrent engineering)*. Modularity, abstraction, hierarchy and layering are useful methods, but at some degree of interdependence they become ineffective, as discussed in section 1.

Other suitable mechanisms include:

- *Trust:* An efficient method for agents to know with whom to collaborate, and whom to avoid, is managing their levels of trust towards their

peers. Trust can be established through direct interaction as well as through recommendation from peers who know the agent in question.

- *Gossip:* A difficulty of direct communication is that the receiver of the message must be known in advance. Gossip avoids this, and allows messages to randomly spread across a community.
- *Swarm rules:* Different variants of swarm rules (such as seen in flocks of birds or schools of fish) exist, but they mostly consist of three parts: e.g. (1) keep close to your peers, (2) avoid collisions, (3) move forward. Such simple, local rules allow any number of agents to act in a coordinated way without requiring any form of centralised control.
- *Stigmergy:* The deposition of markers in the environment is a way of indirect communication often used by social insects, such as ants deposing pheromones. This leads to *collective intelligence* [94, 95].

Mechanisms generally describe how a process works; patterns (described in 4.2.2) can serve as a more concrete guidance. They define mechanisms in a more systematic way, saying what to do under which conditions.

4.2.1 Friction reduction

Gershenson [17] proposes that *friction* between interacting agents should be reduced. This will result in a higher *satisfaction* of the system, i.e. better performance. To achieve this, *mediators* can arbitrate among the elements of a system. The goal is to minimise conflict, interferences and frictions as well as to maximise cooperation and synergy. See Table 8 for the possible interactions between two agents A and B, where the upper part of the table presents strategies for friction reduction, and the lower part strategies for higher satisfaction.

4.2.2 Patterns

Most mechanisms have been expressed as design patterns, which is a way of referencing mechanisms similar to how it is done in software engineering by Gamma et al. [96].

De Wolf and Holvoet [73] give some guidance for the design of self-* mechanisms under the form of patterns, including a catalogue of coordination mechanisms which allow the emergence of macroscopic properties. Proposed coordination patterns are:

- *Stigmergy:* indirect communication means communication through the environment. Agents depose for example digital pheromones on their current location, and their peers read the information when passing there. In certain cases, indirect communication is more complicated and less specific than the direct exchange of messages. The main advantage is that communication is decoupled. Agents do not need to respond immediately, or wait for a peer to respond.
- *Gradient-field* (also called *computational field*): similar to electric or magnetic fields, computational fields can be sensed by agents who are looking for information or orientation. Notice that gradient-fields can be used to implement other mechanisms, such as stigmergy for task assignment [97] and motion coordination [98].
- *Market-based:* resource allocation is often done by using virtual marketplaces. Agent needing a service make a call for proposals, those offering the service in question answer, and the best offer is selected. This can be done by direct communication, but also works through stigmergy.
- *Tag-based:* tags are observable labels, markings or social cues. They help agents recognise members of a certain group, or agents with a certain characteristic, etc. Tags are especially useful for coordination and group formation.
- *Token-based:* a token is an object which represents the control over a resource or the fulfilling of a role. Token thus exist in limited numbers and are handed from one agent to another when appropriate.

Moreover, Babaoglu et al. [99] recommend the use of basic biological processes as design patterns in distributed computing:

- *Diffusion:* loose entities tend to naturally spread over a free space. They are transported from an area of high concentration to an area of

lower concentration. This mechanism can for instance be exploited to let mobile robots distribute themselves over an area.

- *Replication:* cells, viruses or software programs may create a copy of themselves for various reasons. In computer science, replication refers to the use of redundant resources to improve reliability, fault-tolerance or performance.
- *Chemotaxis:* bacteria and other small living organisms coordinate their movement according to the concentration of chemicals in their environment, i.e. they move towards food sources or away from toxic substances. This concept is related to gradient-fields discussed above.

4.3 Modelling and analysis

The analysis of complex systems is particularly challenging because of the multiple interactions between the components. It is often difficult to detect which components influence each other, and in which ways. There are a few analysis approaches which are specifically made for complex systems, but this does not mean that other approaches may not be suitable as well, if applied with the appropriate care.

Different ways of modelling complex systems take different approaches to solve the problems and have a different focus [50]. The following list is not exhaustive.

- *Hierarchical mappings* refer to the hierarchical decomposition of systems or tasks into simpler sub-units. The focus is on modularisation, which is typically used in the classical engineering approach, and referred to as *divide and conquer*. As an example, hierarchical mappings could be used to design a car, but they are not very well suited for complex adaptive systems.
- The use of *state equations* or *differential equations* is a formal method which considers the states in which a system can be. The focus is on how the system gets from one state to another. This is important for cases where the dynamic systems must be controlled in a stable and optimised way, e.g. motors.
- *Non-linear / discontinuous mechanics* focus on simple behaviours which can have chaotic effects. For instance fluid turbulences can be modelled by non-linear mathematics.
- *Autonomous agents* are naturally suited to model distributed systems where many entities interact in diverse ways. The focus is on the activities of each agent as well as the agents interactions with each other and the environment.
- *Ecosystems* are typical examples of complex systems (see discussion in section 4.1.2). When using ecosystems as a model, engineers often refer to them as being *digital, synthetic* or *virtual*. The processes in ecosystems 'take advantage of emergence and deliberately mimic evolution to accomplish and manage the engineering outcomes desired' [1].
- *Finite element analysis* is a type of numerical analysis, which is typically used to model complicated geometrical structures. Also flows can be modelled with finite elements, e.g. the behaviour of water in a turbine. The focus is on dynamics.
- Schuh et al. [3] suggest that collaborative systems be modelled as networks, and that there is a difference between *guided networks*, which are explicitly managed by a focal entity, and *self-organised emergent networks*, which are implicitly managed by the context.

4.3.1 Requirements

Design structure networks (DSN) [53] are a structured approach to linking requirements with design features. DSN help the designer assess the cost of design changes in complex systems. Woodard furthermore suggests *system design games* and a set of agent-based models (the Palm-Handspring model, the value network model and the platform competition model) to analyse design decisions and their consequences. The method is based on the *theory of design evolution* by Baldwin and Clark [100], which builds on the theory of CAS by Holland [13]. A detailed explanation of Woodard's work would go beyond the scope of this article.

Concept	Explanation
Tolerance	A shares its resources with B
Courtesy	B searches for alternative resources
Compromise	a combination of tolerance and courtesy
Imposition	forced courtesy
Eradication	A eliminates B
Apoptosis	B eliminates itself
Cooperation	A and B work together for the benefit of the whole
Individualism	for the benefit of the whole, A can increase its own benefit
Altruism	A can reduce its benefit to the benefit of the whole
Exploitation	forced altruism

Table 8: Ways to reduce friction / to increase synergy between the elements A and B.

4.3.2 Multi-scale analysis

Multi-scale analysis relates complexity with structure and function. According to Ashby's *law of requisite variety* [43], at every scale, the variety of the system must be larger than the variety necessary for the task to fulfil. In a generalised form it suggests that the effectiveness of a system organisation can be evaluated by its variety at each scale of tasks to be performed [22]. The limits of this method are given by the ability of a single agent (human being) to understand the interdependencies between the components.

4.3.3 Equation-free macro-scale analysis

Equation-free macroscopic analysis [16] serves both analysis and verification. It is mainly usable for swarms and similar collective phenomena which consist of more than one level or scale. While traditional methods focus on the micro-scale only, this method is adapted for macro-scale behaviour.

The equation-free method needs a good microscopic simulation model from which the macroscopic variables can be measured. The strong points of this method is that it is more feasible than formal proofs, founded by dynamical systems theory (which simulations are not), less computationally intensive than a huge number of begin-to-end simulations, and a mixture between individual-based and aggregate-based simulations. It consists of short bursts of microscopic simulations to extract the info which traditional numerical procedures would obtain from direct evaluation of the macroscopic evolution equation, if this equation was available. It requires time-independent converging macroscopic variables (very difficult to find). The method gives statistically relevant info, not about every run of the system.

4.4 Design

The terms *architecture* and *design* are sometimes confused. The architecture (see section 4.5) is the structure according to which a system is built, whereas the design refers to the process of creating a system (including its architecture). Section 4.4.1 explains design strategies, whereas sections 4.4.2 and 4.4.3 report design abstractions.

4.4.1 Design strategies

Marcus [101] suggests the following design strategies:

- Top-down, which is control-based, with predefined coordination and interactions.
- Bottom-up, which is collaboration-based and self-organising; collaboration and coordination emerge from the interactions.
- 'Middle out', which is coordination-based. It combines existing components and collaborations but also drives new requirements, collaborations and components. It is a mediation between a set or requirements and a set of services,

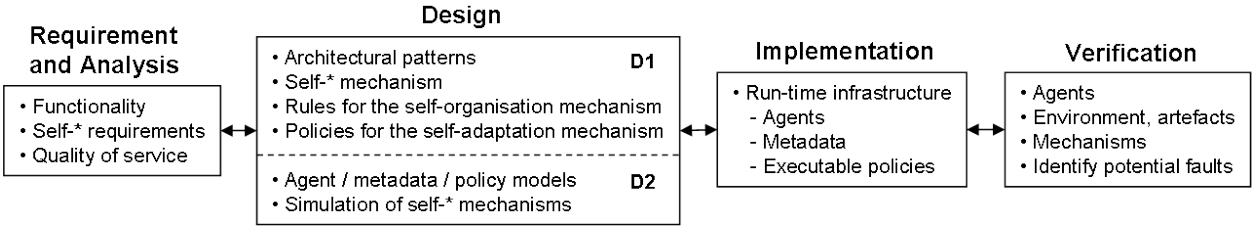

Figure 5: Final state of the MetaSelf development method

new available capabilities and new needed capabilities.

4.4.2 Information flows

De Wolf [16] suggests that *information flows* be established between the various *localities* of the system, which means that the designer focuses on which information needs to be available at which location, at which instant, and where it comes from.

4.4.3 Intelligent networks

Rzevski [48] recommends to create intelligent networks instead of integrated units. This means that intelligence is not inside a single unit but rather emerges from the interactions within the community. The focus should be on adaptability rather than on stability. Three steps in running a system are identified:

1. Sensory perception: detecting and anticipation changes in the environment.
2. Cognition: reasoning about perceived changes and deciding about the best action.
3. Execution: controlling the implementation of cognitive decisions.

4.4.4 BASIC

Schut [95] published a survey on model design for the simulation of collective intelligence. He suggests several levels of model refinement in the design phase, which include problem assessment, modelling (generic, specific and computer model), simulation, verification and validation. The so-called BASIC recipe for modelling consists of determining the following:

1. Action set for all individuals
2. Observation set for all individuals
3. Action → observation methods
4. Costs for individuals for methods from 3
5. Benefits for individuals for methods from 3
6. Observation → action methods for all individuals

The basic recipe can then be augmented with suitable steps for the actual requirements, such as internal states, diversity, non-determinism or adaptivity, as illustrated in [95]. For the specific modelling, diverse models - available in literature - are suggested for typical applications.

4.4.5 Self-made network

Ulieru and Doursat [5] introduce an approach for the bottom-up evolution of architectures which are based on a self-grown network of basic cells, similar to what happens in embryogenesis. This means that the coding of the behaviours are indirect; they guide the behaviour of the components (the cells), and the behaviour of the system as a whole emerges from their interactions.

Concretely, the system consists of self-assembling nodes which have pairs of attachment nodes and pairs of gradient values, which keep track of the node's position in a chain. The ports can be occupied or free, and if free they can be enabled or disabled. Chains are the simplest self-assembled structures, but also considerably more elaborate ones may emerge.

All nodes carry the same programme with three routines for updating the gradient values, port management and link creation. The parameters given to these routines determine then the topology of the self-assembled structures. Depending on the application, the nodes (or agents) may be given additional characteristics, and they may be heterogeneous.

4.4.6 Genetic programming

Genetic programming consists of taking instructions from programs and mixing them based on evolutionary algorithms. A reference model for genetic programming was created by Cramer [102] and formalised by Koza [103]. Fitness functions indirectly represent the global goal of the system; so one might object that the whole process is not emergent in the proper sense. However, it is not given in the fitness function HOW the task is to be solved [54]. The functions only help to evaluate the adequateness of the solution. The agents finally equipped with the result of the evolutionary algorithm do not have any info about the objective functions neither about the fitness of their current actions. It remains open how to solve the mentioned co-evolution of different agent types, or how to deal with heterogeneous agents.

Zapf and Weise propose a solution for what they call *offline emergence engineering* [54], based on a combination of strategies from genetic programming and agent software engineering. In offline approaches, once a program is generated, there are no changes any more. Group behaviour emerges before it is put into the real environment: simulation is proposed as a mean to find out if the emerging behaviour is appropriate, and if so, the system is realised. An advantage is that evolution within a simulated environment avoids a potentially long learning phase in the real environment. However, such an approach has obvious weaknesses: no simulation is ever going to be complete, and there are always factors influencing the system in reality which were not completely understood at simulation time, or which simply cannot be represented due to their nature.

In the case of *online emergence engineering* (as opposed to offline emergence engineering), Zapf and Weise suggest that emergence is planned[6] to occur during execution. Nevertheless, through thorough analysis of the components and their multi-lateral interactions, the range of emergent phenomena can certainly be limited, and engineers can design ways for the system to cope with them. For illustration, consider a mobile robot society based on ant-inspired mechanisms: If the rules and mechanisms are evolved beforehand, simulated to be sure that they work, and only implemented afterwards, this is offline engineering. In case the engineer takes basic rules, implements them, and lets them evolve while already running on the real robots, it is online engineering.

4.4.7 Emergence based engineering

Deguet et al. [104] describe concepts to build systems that will produce emergent phenomena. Emergence happens between the design and the observation: so-called *design-to-behaviour emergence*. *Downward causation* applied to code and behaviour means that the code / algorithm is determined by the system's behaviour, not the programmer / designer. In other words, the designer gives the machine a description of the expected behaviour and gets some code in return. The main idea is to implement or generate the systems without knowing 'how it works'. According to Deguet et al. [104], this can be done by three approaches (each of which is an issue itself!): by imitating phenomena, by using an incremental design process, or by creating self-adaptive systems (and understanding how the (meta-) system will be able to modify itself).

4.5 Architectures

The following architectures are particularly suitable for complex systems.

4.5.1 MetaSelf architecture

MetaSelf [105] is a service-oriented architecture for self-organizing and self-adaptive systems, where the services are provided by components or agents. This

[6] This might be considered as a contradiction in itself: emergence can hardly be planned.

architecture exploits metadata to support decision-making and adaptation, based on the dynamic enforcement of explicitly expressed policies. Metadata and policies are themselves managed by appropriate services. The components, the metadata and the policies are all decoupled from each other and can be dynamically updated or changed.

MetaSelf applications have been made in the area of *dependability explicit computing* [106] and *evolvable assembly systems* [107].

4.5.2 The autonomic manager

As software systems become increasingly complex and difficult to manage, *autonomic computing* [108] was proposed as a way of handling this. Software should actively manage itself instead of passively being managed by a human administrator. Most self-* properties can be achieved under the responsibility of a single autonomous entity (a manager) which controls a hierarchy of other autonomous entities. The *autonomic manager* consists of a central loop which handles all upcoming events within the system. The autonomic manager follows the *MAPE loop* [109], which stands for monitoring, analysis, planning and execution, supported by a knowledge base.

An alternative to this centralised approach is decentralised autonomic computing (see section 4.1.3), where interacting and fairly autonomous individuals replace the manager.

4.5.3 The three layer architecture

Kramer and Magee propose a three layer architecture to realise self-adaptive and self-managing computing systems [110], where the components configure their interactions themselves. The lowest layer is the *component control*, which includes sensors, actuators and control loops. The middle layer takes care of *change management*. It is a sequencing layer, to which the lower layer reports state changes. New control behaviours are planned here, and parameters for existing control behaviours are adapted. Finally, the highest layer implements the *goal management*. Time consuming planning is executed at this level, according to the change requests coming from the middle layer and the high level goals specified by the user.

4.5.4 Controller /observer architecture

Organic computing [111] is a project [7] which combines software engineering with neuroscience and molecular biology. Within this framework, Schoeler et al. [112] developed a controller/observer architecture to 'keep emergent behaviour within predefined limits'. It allows the system to make free decisions within so-called *adaptive islands*, limited by pre-set objectives and constraints.

The basic structure consists of an execution unit which receives an input and generates an output. Above the execution unit, there is an observer/controller unit. The observer receives input from the environment as well as from the execution unit. The controller compares the situation reported by the observer to the goals set by the user and reacts by reconfiguring the execution unit.

4.5.5 Task-based adaptation

Task-based adaptation [113] is performed by self-adapting computing infrastructures which automate their configuration and reconfiguration. Dynamic task selection can be based on an evolving threshold mechanism and agent stimuli [114]. External stimuli come from the environment (as it is, not modified by other agents), from interactions with other agents, and in the form of stigmergy [94], which is indirect communication, or communication through the environment.

The key ideas in task-based adaptation are:

- Explicit representation of user tasks to determine the required service qualities.
- Decoupling task and preference specification from the low level mechanisms; that is a clean separation between what is needed and how it is carried out.
- Efficient algorithms to calculate in real-time near-optimal resource allocations and reallocations.

[7] http://www.organic-computing.org

In task-aware systems, the users specify their tasks and goals, and it is the job of the system to automatically map them into the capabilities available in the ubiquitous environment. Computing applications can adapt and reconfigure themselves according to the current tasks to be fulfilled [113, 115, 116]. Such systems automate *human multiple objective trade-off*, considering situation-dependent preferences (knowledge-based decisions).

4.6 Development and implementation

This section describes methods for development and implementation of the previously created concepts and architectures.

4.6.1 The customised unified process

De Wolf [16] proposes a design methodology based on the *unified process (UP)* [117], which is an existing industry-ready software engineering process. The UP was customised to explicitly focus on engineering macroscopic behaviour of self-organised emergent multi-agent systems.

During the **Requirement Analysis phase** the problem is structured into functional and non-functional requirements, using techniques such as use cases, feature lists and a domain model that reflects the problem domain. Macroscopic requirements (at the global level) are identified. The **Design phase** is split into *Architectural Design* and *Detailed Design* addressing microscopic issues. *Information Flow* (a design abstraction) traverses the system and forms feedback loops. *Locality* is 'that limited part of the system for which the information located there is directly accessible to the entity' [16]. Activity diagrams are used to determine when a certain behaviour starts and what its inputs are. Information flows are enabled by decentralised coordination mechanisms, defined by provided design patterns. During the **Implementation phase**, the design is realised by using a specific language. When implementing, the programmer focuses on the microscopic level of the system (agent behaviour). In the **Testing and Verification phase**, agent-based simulations are combined with numerical analysis algorithms for dynamical systems verification at macro-level.

The CUP approach has been applied to autonomous guided vehicles and document clustering [16].

4.6.2 Policies and metadata

A way to guide a system in its development without hard-coding its behaviour is the use of policies, as suggested by Kephart [118, 119] in the context of autonomic computing [108]. Policies can express actions, goals and utility functions. Depending on their type, they lead one or several agents to directly execute an action (for instance, if the gripper blocks, try to re-initialise it), to maintain their behaviour as to reach a certain goal (e.g., always keep the speed below 3m/s), or to follow a more complicated guideline and choose appropriate actions (such as: reduce the effort of reconfiguration).

Policies always work in conjunction with corresponding metadata, which is data that is not directly processed in operation. Metadata can describe the performance of an axis, the interfaces of a gripper, the preferential partners of a mobile robot or the current availability of a GPS module.

4.6.3 MetaSelf design method

The MetaSelf development method [120], which consists of four phases, is illustrated in Figure 5.

The **Requirement and Analysis phase** identifies the functionality of the system along with self-* requirements specifying where and when self-organisation or self-management is needed or desired. The required quality of service is determined.

The **Design phase** consists of two sub-phases. In the first part, **D1**, the designer chooses architectural patterns (e.g. autonomic manager or observer/controller architecture) and self-* mechanisms, governing the components' interactions and behaviour (e.g. trust, gossip, or stigmergy, that is indirect coordination through changes in the environment [94]). Rules for self-organisation and policies for self-adaptation are defined. In the second part, **D2**, the individual autonomous components

(services, agents, etc.) are designed. The necessary metadata and policies are selected and described. The self-* mechanisms are simulated and possibly adapted / improved.

The **implementation phase** produces the runtime infrastructure including agents or services, metadata and executable policies.

In the **verification phase**, the designer makes sure that agents, the environment, artefacts and mechanisms work as desired. Potential faults and their consequences are identified, similar to the way *failure modes and effects analysis (FMEA)* [121] works, and measures to avoid the identified faults are taken accordingly.

4.6.4 Evolutionary engineering

In Bar-Yam's *evolutionary engineering (EE) / enlightened evolutionary engineering* (E^3) [22, 52], the advances a system makes are often unanticipated and not fully understood, but the system does *learning by doing*. Evolutionary processes are based on incremental iterative change and cyclical feedback. EE includes methods which involve rapid parallel exploration and a context designed to promote change through competition between design / implementation groups, with field testing of multiple variants. Examples of evolutionary methods in software engineering are: *spiral development*, *extreme programming* and the *open source* movement. The functioning products which are in use at a certain moment in time are considered as the evolving population which will be replaced by new generations of products. If the function of a system needs to change, the system can adapt because there are many possible variants of subsystems that can be generated. The focus of E^3 is on creating environment and process rather than a product, and it continually builds on what already exists. Operational systems include multiple versions of functional components, and E^3 uses multiple parallel development processes. More effective components are gradually introduced.

Bar-Yam proposes the following methods:

1. Analysing the environment and temporarily modifying it to influence the complex system's self-directed development. (Complex systems cannot be completely isolated from their environments.)
2. Tailoring developmental methods to specific scales and regimes (i.e. phases in the life-cycle of a complex system, such as development and operation).
3. Identifying or defining a targeted outcome space at multiple scales and in multiple regimes. (Outcome spaces are close to specifying 'requirements' or 'desired capabilities' for complex systems.)
4. Establishing rewards and penalties, including the explicit formulation of satisfying criteria. (Not to confound with direction and guidance, which directly concern agent behaviour; rewards and penalties refer to agent generated outcomes.)
5. Judging actual results and allocate prices. This is associated with the criteria of rewards but also involves the explicit consideration of other outcomes.
6. Formulating and applying developmental stimulants.
7. Characterising continuously, i.e. capturing and publishing information about the way things are at every moment in a complex system. Among others, this helps agents take decisions and allows tracking the evolution of the system.
8. Formulating and enforcing safety regulations (policing).

Related to evolutionary engineering, and maybe better-known, is *evolutionary computation*[122]. It belongs to the field of *artificial intelligence*; it is mostly concerned with optimisation tasks and uses the mechanisms of evolutionary reproduction and inheritance. Evolutionary computation is not to be confused with *genetic programming* (section 4.4.6).

4.6.5 The AMAS theory and ADELFE

Engineering systems which generate emergent functionalities is the goal of Gleizes et al. [79, 123]. Their AMAS (Adaptive Multi-Agent System) theory claims that for any functionally adequate system, there ex-

ists at least one cooperative internal medium system that fulfils an equivalent function in the same environment. ADELFE [124] is an engineering methodology for adaptive multi-agent systems, based on the AMAS theory. ADELFE is limited to cooperative systems and does not provide support for the achievement of specific goals.

The main ADELFE strategy is to maintain cooperation, or in other words, to avoid so-called *non-cooperative situations (NCS)*. Agents try to anticipate these NCS, and act accordingly. This means that designers have to describe their own specific NCS set and plan the respective actions for each kind of agent. Notice that it is certainly not always possible to preview all the NCS which can occur, and designing corrective actions for them is not easy, neither.

A cooperative agent in the AMAS theory has the following characteristics: it is autonomous; it is unaware of the global function of the system (this emerges from the agent level towards the multi-agent level); it can detect NCSs and acts to return in a cooperative state; it is not altruistic but benevolent (it seeks to achieve its goal while being cooperative).

4.6.6 A general methodology

The general methodology by Gershenson [17] provides guidelines for system development. Particular attention is given to the vocabulary used to describe self-organising systems. It is composed of five iterative steps or phases: representation, modelling, simulation, application and evaluation.

In the **Representation phase**, according to given constraints and requirements, the designer chooses an appropriate vocabulary, the abstractions level, granularity, variables, and interactions that have to be taken into account during system development. Then the system is divided into elements by identifying semi-independent modules, with internal goals and dynamics, and with interactions with the environment. The representation of the system should consider different level of abstractions.

In the **Modeling phase**, a control mechanism is defined, which should be internal and distributed to ensure the proper interaction between the elements of the system, and produce the desired performance. However, the mechanism cannot have strict control over a self-organising system; it can only steer it. To develop such a control mechanism, the designer should find aspects or constraints that will prevent the negative interferences between elements *(reduce friction)* and promote positive interferences *(promote synergy)*. The control mechanism needs to be adaptive, able to cope with changes within and outside the system (i.e. be *robust*) and active in the search of solutions. It will not necessarily maximise the satisfaction of the agents, but rather of the system. It can also act on a system by bounding or promoting randomness, noise, and variability. A mediator should synchronise the agents to minimise waiting times.

In the **Simulation phase**, the developed model(s) are implemented and different scenarios and mediator strategies are tested. Simulation development proceeds in stages: from abstract to particular. The models are progressively simulated, and based on the results, the models are refined and simulated again. The **Application phase** is used to develop and test model(s) in a real system. Finally, in the **Evaluation phase**, the performances of the new system are measured and compared with the performances of previous ones.

This methodology was applied to traffic lights, self-organising bureaucracies and self-organising artefacts [17].

4.6.7 Agents and artefacts meta-model

Gardelli et al. [125] use architectural pattern based on the *agents and artefacts (A&A) metamodel* (remember section **??**) which features agents as proactive goal-driven entities, and artefacts as encapsulated services to be exploited by agents. The environment plays an important role in this approach. It consists of artefacts and environmental agents, which are incorporated self-organisation mechanisms. These environmental agents are responsible for sustaining feedback loops between the agents and the environment.

This approach consists of three iterative design stages: modelling, simulation and tuning. In the modelling phase, the agents' behaviour is designed, and architectural structures are sketched. After-

wards, simulation is used to verify the suitability of the agents and the architecture. In the tuning phase, parameters are adapted in order to optimise the system's performance.

4.7 Validation and Verification

After creating solutions at micro-level, the system verification mainly aims at giving guarantees that the resulting macroscopic behaviour meets the requirements [16]. This is almost never straight-forward. For instance, software code cannot proven to be correct, or to have been exposed to all relevant environmental scenarios. It is thus appropriate to talk about *acceptable behaviour* [54], or to give more detailed indications about the verified scenarios.

Most of the approaches which have been proposed for modelling in section 4.3) can also be used for validation and verification purposes, in particular those in sections 4.3.2 and 4.3.3. Sometimes, macroscopic behaviour can only be verified by begin-to-end simulations; efforts to formalise emergence are typically limited to rather simple application scenarios [16]. But as simulations are always abstractions of reality, they alone are often not enough to prove that a complex engineered system will comply with the requirements. Especially self-organisation and emergence challenge researchers. Different subsystems depend on and interact with each other in many often very complex, dynamic and unpredictable ways.

Not all verification methods are equally useful for any case. Most often a combination of different methods will do best. De Wolf [73] proposes the methods represented in Table 9, together with their typical applications.

4.8 Applied approaches

A view of complex systems engineering from the perspective of **integrated circuit design evolution** was given by Bramlett [126]. It seems that, different from other perspectives, for CPU design, component coupling is important, and the systems are considered as closed and highly optimised. The design process can be seen as a series of phase transitions in convergence towards design requirements, which is an emergent property. Abstractions at different levels and granularities are used to define convergence phases, rates and transitions. Often the design process itself is far more complex than the artifact it produces. The author also states that there is a need for open architectures for cross-disciplinary engineering, taking the human as part of the system.

Rzevski [48] presents complexity engineering at the example of an **intelligent variable geometry compressor** and a family of **space exploration robots**); however, some theoretical background about the used strategies may be missing. Remarkably, Rzevski's strategy for *self-repair* is isolating defective parts and thus making them harmless. Such an approach certainly makes sense in practice, but it does not correspond to the usual interpretation of the term *reparation* as it does not repair the defect neither consider the consequences of an isolation on the rest of the system.

5 Discussion, conclusion and directions

After reviewing numerous existing concepts and methods in complexity engineering, we now analyse the general situation. Section 5.1 discusses various complexity engineering related issues which are important to consider. The role of the observer is addressed in section 5.1.1, and challenges as well as limitations of self* systems in 5.1.2. At the end, we draw conclusions (section 5.2) and indicate directions for further research (section 5.3).

5.1 Discussion

The methods and approaches cited in this article are mostly from the area of computer science. This is due to the nature of complexity engineering: the systems in question usually need some kind of intelligence and a corresponding control system, which leads us typically to computer science. Purely mechanical systems are rarely complex and adaptive or self-organised.

***

Although this article is directed at promoting complexity engineering, the authors are aware of the fact

Method	Application
Unit-based and integration testing	most useful for one-shot microscopic properties
Formal proof	microscopic; not usable for interaction models
Statistical experimental verification	long-term ongoing properties; expensive due to large number of experiments
Equation-based macroscopic verification	adaptation-related; only if the macroscopic property in question can be modelled as a variable in a (partial) differential equation
Equation-free macroscopic verification	long-term ongoing properties with smooth and continuous behaviour, adaptation-related; time-dependent variable reflecting the property in question has to be found (see section 4.3.3)
Time series analysis based on chaos theory	adaptation-related long-term behaviour, measuring complexity for instance

Table 9: Verification methods

that complexity engineering methods are not always the most adequate solution. They should be chosen when classical engineering comes to its limits (compare section 2.2), or when alternative ways of solving a problem are desired.

***

The claim that decentralised control should be avoided has been quite prominently uttered in the last few years. But is it always favourable to build a system with purely decentralised control? Decentralised systems also have weaknesses. They are often not optimal, they take longer to solve problems, and may use more resources to do so. On the positive side, distributed systems are robuster and they can better cope with disturbances.

***

Due to their nature, the validation of complex systems with emergence and self-* properties is difficult. Formal approaches at agent level do not automatically cover global phenomena. Simulations are another way to verify system behaviour, but there are the obvious limitations of time requirements and non-completeness to this approach [123]. Formal modelling techniques can capture important features of the design choices and enable designers to reason about them in a useful way [53]. We may have to accept that we will never be able to completely control or predict the behaviour of a complex system; we should rather cope with this by adapting our actions to the new situations [17]. This indicated that deterministic models or predictions are not necessary; having realistic default expectations with the possibility to correct errors or exceptions after they have occurred, works quite well in practice.

5.1.1 The role of the observer

The role of the observer in determining whether or not a system exhibits emergence was treated in section 3.4. The discussion here is more general, not limited to emergence.

What we perceive as an observer (or as many different observers) is often different from what really exists [17]. The observer mostly has a very limited perspective. Not everything happening in a system is visible; the fact that something cannot be seen does not mean that it does not exist.

From the perspective of the observing designer, there is always a temptation to suppose that the created interactions do indeed take place, even if they are not visible. The designer should therefore try not to jump to conclusions which may not be well-founded. Similarly, an observer who is not the designer is always tempted to make interpretations of the observed and find explanations which may not correspond to reality. Also here, caution is appropriate.

According to discussions at the *4th Technical Fo-*

rum Group on Self-Organisation (TF4), some researchers think that an emergent phenomenon is meaningful to the observer (only?), and only if the observer is also the designer. Only the observer-designer determine if a phenomenon is indeed emergent because this person knows how the system works. This means that somebody who does not understand how a system works cannot correctly judge what is happening, i.e. cannot say if a phenomenon is a self-* property, or if it is under centralised control. In many situations, a system will, however, perform differently when under centralised control than when acting in a distributed-autonomous way. A careful observer may be able to determine the differences and come to the right conclusions.

As a matter of fact, a system which is made to run independently from a human observer (i.e. literally all systems we are considering here), will function while being observed or not. We therefore argue that observation can only help the observer to understand the system, but it does not change anything at the level of the system.

5.1.2 Challenges and limitations with self-* systems

Self-* properties (addressed in sections 3.2 and 3.3) are an important part of complexity engineering. They allow systems to play active and increasingly autonomous roles in accomplishing their tasks, but there are also challenges and limitations to the possibilities of self-* properties:

- *Sensitivity to initial conditions:* Systems may efficiently find a way to accomplish their task under certain initial conditions, but not be able to do so when the conditions are slightly different. *Autonomous guided vehicles (AGVs)* may serve as an example: we suppose that their task is to pick up a variety of finished products from assembly stations and deliver them to boxes according to customer orders. If the AGVs start from distributed locations, they may very quickly settle into an efficient rhythm of picking up and delivering products. But when the AGVs start from a single point, it may take them much longer to coordinate the tasks between them, and thus their performance is affected. Engineers thus have to consider their system's sensitivity to initial conditions, and attempt to find solutions to mitigate the effects.
- *Parameter tuning:* Many applications depend on diverse parameters which have to be tuned in order for the system to run smoothly. Human operators often supervise the tuning, or do it manually by trial-and-error. Suitable strategies need to be developed if the system is to do is autonomously.
- *Latency to find new stable states:* Most self-* systems can eventually find stable states or stable solutions to achieve their tasks, but it takes time. This means that the designer and user of self-* systems must be able to accept delays.
- *No solution found / no convergence:* In certain cases, a self-* system may not be able to solve the task given, or its calculations may never converge. The designer has to preview this and arrange for a way out, such as alerting the user and/or settling for a solution which requires the relaxation of certain constraints.
- *Analysis of self-* properties:* It is inherently difficult to analyse self-* properties. The system may find ways to fulfil tasks which the designer did not plan or preview. The other way round, the designer may intend the system to act in a certain way, and in reality, it is all different. Also the interplay between various self-* properties is difficult to analyse. Further research efforts are certainly necessary.
- *Dependability / resilience:* it must be assured that the system does what it is supposed to do, independent from the actual situation and circumstances, and this is challenging, especially for the type of system considered here. Thanks to their redundancy, these systems are often inherently robust to certain types of failures, and this robustness comes for free. They may, however, be fragile when facing other faults. For further discussion seee [26].

5.2 Conclusions

The application of complexity engineering methods should always be accompanied by a reflection on the reasons why these methods have been chosen. Are they useful for the actual application? Or might other methods be more suitable? The engineering method should always be selected with care.

A fundamental challenge of complexity engineering is that it touches many different domains; it is therefore difficult to decide about generally applicable methods. For instance, network models and statistics may be helpful when creating wireless communication systems but not at all for building manufacturing systems. This article tries to structure the existing methods and thus make it easier for engineers to choose a method which is suitable for their applications.

Furthermore, we have positioned complexity engineering within other engineering domains, such as systems engineering and classical engineering. We reviewed the definitions of important notions such as self-organisation and emergence, and explained the controversies between unpredictability, complexity and other related terms.

This article ends with directions for further research which we consider important for the development of complexity engineering.

5.3 Further research directions

Complexity engineering has still not been established as a proper engineering domain. Research remains scattered and focused on specific examples, which is the reason why most methodologies are not generally applicable. We would like to encourage other researchers to make efforts in complexity engineering, and to coordinate their research with peers. A general framework for complexity engineering should be created, linking existing and new methods with each other, giving receipts for how to approach which type of problem. Complexity engineering requires particular attention concerning the following issues [4]: theory, universal principles, implementation substrates, designing, programming and controlling methodologies as well as collecting and sharing of experience.

Although academia increasingly discovers their interest in complexity engineering, industry is reluctant. It is difficult to persuade industrials to give away total control. Complexity is mostly perceived as disturbing, annoying or overwhelming. Researchers should therefore not only develop methodologies for complexity engineering, but at the same time also try to persuade industry of the benefits which using complexity can offer.

Industry requires dependable methods. Self-organised emergent MAS will only be acceptable in an industrial application if one can give guarantees about the macroscopic behaviour [16]. This can be shown experimentally or proven formally. Both formal prove and experimental evidence have advantages and disadvantages. On one hand, experiments often provide statistical evidence that the desired results will often appear under certain circumstances. But it can also mean that the adverse conditions which lead to failure have not been encountered yet. Formal proof, on the other hand, always uses abstractions, and making the right abstractions is difficult. Formal proofs are useful for understanding certain aspects of a system, but they can never express the complete reality. Additionally, they depend on the language chosen to describe the system. Every language has a certain expressivity. This expressivity may be suitable for certain aspects of a system, but limit the model in capturing others.

Methods to provide sufficient evidence of dependability should be developed especially for complexity engineering methods, given that they are often different from traditional engineering methods due to the use of self-* properties and emergence.

Acknowledgements

This work was started while Regina Frei received a PhD grant from the Portuguese Foundation for Science and Technology; she currently receives a postdoc grant from the Swiss National Science Foundation.

References

[1] D. Norman and M. Kuras, "Engineering complex systems." `http://www.mitre.org`, 2004.

[2] K. Delic and R. Dum, "On the emerging future of complexity sciences," *ACM Ubiquity*, vol. 7, no. 10, p. 1, 2006.

[3] G. Schuh, A. Sauer, and S. Dring, "Modeling collaborations as complex systems," in *4th Int. Industrial Simulation Conf. (ISC)*, (Palermo, Italy), pp. 168–174, 2006.

[4] J. Buchli and C. Santini, "Complexity engineering, harnessing emergent phenomena as opportunities for engineering," tech. rep., Santa Fé Institute Complex Systems Summer School, NM, USA, 2005.

[5] M. Ulieru and R. Doursat, "Emergent engineering: A radical paradigm shift," *J. of Autonomous and Adaptive Communications Systems - to appear*, 2010.

[6] R. Frei and J. Barata, "Distributed systems - from natural to engineered: three phases of inspiration by nature," *Int. J. of Bio-inspired Computation*, vol. 2, no. 3/4, pp. 258–270, 2010.

[7] L. De Castro, *Fundamentals of Natural Computing*. New York, USA: Chapman & Hall/CRC Computer and Information Sciences, 2006.

[8] R. Frei, *Self-organisation in Evolvable Assembly Systems*. PhD thesis, Department of Electrical Engineering, Faculty of Science and Technology, Universidade Nova de Lisboa, Portugal, 2010.

[9] R. Frei and G. Di Marzo Serugendo, "Self-organising assembly systems," *Submitted to IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 2010.

[10] C. Lucas, "The complexity & artificial life research concept for self-organizing systems." `http://www.calresco.org`, 2008.

[11] S. Wolfram, "Approaches to complexity engineering," *Physica*, vol. D, no. 22, pp. 385–399, 1986.

[12] J. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1975.

[13] J. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[14] J. Holland, *Hidden Order: How Adaptation Builds Complexity*. Reading, MA, USA: Addison-Wesley, 1995.

[15] J. Holland, *Emergence - From chaos to order*. Oxford, UK: Oxford University Press, 1998.

[16] T. De Wolf, *Analysing and engineering self-organising emergent applications*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 2007.

[17] C. Gershenson, *Design and control of self-organizing systems*. PhD thesis, Faculty of Science and Center Leo Apostel for Interdisciplinary Studies, Vrije Universiteit, Brussels, Belgium, 2007.

[18] M. Bjelkemyr, D. Semere, and B. Lindberg, "An engineering systems perspective on system of systems methodology," in *IEEE System of Systems Engineering*, (San Antonio, Texas, USA), pp. 1–7, 2007.

[19] S. Bullock and D. Cliff, "Complexity and emergent behaviour in ICT systems," tech. rep., HP-2004-187, Hewlett-Packard Labs, 2004.

[20] P. Kenger, *Module property verification - a method to plan and perform verifications in modular architectures*. PhD thesis, Department of Production Engineering, Royal Institute of Technology (KTH), Stockholm, Sweden, 2006.

[21] S. Auyang, *Foundations of complex-system theories in economics, evolutionary biology, and statistical physics*. Cambridge, UK: Cambridge University Press, 1998.

[22] Y. Bar-Yam, "When systems engineering fails - toward complex systems engineering," in *IEEE Int. Conf. on Systems, Man & Cybernetics (SMC)*, vol. 2, (Washington DC, USA), pp. 2021– 2028, 2003.

[23] M. Gell-Mann, "What is complexity?," in *Complexity*, vol. 1, New York, USA: John Wiley and Sons, Inc., 1995.

[24] T. Choi, K. Dooley, and M. Rungtusanatham, "Supply networks and complex adaptive systems: Control versus emergence," *Operations Management*, vol. 19, pp. 351–366, 2001.

[25] C. Mueller-Schloer and B. Sick, "Controlled emergence and self-organization," in *Organic Computing, Understanding Complex Systems* (R. Wuertz, ed.), pp. 81–104, Springer Berlin Heidelberg, 2008.

[26] G. Di Marzo Serugendo, "Robustness and dependability of self-organising systems - a safety engineering perspective," in *Int. Symp. on Stabilization, Safety, and Security of Distributed Systems(SSS)*, vol. 5873 of *LNCS*, (Lyon, France), pp. 254–268, Springer, Berlin Heidelberg, 2009.

[27] M. Waldrop, *Complexity*. New York, USA: Simon & Schuster Paperbacks, 1992.

[28] C. Johnson, "What are emergent properties and how do they affect the engineering of complex systems?," *Reliability Engineering and System Safety*, vol. 91, no. 12, pp. 1475–1481, 2005.

[29] F. Heylighen, "Complexity and self-organisation," in *Encyclopedia of Library and Information Sciences* (M. J. Bates and M. N. Maack, eds.), Taylor & Francis, 2008.

[30] L. Steels, "Towards a theory of emergent functionality," in *From Animals to Animats: 1st Int. Conf. on Simulation of Adaptive Behaviour* (J.-A. Meyer and S. Wilson, eds.), (Paris, France), pp. 451–461, 1991.

[31] M. Kuras, "Complex-system engineering." `http://cs.calstatela.edu/wiki/images/c/c5/Kuras.pdf`, 2006.

[32] S. Wolfram, *A new kind of science*. Champaign, IL, USA: Media, 2002.

[33] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky, "Metaself - an architecture and development method for dependable self-* systems," in *Symp. on Applied Computing (SAC)*, (Sion, Switzerland), pp. 457–461, 2010.

[34] M. Puviani, G. Di Marzo Serugendo, R. Frei, and G. Cabri, "A method fragments approach to methodologies for engineering self-organising systems," *Submitted to ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2010.

[35] N. Correll and F. Mondada, "Modeling self-organized aggregation in a swarm of miniature robots," in *Int. Conf. on Robotics and Automation (ICRA), Workshop on Collective Behaviors inspired by Biological and Biochemical Systems*, (Rome, Italy), 2007.

[36] J. Halloy, G. Sempo, G. Caprari, C. Rivault, M. Asadpour, F. Tache, I. Said, V. Durier, S. Canonge, J. Ame, C. Detrain, N. Correll, A. Martinoli, F. Mondada, R. Siegwart, and J.-L. Deneubourg, "Social integration of robots into groups of cockroaches to control self-organized choices," *Science*, vol. 318, no. 5853, pp. 1155–1158, 2007.

[37] G. Reeves and S. Fraser, "Biological systems from an engineer's point of view," *PLoS Biology*, vol. 7, no. 1, pp. 32–35, 2009.

[38] G. Nicolis and I. Prigogine, *Self-organization in non-equilibrium systems: From dissipative*

structures to order through fluctuations. New York: J. Wiley & Sons, 1977.

[39] C. G. Langton, "Studying artificial life with cellular automata," in *5th Annual Conf. of the Center for Nonlinear Studies, Los Alamos* (D. Farmer, A. Lapedes, N. Packard, and B. Wendroff, eds.), Evolution, Games and Learning: Models for Adaptation in Machines and Nature, (Amsterdam, The Netherlands), pp. 120–149, 1986.

[40] F. van Eijnatten, "Methodological aspects of chaos and complexity in organisation and management," tech. rep., Eindhoven University of Technology, The Netherlands, 2005.

[41] M. Mitchell, "Complex systems: Network thinking," in *Working papers*, `http://www.santafe.edu/research/publications/workingpapers/06-10-036.pdf`, Sante Fé Institute, NM, USA, 2006.

[42] D. Oliver, T. Kelliher, and J. Keegan, *Engineering complex systems with models and objects*. New York: McGraw-Hill, 1997.

[43] W. Ashby, *An introduction to cybernetics*. London: Chapman & Hall, 1956.

[44] P. Ball, *Critical Mass: how one thing leads to another*. London, UK: Arrow Books, 2004.

[45] M. Gladwell, *The Tipping Point: how little things can make a big difference*. London, UK: Abacus, 2000.

[46] Y. Bar-Yam, *Dynamics of complex systems*. Studies in Nonlinearity, Reading, MA, USA: Addison-Wesley, 1997.

[47] D. Newman, "Emergence and strange attractors," *Philosophy of Science*, vol. 63, no. 2, pp. 245–261, 1996.

[48] G. Rzevski, "Designing complex engineering systems," in *Volga Conf. on Complex Adaptive Systems, Keynote paper*, (Samara, Russia), 2004.

[49] G. Rzevski and P. Skobelev, "Emergent intelligence in multi-agent systems," tech. rep., Magenta Technology, 2007.

[50] W. Rouse, "Engineering complex systms: implications for research in systems engineering," *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, vol. 33, no. 2, pp. 154–156, 2003.

[51] R. Abbott, "Complex systems + systems engineering = complex systems engineering," in *Conf. on Systems Engineering Research, Position paper*, (Los Angeles, CA, USA), 2006.

[52] Y. Bar-Yam, "About engineering complex systems: Multiscale analysis and evolutionary engineering," in *Engineering self-organising systems: Methodologies and applications, ESOA 2004* (S. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, eds.), vol. 3464 of *LNCS*, pp. 16–31, Springer Berlin, 2005.

[53] C. Woodard, *Architectural strategy and design evolution in complex engineered systems*. PhD thesis, Business Studies Department, Harvard Univ., Cambridge, MA, USA, 2006.

[54] M. Zapf and T. Weise, "Offline emergence engineering for agent societies," in *Proc. of the Fifth European Workshop on Multi-Agent Systems EUMAS'07*, (Hammamet, Tunesia), 2007.

[55] S. Kauffmann, *At home in the universe: the search for the laws of self-organization and complexity*. Oxford, UK: Oxford University Press, 1995.

[56] F. Heylighen, "The science of self-organization and adaptivity," in *The Encyclopedia of Life Support Systems* (E. Kiel, ed.), Knowledge Management, Organizational Intelligence and Learning, and Complexity, Oxford, UK: EOLSS Publishers, 2003.

[57] S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-organization in biological systems*. Princeton, NJ, USA: Princeton University Press, 2001.

[58] L. Amaral and J. Ottino, "Complex networks - augmenting the framework for the study of complex systems," *European Physical Journal B*, vol. 38, no. 2, pp. 147–162, 2004.

[59] O. Kuzgunkaya and H. ElMaraghy, "Assessing the structural complexity of manufacturing systems configurations," *Int. J. of Flexible Manufacturing Systems*, vol. 18, no. 2, pp. 145–171(27), 2006.

[60] H. Spilker, "Werden Roboter zur Gefahr für die Menschen? Interview with Prof. Alois Knoll," *Technology Review*, vol. June, p. 106, 2007.

[61] B. Edmonds, "What is complexity? - the philosophy of complexity per se with application to some examples in evolution," in *The Evolution of Complexity* (F. Heylighen and D. Aerts, eds.), Dordrecht: Kluwer, 1999.

[62] S. Maguire and B. McKelvey, "Complexity & management: Moving from fad to firm foundations," *Emergence: A J. of Complexity Issues in Organizations & Management*, vol. 1, no. 2, pp. 19–61, 1999.

[63] T. Philipp, F. Boese, and K. Windt, "Autonomously controlled processes - characterisation of complex production systems," in *3rd Int. CIRP Conf. on Digital Enterprise Technology (DET)*, (Setubal, Portugal), 2006.

[64] J. Jost, "External and internal complexity of complex adaptive systems," *Theory in Biosciences*, vol. 123, no. 1, pp. 69–88, 2004.

[65] S. Grobbelaar and M. Ulieru, "Complex networks as control paradigm for complex systems," in *IEEE Int. Conf. on Systems Man and Cybernetics (SMC)*, (Montreal, Canada), pp. 4069–4074, 2007.

[66] F. Heylighen, "What is complexity?." `http://pespmc1.vub.ac.be/COMPLEXI.html`, 1996.

[67] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi, "Dependable self-organising software architectures - an approach for self-managing systems," tech. rep., BBKCS-05-06, School of Computer Science and Information Systems, Birkbeck College, London, UK, 2006.

[68] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, "Self-organisation and emergence in MAS: An overview," *Informatica*, vol. 30, pp. 45–54, 2006.

[69] L. Correia, "Self-organised systems: fundamental properties," *Revista de Ciencias da Computacao*, vol. 1, no. 1, pp. 1–10, 2006.

[70] T. De Wolf and T. Holvoet, "Emergence versus self-organisation: Different concepts but promising when combined," in *Engineering Self-Organising Systems* (S. A. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, eds.), vol. 3464 of *LNAI*, pp. 1–15, Berlin Heidelberg: Springer, 2005.

[71] G. Di Marzo Serugendo, M. Gleizes, and A. Karageorgos, "Self-organization in multi-agent systems," *Knowledge Engineering Review*, vol. 20, no. 2, pp. 165–189, 2005.

[72] G. Muehl, M. Werner, M. Jaeger, K. Herrmann, and H. Parzyjegla, "On the definitions of self-managing and self-organizing systems," in *KiVS 2007 Workshop: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)* (T. Braun, G. Carle, and B. Stiller, eds.), (Bern, Switzerland), pp. 291–301, 2007.

[73] T. De Wolf and T. Holvoet, "A taxonomy for self-* properties in decentralised autonomic computing," in *Autonomic Computing: Concepts, Infrastructure, and Applications* (M. Parashar and S. Hariri, eds.), pp. 101–120, CRC Press, Taylor and Francis Group, 2007.

[74] S. Brueckner, *Return from the ant - synthetic ecosystems for manufacturing control*. PhD thesis, Institute of Computer Science, Humboldt-University, Berlin, Germany, 2000.

[75] C. Mueller-Schloer, "Organic computing: on the feasibility of controlled emergence," in *2nd IEEE/ACM/IFIP Int. Conf. on Hardware/Software Co-Design and System Synthesis CODES+ISSS*, (New York, USA), pp. 2–5, ACM, 2004.

[76] H. Parunak and R. VanderBok, "Managing emergent behaviour in distributed control systems," in *Instrument Society of America (ISA-Tech)*, (Anaheim, Canada), p. 97, 1997.

[77] C. Castelfranchi, "The theory of social functions: Challenges for computational social science and multi-agent learning," *Cognitive Systems Research*, vol. 2, no. 1, pp. 5–38, 2001.

[78] Y. Louzoun and H. Atlan, "The emergence of goals in a self-organizing network: A non-mentalist model of intentional actions," *20*, vol. 2, pp. 156–171, 2007.

[79] D. Capera, G. Picard, and M.-P. Gleizes, "Applying ADELFE methodology to a mechanism design problem," in *Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 3, (New York, USA), pp. 1508–1509, 2004.

[80] J. Fromm, "Ten questions about emergence." `http://arxiv.org/abs/nlin/0509049`, 2005.

[81] M. Bedau, "Weak emergence," *Philosophical Perspectives: Mind, Causation, and World*, vol. 11, pp. 375–399, 1997.

[82] P. Ranjan, S. Kumara, A. Surana, V. Manikonda, M. Greaves, and W. Peng, "Decision making in logistics: A chaos theory based analysis," *CIRP Annals - Manufacturing Technology*, vol. 52, no. 1, pp. 381–384, 2003.

[83] J. Gleick, *Chaos*. London: Vintage, 1987.

[84] C. Webb and F. Lettice, "Performance measurement, intangibles, and six complexity science principles," in *3rd Int. Conf. on Manufacturing Research (ICMR)*, (Cranfield, UK), 2005.

[85] M. Buchanan, *Ubiquity: the science of history... or why the world is simpler than we think*. London: Phoenix, 2000.

[86] M.-O. Hongler, *Chaotic and stochastic behaviour in automatic production lines*. Springer Berlin Heidelberg, 1994.

[87] K. Ryu, E. Yucësan, and M. Jung, "Dynamic restructuring process for self-reconfiguration in the fractal manufacturing system," *Int. J. of Production Research*, vol. 44, no. 15, pp. 3105–3129, 2006.

[88] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[89] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. November, pp. 1118–1121, 2006.

[90] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi, "A metadata-based architectural model for dynamically resilient systems," in *ACM Symposium on Applied Computing (SAC)*, (Seoul, Korea), pp. 566–573, ACM, 2007.

[91] R. Sole, R. Ferrer-Cancho, J. Montoya, and S. Valverde, "Selection, tinkering and emergence in complex networks," *Complexity*, vol. 8, no. 1, pp. 20–31, 2002.

[92] H. V. D. Parunak, J. Sauter, and S. Clark, "Toward the specification and design of industrial synthetic ecosystems," in *4th Int. Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages (ATAL)*, (London, UK), pp. 45–59, Springer-Verlag, 1998.

[93] C. Wu and E. Chang, "Exploring a digital ecosystem conceptual model and its simulation prototype," in *IEEE Int. Symp. on Industrial Electronics (ISIE)*, (Vigo, Spain), pp. 2933–2938, 2007.

[94] E. Bonabeau, M. Dorigo, and G. Théraulaz, *Swarm intelligence*. New York, USA: Oxford University Press, 1999.

[95] M. Schut, "On model design for simulation of collective intelligence," *Information Sciences*, vol. 180, pp. 132–155, 2010.

[96] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, USA: Addison Wesley professional computing series, 1994.

[97] D. Weyns, N. Boucke, and T. Holvoet, "Gradient field-based task assignment in an agv transportation system," in *Proc. of 5ht Int. Conf. on Autonomous Agents*, (New York, NY, USA), pp. 842–849, ACM, 2006.

[98] M. Mamei, F. Zambonelli, and L. Leonardi, "Co-fields: A physically inspired approach to motion coordination," *IEEE Pervasive Computing*, vol. 3, no. 2 April-June, pp. 52–61, 2004.

[99] O. Babaoglu, G. Canright, A. Deutsch, G. Caro, F. Ducatelle, L. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes, "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, pp. 26–66, 2006.

[100] C. Baldwin and K. Clark, *Design rules*, vol. 1: the power of modularity. Cambridge, MA, USA: MIT Press, 2000.

[101] R. Marcus, "Complex systems engineering for the global information grid." `http://cs.calstatela.edu/wiki/images/a/a4/Marcus.ppt`, 2006.

[102] N. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Int Conf. on Genetic Algorithms and their Applications*, (Mahwah, NJ, USA), pp. 183–187, 1985.

[103] J. Koza, *Genetic programming, on the programming of computers by means of natural selection*. Cambridge, MA, USA: A Bradford Book, The MIT Press, 1992.

[104] J. Deguet, L. Magnin, and Y. Demazeau, "Emergence and software development based on a survey of emergence definitions," *Studies in Computational Intelligence*, vol. 56, pp. 13–21, 2007.

[105] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi, "Metaself - a framework for designing and controlling self-adaptive and self-organising systems," tech. rep., BBKCS-08-08, School of Computer Science and Information Systems, Birkbeck College, London, UK, 2008.

[106] R. Paes, C. Lucena, and G. Carvalho, "Using interaction laws to implement dependability explicit computing in open multi-agent systems," in *Brasilian Symposium on Software Engineering (SBES)*, (Joao Pessoa, Brazil), pp. 59–75, 2007.

[107] R. Frei, G. Di Marzo Serugendo, and J. Barata, "Designing self-organization for evolvable assembly systems," in *IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems (SASO)*, (Venice, Italy), pp. 97–106, 2008.

[108] J. Kephart and D. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[109] IBM, "An architectural blueprint for autonomic computing.," Tech. Rep. June, 2005.

[110] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of software engineering (FOSE)*, (Washington, DC, USA), pp. 259–268, IEEE Computer Society, 2007.

[111] R. Wuertz, ed., *Organic computing*. Understanding Complex Systems, Berlin Heidelberg: Springer, 2008.

[112] T. Schoeler and C. Mueller-Schloer, "An observer/controller architecture for adaptive reconfigurable stacks," in *Int. Conf. on Architecture of Computing Systems (ARCS)*, (Innsbruck, Austria), pp. 139–153, 2005.

[113] J. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw, "Task-based adaptation for ubiquitous computing," *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 36, no. 3, pp. 328–340, 2005.

[114] T. De Wolf and T. Holvoet, "Adaptive behaviour based on evolving thresholds with feedback," in *AISB, 3rd Conf. on Adaptive Agents and Multi-Agent Systems (AAMAS)*, (Melbourne, Australia), pp. 91–96, 2003.

[115] D. Garlan, V. Poladian, B. Schmerl, and J. Sousa, "Task-based self-adaptation," in *Workshop on Self-healing systems, 1st ACM SIGSOFT workshop on Self-managed systems*, (Newport Beach, CA, USA), pp. 54–57, 2004.

[116] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, (Shanghai, China), pp. 2–8, 2006.

[117] I. Jacobson, G. Booch, and J. Rumbaugh, *The unified software development process*. Reading, MA, USA: Addison Wesley, 1999.

[118] J. Kephart and W. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proc. 5th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY)*, (New York, USA), pp. 3– 12, 2004.

[119] J. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Computing*, vol. 11, no. 1, pp. 40–48, 2007.

[120] G. Di Marzo Serugendo and R. Frei, "Experience report in developing and applying a method for self-organisation to agile manufacturing," tech. rep., BBKCS-09-06, School of Computer Science and Information Systems, Birbeck College, London, UK, 2009.

[121] R. McDermott, R. Mikulak, and M. Beauregard, *The basics of FMEA*. New York, USA: CRC Press, Taylor & Francis Group, 2008.

[122] K. De Jong, *Evolutionary computation: a unified approach*. Cambridge, MA, USA: MIT Press, 2006.

[123] M.-P. Gleizes, V. Camps, J.-P. George, and D. Capera, "Engineering systems which generate emergent functionalities," in *Engineering Environment-Mediated Multiagent Systems - Satellite Conf. held at The European Conf. on Complex Systems (EEMMAS 2007)*, (Dresden, Germany), 2007.

[124] C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard, "Engineering adaptive multi-agent systems: The adelfe methodology," in *Agent-oriented Methodologies,* (B. Henderson-Sellers and P. Giorgini, eds.), pp. 172–202, Hershey, PA, USA: Idea Group Pub., 2005.

[125] L. Gardelli, M. Viroli, M. Casadei, and A. Omicini, "Designing self-organising environments with agents and artifacts: A simulation-driven approach," *Int. Journal of Agent-Oriented Software Engineering*, vol. 2, no. 2, pp. 171–195, 2008.

[126] B. Bramlett, "Engineering emergence," tech. rep., MIT Media Lab, Cambridge, MA, USA, 2002.