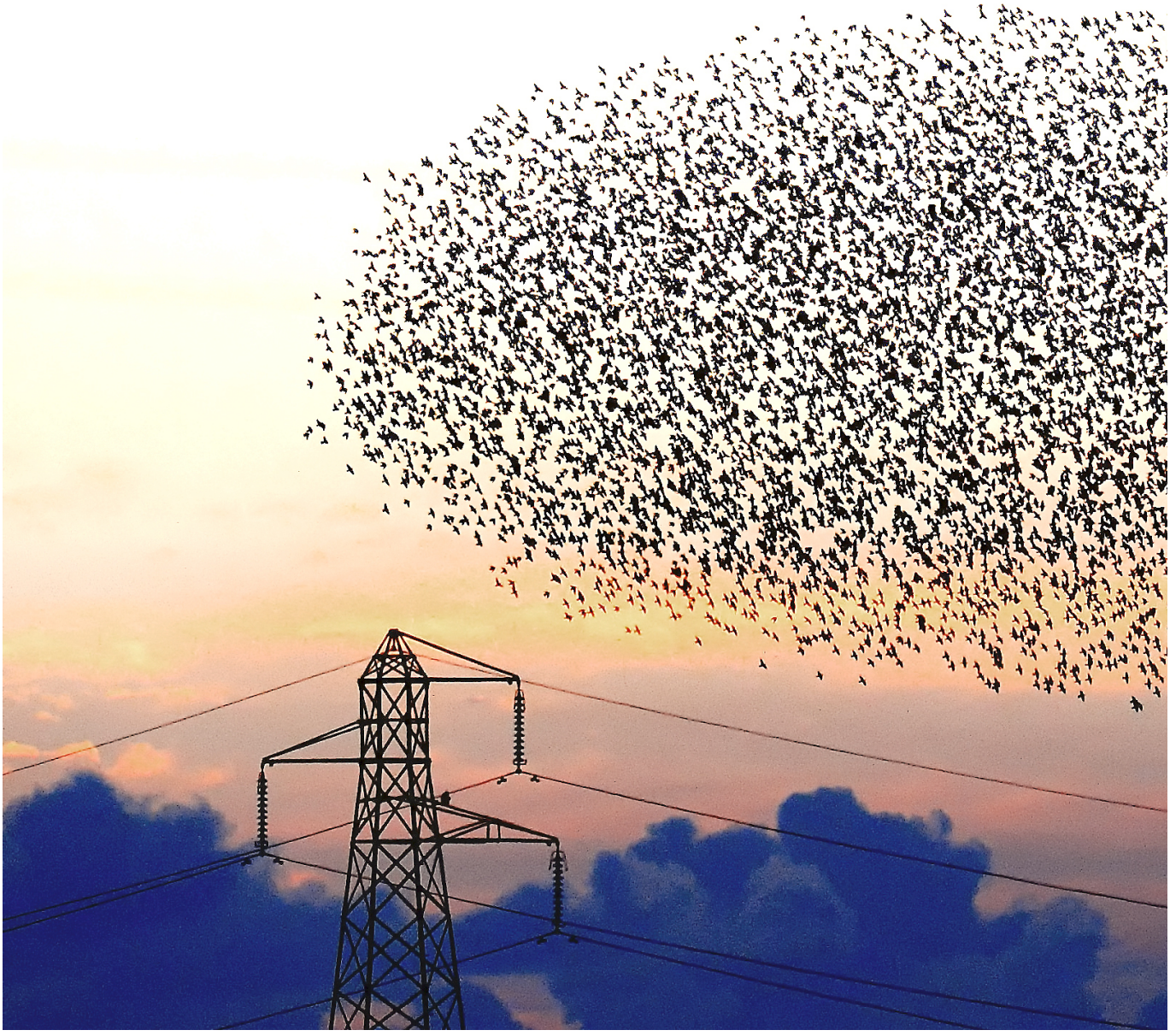
FROM MODULARITY TO EMERGENCE

A PRIMER ON THE DESIGN AND SCIENCE OF COMPLEX SYSTEMS

Chih-Chun Chen and **Nathan Crilly**



CUED/C-EDC/TR.166
Engineering Design Centre
Department of Engineering

UNIVERSITY OF CAMBRIDGE

OVERVIEW

Electrical networks, flocking birds, transportation hubs, weather patterns, commercial organisations, swarming robots... Increasingly, many of the systems that we want to engineer or understand are said to be 'complex'. These systems are often considered to be intractable because of their unpredictability, non-linearity, interconnectivity, heterarchy and 'emergence'. Such attributes are often framed as a problem, but can also be exploited to encourage systems to efficiently exhibit intelligent, robust, self-organising behaviours. But what does it mean to describe systems as complex? How do these complex systems differ from the more easily understood 'modular' systems that we are familiar with? What are the underlying similarities between different systems, whether modular or complex? Answering these questions is a first step in approaching the design and science of complexity. However, to do so, it is necessary to look beyond the specifics of any particular system or field of study. We need to consider the fundamental nature of systems, looking for a common way to view ostensibly different phenomena.

This primer introduces a domain-neutral framework and diagrammatic scheme for characterising the ways in which systems are modular or complex. Rather than seeing modularity and complexity as inherent attributes of systems, we instead see them as ways in which those systems are characterised by those who are interested in them. The framework is not tied to any established mode of representation (e.g. networks, equations, formal modelling languages) nor to any domain-specific terminology (e.g. 'vertex', 'eigenvector', 'entropy'). Instead, it consists of basic system constructs and three fundamental attributes of modular system architecture, namely structural encapsulation, function-structure mapping and interfacing. These constructs and attributes encourage more precise descriptions of different aspects of complexity (e.g. emergence, self-organisation, heterarchy). This allows researchers and practitioners from different disciplines to share methods, theories and findings related to the design and study of different systems, even when those systems appear superficially dissimilar.

Publication details
Chen, C.-C. and Crilly, N. (2016) 'From modularity to emergence: a primer on the design and science of complex systems'. *Technical Report CUED/C-EDC/TR.166*. University of Cambridge, Department of Engineering. ISSN 0963-5432. http://dx.doi.org/10.17863/CAM.4503

Acknowledgements
This work was funded by the UK's Engineering and Physical Sciences Research Council (EP/K008196/1). Document designed by Design Science / design-science.co.uk.

Cover image
Starlings flock around an electricity pylon in the small village of Rigg in the Scottish Borders. Tens of thousands of these birds collectively exhibit spectacular murmations but their interaction with the power lines also cause local outages. Two types of system are at play here: one biological, the other technical. Either of these systems can be seen to decompose into discrete 'modules' that have clear boundaries (birds, pylons) but they can also be seen to integrate into larger assemblages (flocks, networks). Interactions within and between the assemblages (and across systems) can result in unanticipated 'emergence' and other forms of 'complex' behaviour. (Image credit: Owen Humphreys/PA Images)

HOW TO READ THIS PRIMER

The pages of this primer include four different kinds of content.

Note that when we use the term 'system', what we really mean is a system *characterisation*; we do not make any metaphysical claims about the decomposability of physical entities. In addition to defining subsystems, components and supersystems, with respect to a given system, we define an 'environment' of the system as a set of entities and relationships that are not in the set of entities and relationships constituting the system but that belong to a supersystem of the system. The difference between 'the supersystem of s' and 'the environment of s' is that the supersystem of s includes s, whereas the environment of s does not (see Figure 1).

Entities can also be characterised at different levels of *abstraction*. Two elements can be seen to be different to each other at one level but the same as each other at another, more abstract level, where they belong to the same class or 'type'. Classificatory relationships between characterisations determine which characterisations can be treated as equivalent (see Figure 2).

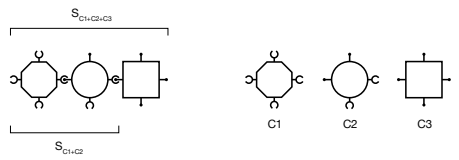
We define a 'type' as a taxonomic group or 'class' associated with a set of subtypes and instances. With respect to a given system type, S,

- A **subtype** of S is a taxonomic group containing a subset of the entity types, entity instances and characterisations contained in the set defined by S.
- A **supertype** of S is a taxonomic group containing a superset of the entity types, entity instances and characterisations contained in the set defined by S.
- An **instance** of S is a concrete realisation of S (an entity in the world) which belongs to the set of entities defined by S.

2.1.2. Hierarchies and heterarchies

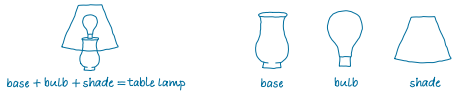
The terms 'level' and 'hierarchy' are frequently found in systems discourse. The part-whole (composition) and subtype-supertype (classification) relationships defined above give us a means of more precisely understanding these terms.

Figure 1: In this diagrammatic scheme, there are different types of entity (represented by different shapes and interfaces). Here, C1, C2 and C3 represent component types and can be combined to make a system type $S_{C1+C2+C3}$. System type S_{C1+C2} is a subsystem of $S_{C1+C2+C3}$. Entity C3 is a component of $S_{C1+C2+C3}$ but is the environment of system S_{C1+C2} (assuming no other entities exist, otherwise it is just part of the environment). These basic aspects of composition apply both to types and instances of entities.



Architectural characterisation. An architectural characterisation of the lamp system might refer to the way the lamp base, bulb and lampshade fit together. An architectural characterisation of the rainforest ecosystem might refer to the way the producers, consumers and abiotic elements interact with each other.

In a table lamp ($S_{C1+C2+C3}$), C1 might refer to the base, C2 to the bulb, and C3 to the shade. Similarly, in a rainforest ecosystem ($S_{C1+C2+C3}$), C1 might refer to the producers, C2 to the consumers, and C3 to the abiotic elements. The subsystem S_{C1+C2} might refer to the abiotic elements.



1. In the main text, we describe systems (and the ways in which they can be understood) in abstract terms. The language is domain-neutral, allowing the text to relate to many different systems and the different disciplines that study them.

2. In the right-hand margins there are concrete examples relating to the abstract descriptions in the main text. Two systems are used throughout: a table lamp, which is generally considered to be 'non-complex' and a rainforest ecosystem, which is generally considered to be 'complex'. These examples are introduced on page 7.

3. The key concepts and constructs discussed in the main text are represented with abstract diagrams that are domain-neutral.

4. Some of the abstract diagrams are redrawn underneath, translating them into illustrations of the table lamp example. This is useful in the earlier parts of the primer, so that the 'visual language' used in the abstract diagrams can be understood. In the later parts of the primer, the concepts are better represented by the more abstract diagrams but their relation to the examples is still discussed in the margin.

1. INTRODUCTION

In both Engineering and Science, the term 'complex system' is used to characterise an entity that is either being designed or observed. This often means that the system has an analytically challenging number of interacting elements, which are described at different levels and which need to be understood from different perspectives. When the relationships between these different levels and perspectives are not well-defined (or are subject to change), the system can be seen as exhibiting unexpected behaviours, sometimes referred to as 'emergence'. Such emergent behaviours might correspond to unanticipated failures or to robust 'self-organising' patterns that can be exploited. It might be tempting to see emergence, self-organisation and other aspects of complexity as inherent to some systems. However, this primer makes no such assumption. Instead, our starting premise is that any system can be described in a multitude of ways. What distinguishes a complex system from a non-complex system is that we do not understand that system well enough to realise our objectives. In other words, 'complexity' is subjective; it describes the stance that is being taken towards a system[1]. That complexity can itself be characterised in many different ways (e.g. emergence) depending on the different ways in which this shortfall in understanding is manifest (e.g. unpredictability).

While 'complexity' in the design context has traditionally been cast in a rather negative light due to the unpredictability it often implies, attempts have also been made to harness complexity (e.g. as seen in 'complexity engineering' (Ottino, 2004) or 'learning from nature' (Dressler & Akan, 2010)). The goal has been to create more efficient systems with desirable change-related properties, such as adaptability, robustness, resilience and evolvability (discussions of these properties can be found in (Fricke & Schulz, 2005; McManus & Hastings, 2006; Ross et al., 2008; Ryan et al., 2013; Schoettl & Lindemann, 2014). In all these cases, concepts of complexity, self-organisation and emergence become central to design practice. Furthermore, a complex systems perspective is becoming increasingly common when tackling design and engineering problems which cut across traditional domain boundaries and involve both designed and non-designed entities (Chen & Crilly, 2016). There are many examples of this:

- distributed computational systems and the internet are studied as natural ecologies (Gao, 2000; Forrest et al., 2005);
- evolutionary design and evolutionary computing study the way selection and diversification mechanisms operate in different environmental conditions (fitness landscapes) to give different solution spaces (Bentley, 2002; De Jong, 2002);
- complex sociotechnical systems are characterised as partially designed and partially evolving (de Weck et al., 2011);
- bio-engineering seeks to design and manufacture artificial systems from biological substrates (Endy, 2005; Knight, 2005).

Adopting a complex systems perspective, such as in the examples above, often requires that knowledge be translated across traditional disciplinary boundaries. (In this primer we use the terms 'discipline' and 'domain' interchangeably, with them both referring to fields based on subject areas, which have established practices, methods, and bodies of knowledge that members of the field's

community use to further the field, e.g. generating products, obtaining further knowledge.) However, despite the fact that many disciplines have made significant contributions to addressing complexity, they rarely benefit from each others' methods, tools or insights due to domain-specific terminology and a lack of explicitness or precision. Within a given context or domain, a lack of explicit precision in how terms are used often matters less because all those concerned tend to share similar assumptions (e.g. designers belonging to the same organisation designing the same product, scientists in the same team studying the same system). Work on rigorously defining complexity-related constructs (e.g. 'emergence', 'self-organisation') tends to assume a particular representation of the system (e.g. a network) or consensus on the terminology used to describe the system (e.g. what the terms 'element', 'component', 'subsystem' mean). The lack of an idealised, comprehensive and consistent representation that generalises across domains makes it difficult for those working within one domain to have confidence in their interpretation of the solutions proposed within another domain (Goldstone & Sakamoto, 2003). This not only limits the dissemination of useful knowledge, but also increases the likelihood that practitioners from different domains will mis-interpret or mis-apply each other's work.

To make the methods, theories and findings from one domain accessible to other domains, we need to consider different aspects of complexity in domain-neutral terms and how they relate to more general systems characterisations. To provide people working in different disciplines and domains with an accessible means to navigate each other's work, this primer develops a domain-neutral framework and diagrammatic scheme that relates the notion of 'complexity' to more fundamental attributes of system architecture, namely structural encapsulation, function-structure mapping and interfacing. These three architectural attributes also constitute three core aspects of modularity, which is seen by some as the antithesis of complexity (or as a panacea for complexity). For designers, modular architectures permit a system to be divided into more manageable parts that can be developed, produced and modified relatively independently. In other words, modularity is seen as a way of 'managing complexity' by containing it within well-defined boundaries. For scientists studying complex systems, modularity offers a way of more manageably understanding the system by conceptually grouping together system elements, states, or behaviours. Relatively strong interactions or dependencies exist within modules, whilst relatively weak interactions exist across them.

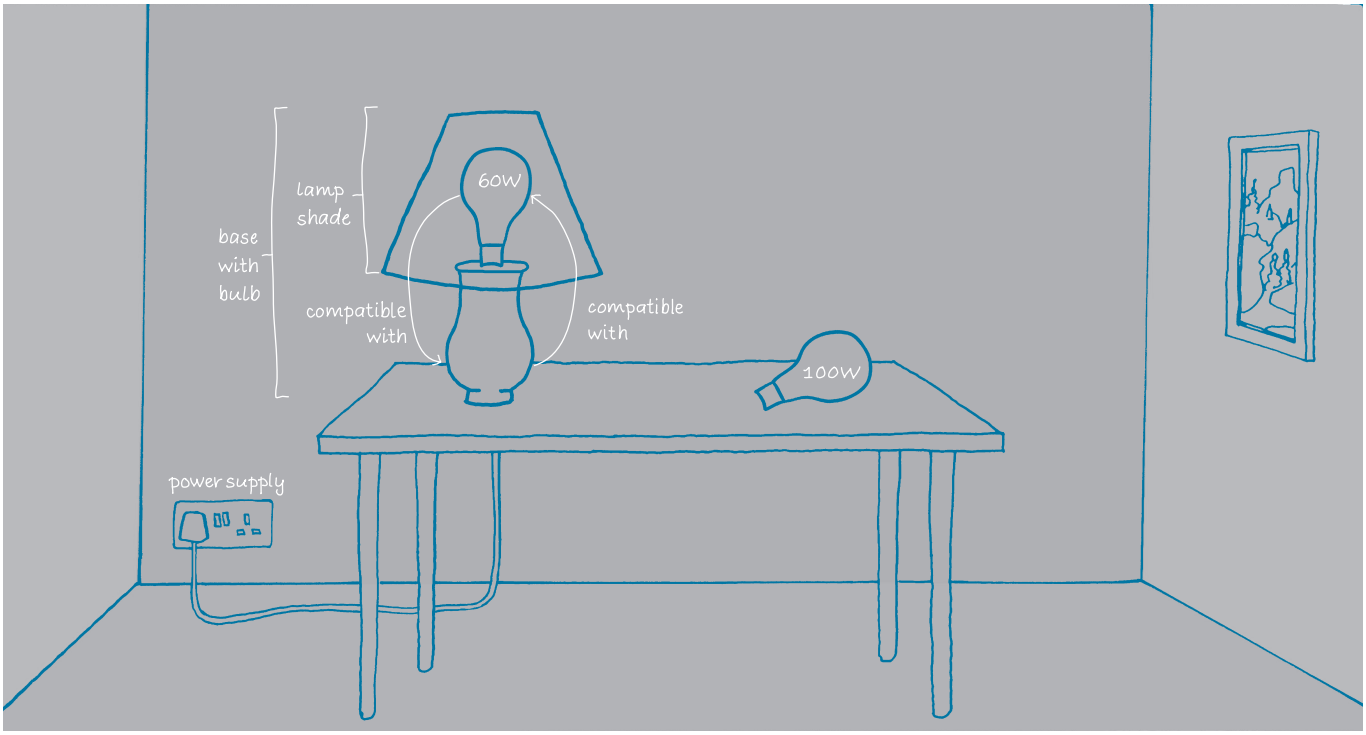
The framework we develop is not tied to any established mode of representation (e.g. networks, equations, formal modelling languages) nor to any domain-specific terminology (e.g. 'vertex', 'eigenvector', 'entropy'). However, it does provide a means of translating between these different formal representations, as well as between formal representations and natural language descriptions. The framework also allows more general systems ontologies (e.g. Bunge, 1977, 1979; Goel & Chandrasekaran, 1989; Gero, 1990; Tomiyama et al., 1993) and systems modelling frameworks (e.g. SysML,[2] CML[3]) to be related to literature on complexity and modularity. Thus, the framework serves as a reference language for the discussion of modularity, complexity and other systems constructs, and the ways in which they are related (as demonstrated in Table 1 and Table 2).

To ensure conceptual explicitness, we include domain-neutral definitions and diagrammatic representations of the key terms introduced. The objective is by no means to comprehensively review the literatures relating to systems, modularity or complexity and therefore we do not endeavour to cite all the 'classic' works from different domains. Instead, we reference other works mainly to illus-

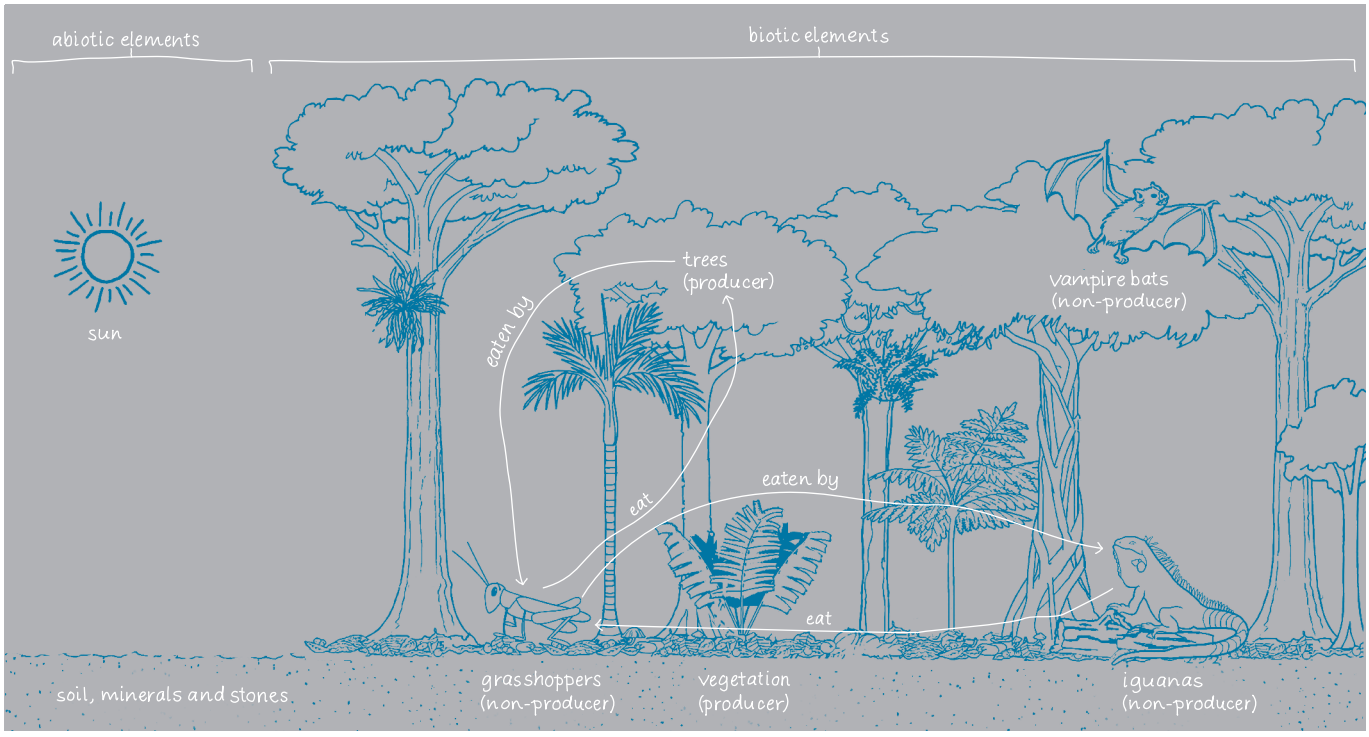
trate terminological discrepancies or to point the reader to further details on the examples given. For domain-specific reviews, the reader is advised to consult introductory texts, on modularity in design (e.g. Ulrich & Eppinger, 1995; Baldwin & Clark, 2000; Gershenson et al., 2003); modularity in science (Newman, 2006); complexity in design (Luzeaux et al, 2013; Sheard et al., 2015); complexity in science (Mitchell, 2009; Ladyman et al., 2013); and system characterisations generally (Meadows & Wright, 2008).

The primer is structured as follows. Section 2 introduces a framework for characterising systems, focusing on characterisations that are particularly pertinent to design domains and scientific domains. The framework also defines composition and classification relationships, which form the basis for levels, hierarchies and heterarchies. Section 3 identifies three core aspects of modularity: structural encapsulation, function-structure mapping and interfacing. Based on these, two abstractions are introduced: function-driven encapsulation and interface compatibility. Section 4 uses the systems characterisation framework (introduced in Section 2) and the aspects and abstractions of modularity (introduced in Section 3) to characterise different aspects of complexity. Section 5 concludes the primer by summarising the relationships between the different aspects of modularity and complexity.

A table lamp is a device found in people's homes that converts electrical energy to light. It is a system made up of a light bulb, lamp base, and lampshade. In order for the table lamp to work, the bulb must be compatible with the lamp base in terms of its attachment mechanism (e.g. bayonet, screw) and wattage (e.g. 60W, 100W). The shade must also be attached for the system to be complete.



A rainforest ecosystem is a self-sustaining organisation found in high rainfall areas. It is a system made up of producers (e.g. trees, fern and moss), non-producers (e.g. grasshoppers, iguanas and vampire bats), and abiotic elements (e.g. water, sun and minerals). The non-producers can be divided into primary consumers (which eat the producers) or secondary consumers (which eat the primary consumers). In order for the rainforest ecosystem to sustain itself, the producers, non-producers and abiotic elements must all interact with each other in specific ways. For example, there must be sufficient numbers of producers to feed the primary consumers, and sufficient numbers of secondary consumers to keep the populations of primary consumers in check so that they do not consume too many of the producers.



2. CHARACTERISING SYSTEMS

To discuss different aspects of complexity and modularity without being tied to the assumptions that particular domains make about systems, we need to have a set of domain-neutral constructs and terms (Chen & Crilly, 2016). We use the term 'characterisation'[4] to refer to any representation, model, specification or description of an entity. Indeed, even calling an entity a 'system' indicates that a certain stance is being taken towards it; the entity is being characterised as a system. By its very nature, a systems characterisation of an entity assumes it can be characterised in multiple ways, each of which emphasise different elements or aspects, reflecting different perspectives and purposes. Within a given context, characterisations are often reified by the community who apply them (Whitehead, 1919) so that a particular characterisation of an entity is treated as the entity itself or as being inherent to the entity.

For the purposes of this primer, we define a 'system' as a set of entities and relationships, where the relationships are connections or interactions between the entities (for a review of systems definitions see Skyttner, 2005: pp. 57–58; Veeke et al., 2008: p. 9). We call the entities in the system the 'elements' of the system, which might be considered 'components' or 'subsystems' with respect to the system, as defined below (of course, these elements might themselves be considered systems in some other characterisation). In order to avoid confusion between cases where we are referring to an entity 'in the world' and cases where we are referring to a *characterisation* of an entity in the world, we use the term 'instance' to refer to the former and 'type' to refer to the latter.[5] (By 'entity in the world' we mean a concrete realisation, but this need not be physical. For example, a process being executed or a procedure being adopted would count as entities in the world within the context of certain system characterisations.) This characterisation might include system architecture, design specifications, functions, behaviour, and so on.

2.1. Composition, classification and levels

In terms of the relationships between entities, we can distinguish between two formal relationships, 'compositional' (part-whole) relationships, and 'classificatory' (subtype-type) relationships. These two relationships provide the basis for defining 'levels' and 'hierarchies' (see Section 2.1.2).

2.1.1. Composition and classification

A composition relationship implies an entity (the 'whole') that can be broken down into a set of further entities (the 'parts'). The term 'element' itself implies a composition relationship between the element and the system. However, different sets of a system's elements can also have part-whole relationships with each other. We use the terms 'subsystem', 'component' and 'supersystem' to characterise such relationships. These are *relational* terms that only make sense when defined with respect to each other and with respect to a given characterisation (see Figure 1). With respect to a given system, s,

- a *subsystem* of s is a subset of the entities and relationships in s;
- a *component* of s is an entity in s that cannot be further decomposed;
- a *supersystem* of s is a superset of the entities and relationships in s.

Note that when we use the term 'system', what we really mean is a system *characterisation*; we do not make any metaphysical claims about the decomposability of physical entities. In addition to defining subsystems, components and supersystems, with respect to a given system, we define an 'environment' of the system as a set of entities and relationships that are not in the set of entities and relationships constituting the system but that belong to a supersystem of the system. The difference between 'the supersystem of s' and 'the environment of s' is that the supersystem of s includes s, whereas the environment of s does not.

Entities can also be characterised at different levels of *abstraction*. Two elements can be seen to be different to each other at one level but the same as each other at another, more abstract level, where they belong to the same class or 'type'. Classificatory relationships between characterisations determine which characterisations can be treated as equivalent (see Figure 2).

We define a 'type' as a taxonomic group or 'class' associated with a set of subtypes and instances. With respect to a given system type, S,

- a *subtype* of S is a taxonomic group containing a subset of the entity types, entity instances and characterisations contained in the set defined by S;
- a *supertype* of S is a taxonomic group containing a superset of the entity types, entity instances and characterisations contained in the set defined by S;
- an *instance* of S is a concrete realisation of S (an entity in the world) which belongs to the set of entities defined by S.

2.1.2. Hierarchies and heterarchies

The terms 'level' and 'hierarchy' are frequently found in systems discourse. The part-whole (composition) and subtype-supertype (classification) relationships defined above give us a means of more precisely understanding these terms. Implicit in the *classification* relationship is the 'resolution' of the characterisation

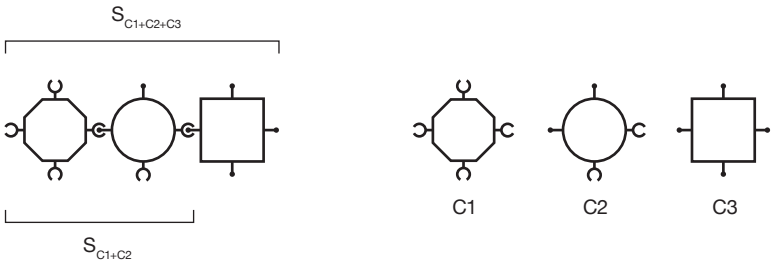


Figure 1. In this diagrammatic scheme, there are different types of entity (represented by different shapes and interfaces). Here, C1, C2 and C3 represent component types and can be combined to make a system type $S_{C1+C2+C3}$. System type S_{C1+C2} is a subsystem of $S_{C1+C2+C3}$. Entity C3 is a component of $S_{C1+C2+C3}$ but is the environment of system S_{C1+C2} (assuming no other entities exist, otherwise it is just part of the environment). These basic aspects of composition apply both to types and instances of entities.

In a table lamp $S_{C1+C2+C3}$, C1 might refer to the base, C2 to the bulb, and C3 to the shade. Subsystem S_{C1+C2} might refer to the 'bulb with base' part of the table lamp. Similarly, in a rainforest ecosystem $S_{C1+C2+C3}$, C1 might refer to the producers, C2 to the consumers, and C3 to the abiotic elements. The subsystem S_{C1+C2} might refer to the biotic elements.

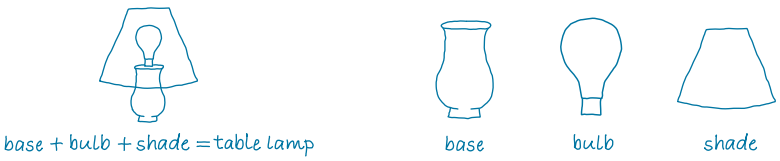
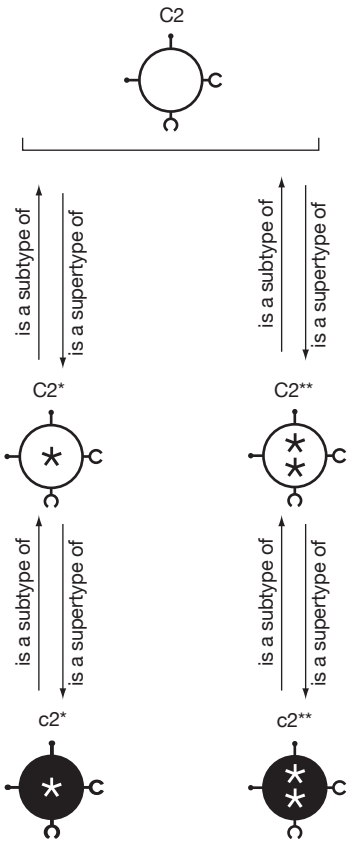
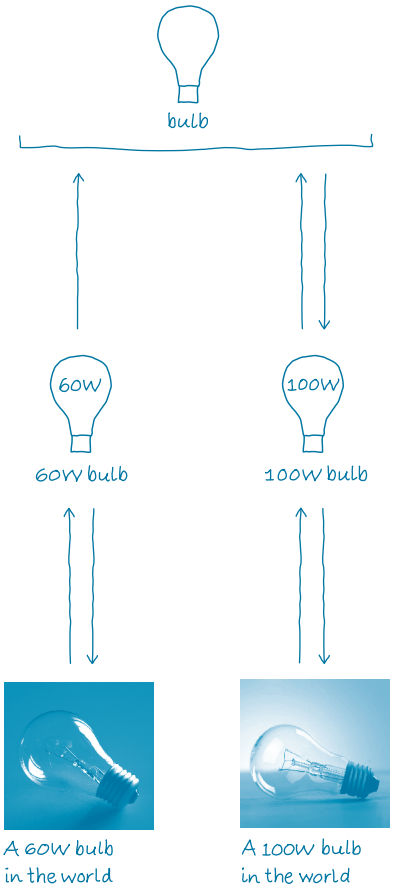


Figure 2. In this diagrammatic scheme, component types (outlined shapes) can be represented at two levels of abstraction: with stars or without stars, where stars represent some feature of the component. These types can also be instantiated (solid shapes). Where components are viewed at a level of abstraction that makes stars visible, there are two options: one star or two stars. Here, two different components are depicted, c2* and c2 (lower-case). Components c2* and c2** are instances of component types C2* and C2** (upper-case), both of which are a subtype of C2. As such, c2* and c2** are also both instances of C2. These basic aspects of classification apply to components, systems, subsystems, supersystems and environments.**



More specific characterisations of bulb (C2) can be given, such as 60W bulb and 100W bulb. Similarly, it is possible to give more specific characterisations of rainforest non-producers (C2). For example, we might use the kind of rainforest to distinguish between tropical rainforest non-producers (C2*) and temperate rainforest non-producers (C2**). These could then be used to refer to a particular set of tropical and temperate rainforest producers respectively, e.g. c2* might refer to the non-producers currently living in a tropical rainforest in Queensland, and c2** could refer to the non-producers currently living in a temperate rainforest in Alaska.



(also known as 'granularity' or 'level of abstraction'), which is the set of distinctions that can be made between the elements. Implicit in the *composition* relationship is what is known as the 'scope' of the characterisation, which is the set of elements involved (see Ryan, 2007 for a more detailed discussion of 'scope' and 'resolution').

We define level as a specification of both the scope and resolution of a characterisation. For example, the level for the system type $S_{C1^*+C2^*+C3^*}$ is defined by the scope of C1*+C2*+C3* and the resolution of $S_{C1^*+C2^*+C3^*}$ as a subtype of $S_{C1+C2+C3}$. Given the definition of 'level', a (clean) hierarchy is defined as a set of related characterisations where the levels do not overlap. A 'classification hierarchy' is a structure in which if one element is the subtype of another element, it cannot also be its supertype. A 'compositional hierarchy' is a structure in which, if one element is the part of another element, that other element cannot also be a part of the first. For example, in $S_{C1^*+C2^*+C3^*}$, the component type C2* is related to the system type $S_{C1^*C2^*C3^*}$ in a compositional hierarchy and to the component type C2 in a classification hierarchy (see Figure 3).

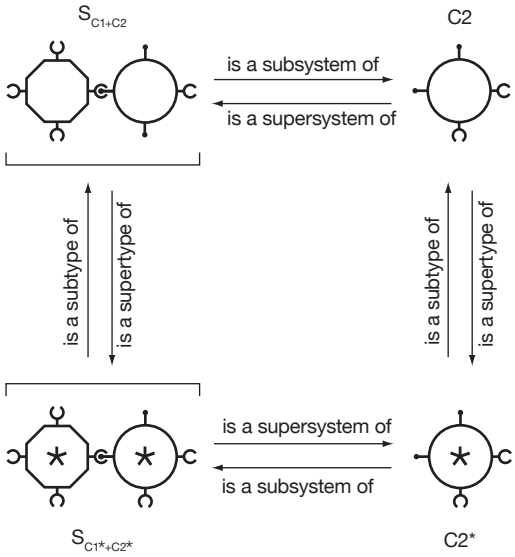
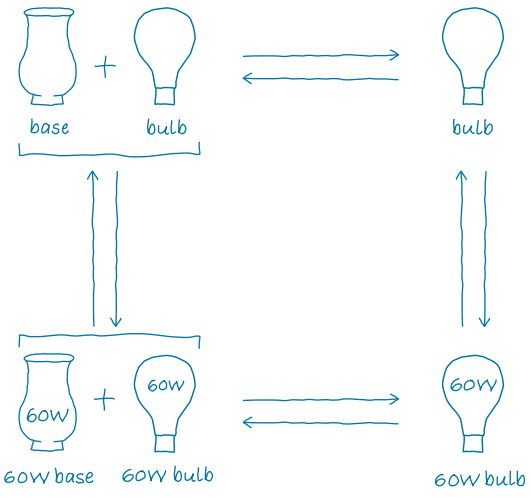


Figure 3. An example of a (clean) hierarchy. C2 is related to S_{C1+C2} in compositional hierarchy, and C2* is related to $S_{C1^*+C2^*}$ in compositional hierarchy. C2* is related to C2 in classificatory hierarchy, and $S_{C1^*+C2^*}$ is related to S_{C1+C2} in classificatory hierarchy.

In this diagram all the compositional and classificatory relationships are straightforward. For example, 'bulb' (C2) has a straightforward subsystem relationship with 'base with bulb' (S_{C1+C2}) and a straightforward supertype relationship with '60W bulb' (C2*). Similarly, 'rainforest non-producers' (C2) has a straightforward subsystem relationship with 'biotic elements of the rainforest' (S_{C1+C2}), and a straightforward supertype relationship with 'tropical rainforest non-producers' (C2*).



In the case of complex systems characterisations, multiple hierarchies overlap in a single characterisation. This is what is referred to as a 'heterarchy' (McCulloch, 1945, Gunji & Kamiura, 2004; Sasai & Gunji, 2008), 'panarchy' (Gunderson & Holling, 2001) or 'entangled hierarchy' (Palla et al., 2005) and can be represented by hypernetworks (Johnson, 2007; Chen et al., 2009). Figure 4 depicts a heterarchy that contrasts with the hierarchy described above. We discuss heterarchy further in Section 4.3.1.

Heterarchy. Not all the relationships between the different characterisations are straightforward classificatory or compositional relationships. For example, the relationship between 'bulb' (C2) and 'base with 60W bulb' system (S_{C1+C2*}), is not a straightforward subsystem-supersystem one. This is because '60W bulb' is specified at a level of detail that exceeds that of the specification of 'base' (which might have variants that are or are not compatible with 60W bulbs). Similarly, the relationship between 'rainforest non-producers' (C2) and 'rainforest producers with tropical rainforest non-producers' (S_{C1+C2*}) is not straightforward (as again, the entities that make up the system are specified at different levels).

2.2. Aspects and mapping relationships

As well as composition and classification relationships between different systems characterisations, there are also mapping relationships. These are used to relate characterisations of different *aspects*[6] of the system, e.g. functions, properties, behaviour, architecture.[7] This section considers three aspects of systems that are important in Design and Science: 'architecture', 'functions' and 'properties'. The pervasiveness of these three concepts is evidenced by the existence of several ontologies relating them, both in design domains (e.g., Goel & Chandrasekaran, 1989; Gero, 1990; Tomiyama et al., 1993) and in scientific domains (e.g., Bunge, 1977; 1979; Wand & Weber, 1990).

2.2.1. Architecture

We define a 'system architecture' as a characterisation of a system in terms of compositional relationships between its elements, where the simplest possible architecture is a single component type. These definitions keep the characterisation of a system's structure distinct from the *mapping* relationships between its structure and function,[8] and is consistent with several definitions and discussions of architecture in the literature (e.g. Simon, 1962; Alexander, 1964; Software Engineering Standards Committee, 2000; Maier & Rechtin, 2009).[9]

Architectural characterisation. An architectural characterisation of the lamp system might refer to the way the lamp base, bulb and lampshade fit together. An architectural characterisation of the rainforest ecosystem might refer to the way the producers, consumers and abiotic elements interact with each other.

Although the terms 'architecture' and 'structure' are typically thought of in terms of spatial relationships between components (e.g. configuration design in the manufacturing literature, Jiao & Tseng, 1999b), we intend 'architecture' to be used in a more general sense here to refer to any relationships (e.g. temporal, logical, social, causal) that might exist between a set of entities. Our definition is therefore general enough to accommodate architectures defined at high levels

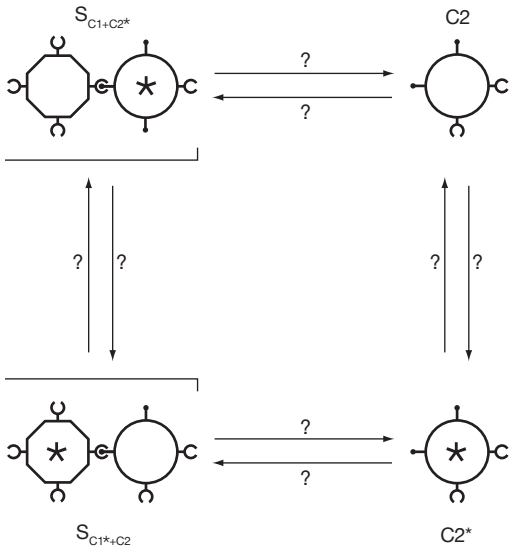


Figure 4. An example of a heterarchy. Although there is a classificatory hierarchical relationship between C2 and C2*, the relationship between $S_{C1*+C2*}$ and S_{C1+C2} cannot be characterised by classificatory hierarchy alone. The relationship between C2 and S_{C1*+C2}, and between C2* and S_{C1*+C2} cannot be characterised by compositional hierarchy alone.

of abstraction with respect to their applications, such as reference architectures (Holtta & Salonen, 2003; Cloutier et al., 2010) and product family architectures (Cloutier et al., 2010). It is also worth noting that while these high level architectures define a set of constructs with which to decompose certain system types, they themselves are system types with a particular architecture (in the same way that grammars are as much linguistic systems as are the languages defined by those grammars).

2.2.2. Functions

The term 'function' is much discussed across various literatures on how systems operate (see reviews in Erden et al., 2008; Crilly, 2010; Houkes & Vermaas, 2010; Preston, 2009; Vermaas & Dorst, 2007), and it is not always easy to see how a single definition can apply across domains (e.g. to both artefacts and organisms). Generally, however, functions describe what a system *should do* in serving some entity, such as satisfying the goals of some agent (e.g. users, designers) or permitting the system to survive and reproduce (e.g. in an ecosystem or market). We leave debate over the nuances of such definitions to other authors and instead focus on clarifying the relationship that functional characterisations have to other kinds of characterisation. Even though the realisation or 'fulfilment' of a function by an entity is dependent on its properties and architecture, the functional characterisation of the entity can be considered independently of these other aspects.

It is also worth emphasising that we do not preclude associations being made between functions and other aspects of systems. For example, a functional requirement of a product might be that it has to adhere to a particular architecture or possesses certain specific properties. Furthermore, functions can themselves be treated as entities in their own right and given compositional characterisations (the 'subfunctions' it is composed of or decomposes into) and classificatory characterisations (the functions it is seen to be a variant of and which variants it itself has).[10]

2.2.3. Properties

We use 'property' as an umbrella term for anything that can be said to be true of an entity (this might even include having a particular architecture or function). When this is expressed statically (or atemporally[11]), we call the property a 'state' (Tomiyama et al., 1993). When it is expressed dynamically (through time), we use the term 'behaviour', or more precisely, 'state transitions' (Gero, 1990; Kam et al., 2001) and 'state transition rules' (see also Section 4.3.3).

Function-based characterisation. A function-based characterisation of the lamp system might be to provide users with diffused light. A function-based characterisation of the rainforest ecosystem might be to sustain populations of particular species.

States. The states of the lamp system might include being "on" or "off" while the states of a rainforest ecosystem might include being in "wet season" or in "dry season".

State transitions. The state transitions for the lamp system would be "on → off", "off → on", with the state transition rules being "if circuit broken, on → off", "if circuit formed, off → on". The state transitions for the ecosystem would be "wet season → dry season" and "dry season → wet season", with the state transition rules "if rainfall exceeds x, dry season → wet season", "if rainfall drops below x, wet season → dry season".

Some of these characterisations might seem "unnatural" when applied to the rainforest ecosystem example. For example, it seems strange to see the relationships between biotic and abiotic rainforest elements as architectural, to assign the rainforest ecosystem the function of sustaining a population of species or to see "dry season" and "wet season" as states. The reason why these characterisations feel a lot more natural in the table lamp example is the human-centric nature of the way the function is defined, and from which the other characterisations (architectural and property-based) are derived.

3. ASPECTS AND ABSTRACTIONS OF MODULARITY

A system characterisation with a straightforward compositional hierarchy describes components and subsystems as interacting (or interfacing) with each other in well-defined, well-understood ways and is said to be 'modular'. Although there exist many different notions of 'modularity', they can be understood and distinguished on the basis of three fundamental attributes of system architecture: structural encapsulation, function-structure mapping, and interfacing (Section 3.1). Table 1 illustrates how these fundamental attributes can be used to consolidate different definitions of modularity found in the literature. From these three fundamental architectural attributes, we can derive two further abstractions, function-driven encapsulation and interface compatibility (Section 3.2).

The table lamp system is usually considered to be more modular than the rainforest ecosystem. This is because the interactions between the bulb, lamp base and lampshade are well-defined with respect to their fulfilment of the lamp's function, while in the case of the rainforest ecosystem, the interactions between the producers, consumers and abiotic elements are less well-defined, and the interactions are often inter-dependent.

3.1. Three core aspects of modularity

The three fundamental attributes of system architecture that we associate with modularity are represented diagrammatically in Figure 5. In this primer, we treat these as the three core aspects of modularity and require that all three of them be satisfied for a set of system elements to be collectively characterised as a 'module':

- structural encapsulation means that the elements can collectively be treated as a single encapsulated component;
- one-to-one function-structure mapping means that the set of elements collectively map to a particular function;
- interfacing means that as a collective, the set of elements has well-defined interactions with other system elements.

3.1.1. Structural encapsulation

We use the term 'structural encapsulation' to refer to the grouping of related *system elements*, i.e. subsystems, into units that can then be treated as component types at some level of abstraction. Structural encapsulation also implies 'interface decoupling' since it allows the relationships between a set of related system elements to be considered independently from its interactions with other system elements.

Structural encapsulation. In the table lamp example, 'bulb' can be treated as a structurally encapsulated unit, and so can 'base with bulb' (which includes 'bulb'). In the rainforest ecosystem example, 'consumers' can be treated as a structurally encapsulated unit, and so can 'biotic elements'. This latter example might feel more forced because we are not used to seeing sets of species as an encapsulated unit, but this is purely due to our habits in characterisation. In a similar way, most of us are not used to seeing a light bulb as a collection of different types of elements or atoms.

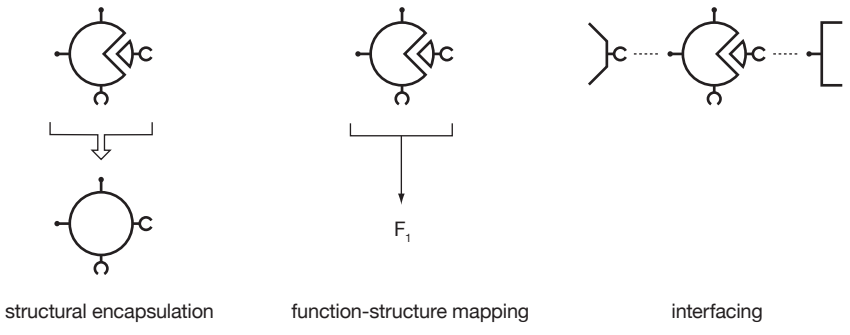


Figure 5. Three aspects of modularity: structural encapsulation (the module is defined by its composition and structure relating its elements to each other, as indicated by the block arrow), function-structure mapping (the module is defined by the collective mapping of a structured set of elements to a function – in this case, F_1, as indicated by the arrow), and interfacing (the module is defined by how a set of elements interacts with other systems, as indicated by the dotted lines).

3.1.2. Function-structure mapping

We use the term 'function-structure mapping' to refer to the mapping between a set of related system elements (i.e. a subsystem) and a function. This structured set of system elements can then be encapsulated into a component type because they are collectively associated with the function. We refer to such encapsulation as 'function-driven' (see Section 3.2.1 below).

Function-structure mapping. In the table lamp example, 'base with bulb' could map to the function of converting electrical energy to light energy, and 'lampshade' could map to the function of diffusing the light emitted. In the rainforest ecosystem example, 'biotic elements' might map to the function of maintaining the food web, and 'abiotic elements' could map to the function of enabling the biotic elements to survive.

3.1.3 Interfacing

We define the term 'interface' as an aspect of the element that allows it to interact with another element or set of other elements in the same system. For those designing physical products it might be most natural to think of interfaces in terms of physical structure or geometric fit. However, interfaces can also be realised in nonphysical ways and the interactions need not be determined by geometry. Examples of non-physical interfaces include standards, protocols, agreements, languages, signals and processes.

Interfacing. In the table lamp example, 'bulb' interacts with 'lamp base' via 'bulb-base interface'. In the rainforest ecosystem example, 'producers' and 'non-producers' interact with each other via 'eaten-eating interface'.

Which aspect(s) of an element is treated as its interface depends on the characterisation adopted, which defines the set of elements with which interaction occurs.[12] This might also mean that multiple interfaces are identified for an element. Indeed, in some cases the interfaces might even be determined by function-structure mapping. For example, what makes the geometry of a given system element its interface might be the requirement of physical fit for the formation of a composite structure to realise a mechanical or chemical function. In such cases, there is an inextricable link between structure and function.

The three aspects of modularity introduced in this primer are useful for structuring our understanding of the modularity literature, allowing it to be more easily understood and compared (Table 1).

Table 1. Different notions of modularity related to the three aspects of modularity introduced in this primer: Structural encapsulation (SE), Function-structure mapping (F-SM) and Interfacing (I).

Aspect of modularity			Examples from literature
SE	*F-SM*	*I*	
×			Abstract characterisation: • A module is a physical or conceptual grouping of components (Jiao & Tseng, 1999b). • Modules contain a high number of components that have minimal dependencies upon and similarities to other components not in the module (Gershenson et al., 1999). Network characterisation: • A subsystem is a module when the number of edges within the subsystem is much higher than the expected number of edges derived from an equivalent random network model with the same number of elements and similar distribution of links between elements with no modular structure (Newman, 2010). In manufacturing and product design: • The most modular architecture is one in which each functional element of the product is implemented by exactly one chunk (subassembly) and in which there are few interactions between chunks. Such a modular architecture allows a design change to be made to one subassembly without affecting the others (Ulrich & Eppinger, 1995). In software design: • There should be no access to, informational flow to, or inter-activity between modules (George & Leathrum, 1985). Modular programming has developed coding techniques which "(1) allow one module to be written with little knowledge of the code in another module, and (2) allow modules to be reassembled and replaced without reassembly of the whole system." (Parnas, 1972: p. 1053).

Table 1 (continued)

Aspect of modularity			Examples from literature
SE	*F-SM*	*I*	
	×		*Abstract characterisation* • The term 'modular' refers to the minimisation of the number of functions per component (Ishii et al., 1995). *In manufacturing and product design* • 'Conceptual' modules perform the same functions even if they have different physical compositions (Otto & Wood, 2001).
×	×		*In manufacturing and product design* • Product modularity is defined in terms of "(1) Similarity between the physical and functional architecture of the design and (2) Minimization of incidental interactions between physical components." (Ulrich & Tung, 1991: p. 73). Therefore a modular product or subassembly has a one-to-one mapping from functional elements in the function structure to the physical components of the product (Ulrich, 1995). • A module is a set of components grouped together in a physical structure and by some characteristic based on the designer's intent (Di Marco et al., 1994; Newcomb et al., 1998). • A module is a component or group of components that can be removed from the product non-destructively as a unit, which provides a unique basic function necessary for the product to operate as desired, and modularity is the degree to which a product's architecture is composed of modules with minimal interactions between modules (Allen & Carlson-Skalak, 1998). • Modularity refers to the "building of complex product or process from smaller subsystems that can be designed independently yet function together as a whole" (Baldwin & Clark, 1997: p. 84). • Modularity requires similarity of functional interactions and suitability of inclusion of components in a module (Huang & Kusiak, 1998). *Abstract characterisation* • A module is a component or subsystem in a larger system that performs specific function(s) and emerges as a tightly coupled cluster of elements sharing dense intra-module interactions and sparse inter-module interactions (Sarkar et al., 2013).
		×	*In manufacturing and product design* • A module is a group of standard and interchangeable components (Galsworth, 1994). • Modular systems are those that are constructed from standardised units of dimensions for flexibility and use (Walz, 1980).
	×	×	*In manufacturing and product design* • A modular product is "a function-oriented design that can be integrated into different systems for the same functional purpose without (or with minor) modifications" (Chang & Ward, 1995 in Gershenson et al., 2003: p. 298). • Modules are groups of components that can easily be re-used or re-manufactured, also considering material compatibility (Sosale et al., 1997). *In software design* • Modularity refers to "tools for the user to build large programs out of pieces" (Chen, 1987, in Gershenson et al., 2003: p. 297).
×		×	*Abstract characterisation* • A module is a structurally independent building block of a larger system with fairly loose connections to the rest of the system. They have well-defined interfaces which allow independent development of the module as long as the interconnections at the interfaces are retained (Holtta & Salonen, 2003). *In manufacturing and product design* • Modularity is design with subsystems "that can be assembled and tested prior to integration… to reduce the time and cost of manufacturing" (Carey, 1997, in Gershenson et al., 2003: p. 298). • Modularity is using sets of units designed to be arranged in different ways (Belle & Kissinger, 1999). • Physical adjacency, energy transfer, information transfer and material exchange can be used to group elements together so they are treated as modules (Pimmler & Eppinger, 1994).
×	×	×	*Abstract characterisation* • Modules are cooperative subsystems which (i) can be combined and configured with similar units to achieve different outcomes; (ii) have one or more well-defined functions that can be tested in isolation from the system and that (iii) have their main functional interactions within rather than between modules (Marshall et al., 1998).

3.2 Two abstractions from modularity

From the three core aspects outlined above, we can derive two further abstractions that also pervade the modularity literature: function-driven encapsulation and interface compatibility.

3.2.1 Function-driven encapsulation

We use the term 'function-driven encapsulation' to describe cases where the criterion for encapsulation is the fulfilment of a function (see Figure 6). What connects elements within a group to each other is that they collectively map to a function, and what makes this set of elements disconnected from other elements is the fact that these other elements do not participate in the fulfilment of that function (being 'connected' or 'disconnected' might also be a matter of degree, and the mapping to a function is specific to a particular level of abstraction and scope). Function-driven encapsulation can be seen as one of a set of many different forms of encapsulation, each of which is distinguished by the kind of criteria that determines encapsulation. For example, we might also have property-driven or behaviour-driven encapsulation where elements are seen as 'connected' when they collectively realise a particular property.[13] However, since our definition of modularity is concerned with the relationship between elements and functions, we require encapsulation to be *function*-driven.[14]

Function-driven encapsulation. In the table lamp example, 'bulb' can be treated as an encapsulated unit by virtue of serving the function of converting electrical energy to light (F1). In the rainforest ecosystem example, 'non-producers' can be treated as an encapsulated unit because they serve the function of controlling the population of 'producers' (F1).

We say that a system architecture is 'completely modular' if every element in the system belongs to a functional group and fulfilment of the system's overall function is completely accounted for by these function-structure mappings (see Figure 7). In the design and management of systems, encapsulation has been said to provide a means of 'managing complexity' by hiding the intricacies of certain regions of the system so that characterisations of them can be separated from the characterisation of the relationships that exist between them and other regions of the system.

Figure 6. An example of function-driven encapsulation: the structural encapsulation of the module is determined by function-structure mapping.

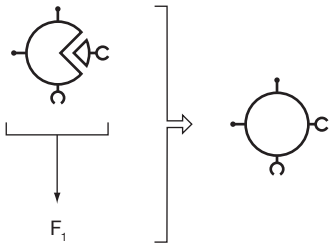
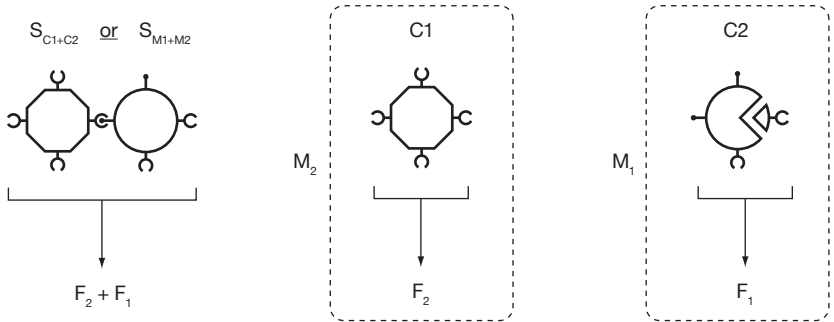


Figure 7. An example of interface compatibility, leading to a modular architecture. The system type S_{C1+C2} has a completely modular architecture since all its elements (both C1 and C2) belong to or constitute modules (M_2 and M_1, respectively). In this case, the modules are defined by function-structure mapping.



3.2.2 Interface compatibility

Interface compatibility refers to the compatibility between different components of the system. This compatibility might be a matter of degree and characterised as the strength of interaction. Interface compatibilities between system components determine how different groups of system elements are able to interact with each other, thus providing a characterisation of the system's architectural constraints. In a completely modular architecture, since all the elements would be modules or would belong to modules and hence be encapsulated in components, interactions between elements in different components would always be via their interfaces. Well-defined interfaces permit components and subsystems with different structures and functions to occupy the same 'position' as each other in the system.

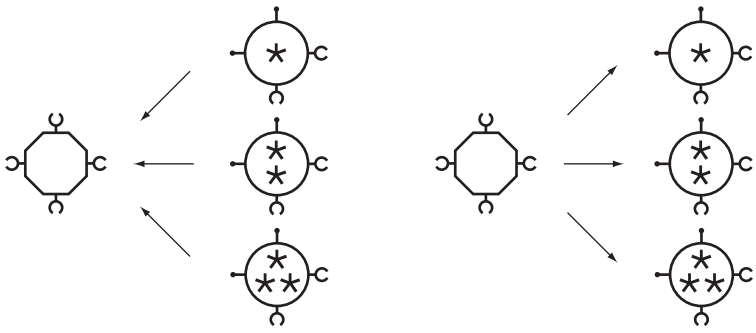
Interface compatibility. In the table lamp example, The bulb is compatible with the base because it has a particular type of attachment mechanism (e.g. screw), which serves as its interface with the base. Similarly, in the rainforest ecosystem example, the non-producers are compatible with the producers because they have particular dietary habits which serve as their interface with the producers.

If all modules (components mapped to functions) in a system had the same mutually compatible interfaces with each other, there would be no architectural constraints at the module level since any module would be able to interact with any other module i.e. architectural degrees of freedom would be maximised, and every component could be 'repositioned'. This is known as 'sectional' modularity (Ulrich & Tung, 1991; Ulrich, 1995), where every component in the system has the same set of interfaces. At the other extreme, where interfaces minimise architectural degrees of freedom and each component has a specific 'position' or 'role' in the system, we have 'slot' modularity (Ulrich & Tung, 1991; Ulrich, 1995). In 'slot' modularity, each component has a unique set of interfaces, which implies that it has a unique set of interactions with other components in the system and hence can only be located in a single specific position with respect to them.

Interface compatibilities can provide a means of controlling which parts of the system can vary. In a given system architecture, different elements of different types (possibly mapping to different functions) can interact with the same set of other elements, so long as they have the same interface compatibilities.[15] In a modular architecture (where the system can be decomposed into components mapped to functions), interface compatibilities determine which components can be swapped or substituted for each other. The terms 'component-sharing', (Ulrich, 1995) 'substitution' (Garud & Kumaraswamy, 1993, Mikkola & Gassmann 2003) and 'standardisation' (Miozzo & Grimshaw, 2005) are used in the literature to refer to cases where, at a particular level of abstraction, different component types have the same interfaces (i.e. they are compatible with the same set of other component types).[16] This 'component-sharing', together with overall architectural similarity between products, can be the basis for establishing product 'families' (Galsworth, 1994; Ulrich, 1995; Jose & Tollenaere, 2005). The term 'component-swapping' (Ulrich, 1995) is used to refer to cases where, at a particular level of abstraction, component types are mapped to different functions but have the same interfaces and therefore can be substituted for each other architecturally (see Figure 8). If these differences in component function have implications for a product's overall function, they provide the basis for the different product variants in product 'families'.[17]

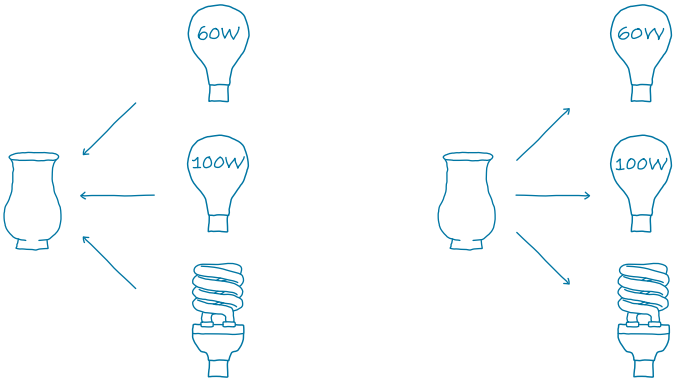
Interface compatibility provides us with a formal means of characterising and analysing architectural variety in terms of elements' compatibilities with each other and the different architectural configurations they permit.

Figure 8. 'Component-swapping' always implies 'component-sharing', and vice versa. When we are taking the perspective of a component (here, the octagon) that can interact with a variety of other entities, the architecture is characterised as 'component-swapping' (different components can be swapped 'in or out' of the octagon). When we are taking the perspective of different entities that can all interact with the same component (the octagon), the architecture is characterised as 'component-sharing' (the octagon is a component that can be shared 'between' different entities).



'component swapping' 'component sharing'

The table lamp example can be seen as permitting 'component-swapping' or 'component-sharing'. When we are taking the perspective of a 'base' that can accommodate a variety of 'bulbs' (e.g. different power ratings, shapes), the architecture is characterised as 'component-swapping'; when we are taking the perspective of different types of 'bulb' that are all compatible with a 'base', the architecture is characterised as 'component-sharing'. Similarly, for the rainforest ecosystem example when we are taking the perspective of 'producers' that satisfy the dietary requirements of different species of 'consumers', the architecture is characterised as 'component-swapping'; when we are taking the perspective of the different species of 'consumer' whose dietary requirements are satisfied by the same 'producers', the architecture is characterised as 'component-sharing'.



base can accommodate a variety of bulbs

different types of bulbs are compatible with base

4. ASPECTS OF COMPLEXITY

The term 'complexity' is used in different ways in the design literature and is often used interchangeably with 'complicated'. We treat these as two distinct concepts (as several authors also do, e.g. Sargut & McGrath, 2011). Characterising a system as 'complicated' is to understand it as having many components, subsystems and interactions; however, as with a simple system, it is theoretically possible to map functions to components and subsystems in a one-to-one fashion, and to describe the interactions between them. By contrast, characterising a system as 'complex' is to understand the system in a way that does not allow this kind of one-to-one mapping or full description of the interactions between components and subsystems. The three aspects and two abstractions of modularity discussed above can be used to distinguish between different aspects of complexity.

4.1 Complexity as non-one-to-one function-structure mappings

Function-driven encapsulation ensures one-to-one mapping between function and architecture. Complexity arises when, at some level of abstraction, the mapping is not one-to-one.

4.1.1 Multi-structural function realisation and architectural robustness

We use the term 'multi-structural function realisation' to describe cases where a function maps to more than one architecture (more than one component type). In Design and Engineering, the possibility of realising a function with different architectures offers the opportunity for robustness and reduction in cost. Robustness comes from the fact that if one architecture mapping to a function is not realised, others may be able to realise it instead. Cost reduction would come from the fact that the number of components required for a given level of robustness might be lower than if this robustness were achieved through duplication of components (see Figure 9).

Redundancy through different architectures. In the table lamp example, redundancy might be achieved through duplicating 'lamp' (duplicated architecture) or through having one 'lamp' and one 'candle'. Similarly, in the rainforest ecosystem example, redundancy might be achieved by having either enough 'iguanas' to consume 'grasshoppers' or by having enough 'iguanas' and 'vampire bats' to do the same.

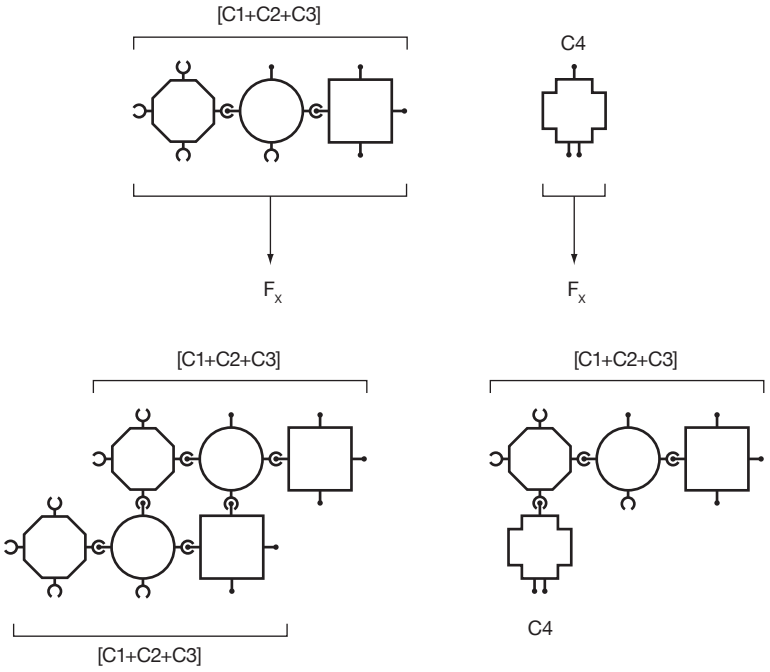


Figure 9. Redundancy through duplicated architectures and distinct architectures. Top row: Both the [C1+C2+C3] architecture and the C4 architecture map to F_x. Bottom left: redundancy in F_x is provided by an architecture with duplication of [C1+C2+C3]. Bottom right: redundancy in F_x is provided by two distinct architectural realisations, [C1+C2+C3] and C4.

In Engineering Design, the term 'principle redundancy' (Pahl & Beitz, 1996) describes cases in which multiple architectures realise the same function. In Biology, the term 'degeneracy' describes cases where, when a particular element is not able to fulfil its function, other means of fulfilling that function are possible (Tononi et al., 1999; Edelman & Gally, 2001; Whitacre, 2010; Chen & Crilly, 2014). For example, a function that was previously associated with a single element might also become distributed among multiple elements.

Compared to duplication, multi-structural function realisation offers a more robust form of redundancy when the different architectures able to realise the function have different points of fragility and strength (see Figure 10). On the other hand, it makes the function-structure mappings harder to analyse, and when there is system failure, it can be difficult to identify the elements involved.

Multi-structural function realisation. If there is a power cut, then having 'lamp' and 'candle' would be more robust than having a pair of 'lamps'. Similarly, if there were a disease affecting only iguanas, it would be more robust to have both iguanas and vampire bats to consume the grasshoppers.

We say that a system is 'architecturally robust' if variety in function is low with respect to architectural variety (the ratio of the number of functions to the number of architectures is low). Architectural robustness is positively associated with evolvability (Whitacre, 2010) since the greater the architectural variation with respect to a function, the larger the set of possibilities to be selected from, and the greater the evolvability. Selection pressures can also be characterised in terms of function realisation. For example, referring back to the architectures in Figure 9, having both [C1+C2+C3] and C4 as possibilities would make the system both *architecturally robust* with respect to F_X (see Figure 10) and *more evolvable* with respect to F_X compared to the case where only one of the architectures could be realised. If the system found itself in an environment requiring F_X to be realised, there would be a selection pressure in favour of the architecture [C1+C2+C3]. We might also say that the *evolvability* of the system with respect to F_X is in virtue of its *adaptability* with respect to F_X. To some extent, this is simply a question of the level at which we are considering the system. For example, a production process might permit a change in parts supplier which then allows a manufacturing firm to resist changes in supplier prices; an organism's ability to change its behaviour in response to different temperature conditions allows it to operate in different environments.[18]

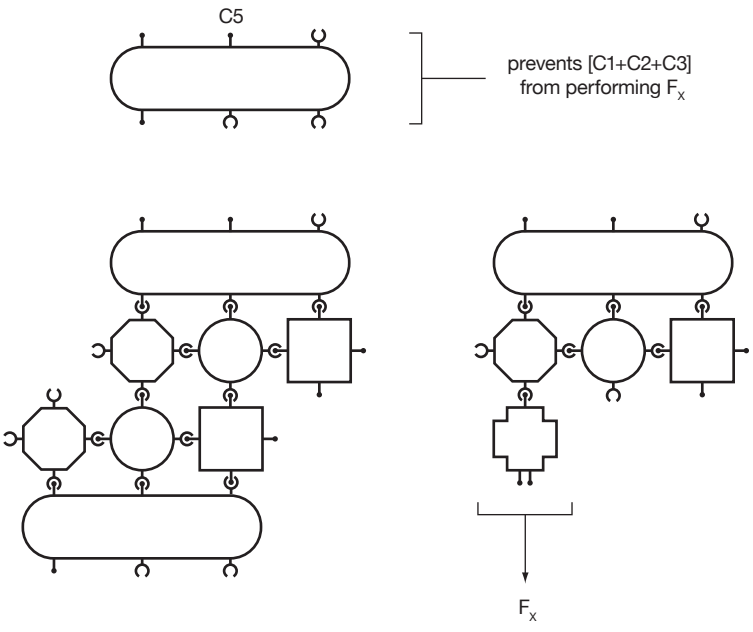


Figure 10. An example of how multi-structural function realisation provides robustness. As in Figure 9, [C1+C2+C3] and C4 are both mapped to F_X. Top row: C5 prevents the architecture [C1+C2+C3] from realising F_X. Bottom left: The presence of C4 prevents [C1+C2+C3]+[C1+C2+C3] from realising F_X. Bottom right: The multi-structural function realisation architecture of [C1+C2+C3]+C4 allows it to be more robust than [C1+C2+C3]+[C1+C2+C3] with respect to realising F_X since it can do so in the presence of C5.

4.1.2 Context-dependent multi-functionality and architectural flexibility

We use the term 'context-dependent multi-functionality' to refer to cases where an architecture maps to different functions based on the wider system architecture it is part of. In systems terms, this means a subsystem realises different functions based on which other systems it is connected to (its environment), i.e. the supersystem it is part of. Figure 11 shows how C2 can be characterised as context-dependently multi-functional. When it is connected to C1 and [C3+C6], it realises F_{Y1}, and when it is connected to C1 and [C3+C7], it realises F_{Y2}.

In design domains, re-purposing of products, product parts and processes are examples of context-dependent multi-functionality. For example, a steel rod realises different functions depending on the wider physical structure it is part of; in software, the same data can have different functions depending on the sections of the program that they flow into; the biochemical function of a protein can depend on the other molecules present; the economic impact of one consumer's purchase depends on the purchasing activities of other consumers.

When the contexts in which different functions are realised are not well-understood, functions may be realised unexpectedly or 'emerge' (sometimes resulting in non-fulfilment of other functions). On the other hand, if the context-dependencies are well-understood, multi-functionality can be exploited to get (desired) functional variety from a given architecture.

We say that a system is 'architecturally flexible'[19] if variety in function is high with respect to architectural variety (in the limit, every architectural variation would be functionally relevant and the ratio of functions to architectures would be unity).[20] This has the potential advantage of allowing a system to realise a greater variety of functions with a relatively small number of elements, but also makes it more difficult to analyse and predict with respect to these functions.

Context-dependent multi-functionality. In the table lamp example, 'bulb' might be seen to map to the function of illuminating a painting in a dark heated room, but might be seen to map to the function of providing heat in a cold room that is already illuminated. In the rainforest ecosystem example, grasshoppers might be seen to map to the function of reducing the population of certain grasses in their natural environment, but might be seen to map to the function of providing food when removed from their natural environment and served as a delicacy.

4.2 Complexity as ill-defined interfaces and shifting system boundaries

A modular system has subsystems (the modules) with well-defined interfaces, resulting in a perfect compositional hierarchy; each module can be treated as a 'closed' system. 'Complexity' arises when interfaces are ill-defined or changing, and the boundaries between the subsystems are constantly changing so that subsystems are 'open' systems. Of course, as with function-structure mappings, this is really a question of characterisation.

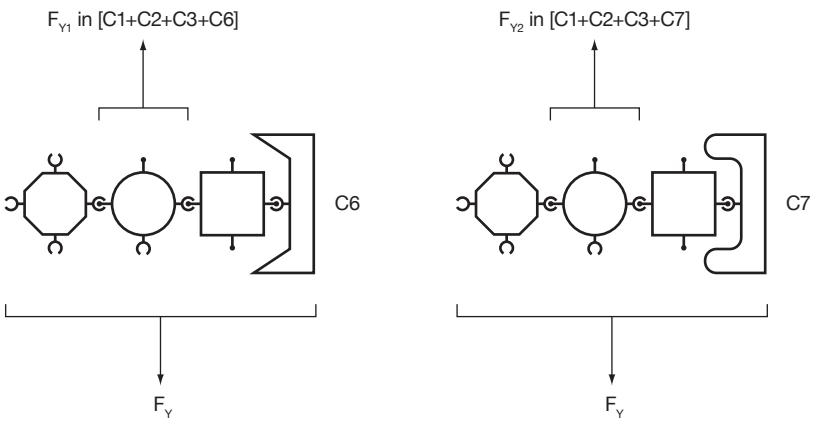


Figure 11. An example of context-dependent multifunctionality. The same component (in this case, the circle, C2) realises different functions by participating in different architectures, even if those architectures realise the same function.

In a 'closed system' characterisation where the system has a well-defined boundary, given knowledge of all the possible characterisations within the boundary, it would be theoretically possible to define all the relationships between all the characterisations. However, when the number of characterisations and/or relationships between them is extremely large or not yet known, an idealised 'open system' characterisation may be used. For example, in design domains, the realisation of a product requires the realisation of an intricate set of connections between physical components, processes, people and organisations; in complex systems science domains, models of entities often consist of a web of interdependencies between a large number of system elements.[21] An 'open system' characterisation of these scenarios would see the system as interacting with itself (as it would with its environment), and would see the interdependencies between the elements of the system as constantly changing.[22]

4.3 Complexity as overlapping levels

Non-overlapping hierarchies are those in which a related set of characterisations do not overlap with respect to their supersystem-subsystem or supertype-subtype relationships. In the case of overlapping hierarchies, this no longer holds.

4.3.1 Multi-level characterisations and heterarchy

The notion of heterarchy was already introduced in Section 2.1.2. Heterarchical characterisations are ones where several hierarchies *overlap* in a single characterisation. These should be distinguished from characterisations which integrate multiple *non-overlapping* hierarchies (e.g. Simon, 1962, Skyttner, 2005). For pragmatic purposes, heterarchies are decomposed into such non-overlapping characterisations, such as in 'System of Systems' (SoS) characterisations (Maier, 1998), which integrate different resolutions without overlap in scope.

Heterarchies can represent cases where different domains work together to understand a single entity (Alvarez Cabrera et al., 2009; van Beek et al., 2010), since different domains might emphasise different system aspects and consequently 'carve up' the entity in ways that overlap. Figure 12 shows an example

Multi-level characterisations and heterarchy: In the table lamp example, 'base with bulb' maps to the function of converting electrical energy to light energy, while 'base with shade' maps to the function of decorating a table. 'Base' therefore participates simultaneously in two lamp subsystems which in turn map to different functions. In the rainforest ecosystem example, 'producers and non-producers' could together be mapped to the function of realising the food web, while 'producers and abiotic elements' could together map to the function of giving 'consumers' access to energy.

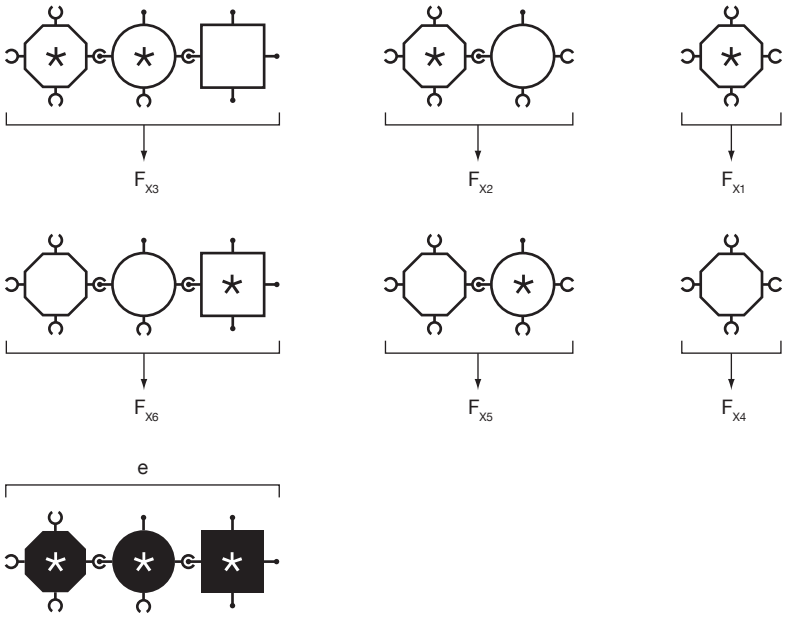


Figure 12. A complex systems characterisation of entity e where functions are mapped to architectures specified at different levels. For example, [C1*+C2*+C3] maps to F_{X3} but [C1+C2+C3*] maps to F_{X6}. The complexity comes from the fact that in order for the realisations of all the functions to be characterised, different levels of abstraction and scope overlap, i.e. heterarchy.

of a complex systems characterisation of the entity e introduced in Section 2 based on the heterarchy in Figure 4. In the real world, these different mappings might represent characterisations from different domains, e.g. programmers, software architects and business analysts working on the same software; cognitive psychologists, neuroscientists, cell biologists and molecular biologists studying the brain.

4.3.2 Endogenous and exogenous functions

In both design and scientific domains, the functions being considered in function-structure mapping often relate to different aspects of the system or even to different systems (with different boundaries), resulting in modular architectures which differ substantially from one another (Holtta & Salonen, 2003). For example, in product design, function-structure mappings may be defined with respect to the product's overall function in use (which is typically linked to the satisfaction of user needs and preferences), but they can also be defined with respect to the product's manufacture or contribution to firm strategy. In Biology, one set of functions might relate to an organism's survival; another might relate to its development or to its role in evolution.

To generalise, the functions in a function-structure mapping might originate from the consideration of different systems, and we can dissociate (i) the system for which the architecture is defined (e.g. the product; organism) from (ii) the system determining the functions to which this architecture maps (e.g. user; ecosystem). In the case of (ii), we might draw a distinction between 'endogenous' functions, which are defined with respect to the system in question, and 'exogenous' functions, which are defined with respect to the supersystem in which it operates (see Crilly, 2013; 2015).[23]

The distinction between endogenous and exogenous functions is important because they can be associated with different levels of uncertainty. Failure to realise endogenous functions lies in improper realisation of the system type (e.g. a system part failing). Failure to realise exogenous functions on the other hand, can be attributed to the system's environment, which can change the function-structure mapping. For example, changes in user preferences might mean that elements of the system that could previously satisfy a particular preference no longer can; a new set of conditions in an organism's environment might mean that certain functions of the organism no longer map to the biological elements they were previously mapped to. If knowledge of the system's environment is inferior to knowledge of the system itself, component types mapping to exogenous functions will have higher levels of uncertainty associated with

Endogenous and exogenous functions. In the table lamp example, both 'lamp' and 'candle' might emit light (endogenous function), but it may be the case that only 'lamp' satisfies the needs of a fire-phobic consumer (exogenous function). Similarly, in the rainforest ecosystem example, both 'vampire bat' and 'iguana' reduce the population of 'grasshoppers' (endogenous function) but it may be that only 'iguana' would motivate a human to visit the rainforest (exogenous function).

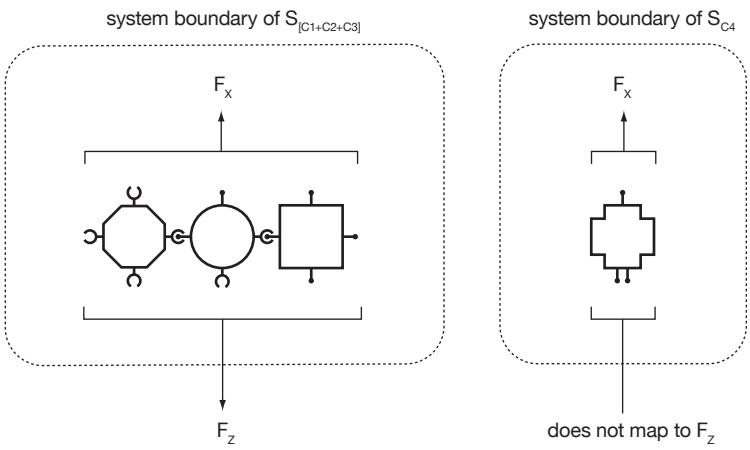


Figure 13. Endogenous and exogenous functions. The architectures [C1+C3+C1] and [C2+C3] map to the same endogenous function F_X but only [C1+C3+C1] maps to the exogenous function F_Y.

them in terms of function fulfilment (e.g. user preferences compared to product specifications; organism behaviour compared to core metabolic functions).

Figure 13 shows how different architectures might map to the same endogenous function but to different exogenous functions. In many cases, endogenous functions and exogenous functions might also be dependent on each other. For example, the realisation of the endogenous function F_X might be dependent on the realisation of F_Y, or vice versa.

4.3.3 Behavioural robustness and flexibility

Although entity change and entity variety can be seen as two distinct concepts, change can also be seen simply as variety observed through time. For example, with an atemporal view, demands to the system due to alterations in physical conditions or consumer preferences (Dahmus et al., 2001) become the same as those made by an environment with a wide range of physical conditions or a market with highly diverse consumer preferences.[24]

While state *transitions* describe the behaviour of a system *instance*, state transition *rules* describe the behaviour of a system *type*.[25,26] State transition rules define the set of state transitions that are realisable (or that must be realised) by instances of the type, thus determining the states that the system can instantiate, its 'state space', depending on its initial state, which also determines the behavioural trajectories it can take.[27] The rules mean that in a given system instance, transitions between states can be 'guided' and 'mutually constraining', so that they follow particular 'trajectories' depending on previous states. This can result in *behavioural* 'robustness' and 'flexibility'.

In the same way that change can be recast as variety, we can give system behaviour (state transitions) an architectural characterisation. In the case of 'behavioural robustness', it is very difficult to get the system to deviate from a particular behaviour. In the case of 'behavioural flexibility', there are few constraints on the states that can be realised by the system, and the architecture of the behaviour has few regularities. Such a system would be chaotic and difficult to manage, predict or understand.

Terms such as 'positive feedback' and 'negative feedback' are used to describe the mechanisms which constrain or 'guide' behaviour (Ashby, 1962; Heylighen & Joslyn, 2001; Babaoglu et al., 2005; Dauscher & Uthmann, 2005; Yamamoto et al., 2007). In the case of positive feedback, a particular state or behaviour increases the likelihood or extent of states or behaviours of the same type, while in the case of negative feedback, it diminishes their extent or likelihood. These two mechanisms and interactions between them form the basis for the 'emergence' of behaviourally robust 'self-* properties' such as self-replication or self-assembly (Babaoglu et al., 2005). They also form the basis of homeostasis or 'autopoiesis', the ability of the system to maintain itself in a viable condition (Maturana & Varela, 1980).

In some complex systems characterisations, the system's *environment* can put the system into a state in which different rules apply or even directly affect which rules apply, thus making different behavioural trajectories available. In even more complicated cases, the system can itself influence its environment to make it more likely to realise particular states, which then reinforce the above effect. Identifying such scenarios is a key endeavour in the complex systems sciences.[28] The environment might also determine the wider implications of the

Positive feedback. In the table lamp example, "thermal runaway" might occur, where an electrical current overheats the conductor. The increase in heat leads to greater thermal conductivity, which leads to more heat being generated, which in turn leads to even higher thermal conductivity (positive feedback). Similarly, in the rainforest ecosystem example, uncontrolled population growth might occur, where individuals continuously reproduce to give rise to more individuals, who then reproduce to give rise to even more individuals (positive feedback).

Negative feedback. In the table lamp example, as the voltage increases, the bulb heats up, which increases electrical resistance, which in turn reduces the temperature (due to reduced current). The reduction in temperature then reduces resistance so that current flows through at a higher rate and heats the filament up again (negative feedback). The temperature-dependent electrical conductivity of the filament keeps this oscillatory pattern in check so long as the filament remains intact and there are no sudden surges of electrical current due to uncontrolled voltage increase.

Similarly, in the rainforest ecosystem example, as the population of a species increases, more resources (e.g. food, water, space) in its environment are required to support it. When the number of individuals reaches the capacity of the environment to support them, fewer individuals will survive and reproduce, leading to a decrease in population (negative feedback). Then when the environment has sufficient resources, this will be reversed and the population will begin to increase again. The capacity of the environment keeps this oscillatory pattern in check provided there are no intervening factors (e.g. entry of competitors or predators, disease).

relationship between functional variety and architectural variety, which in turn can be used to further distinguish between different change-related capabilities. For example, flexibility (on our definition) can mean that a system is fragile in particular types of environment (which might also be characterised in terms of environmental states of a single environment type) because many of the possibilities it has available to it render it non-viable or functionally deficient at some other level of description. On the other hand, in a different set of environments (which might be characterised as different environmental states of a single environment), the system's flexibility might make it resilient because its functional variety allows it to survive.

Architectural robustnesss/flexibility and behavioural robustness/flexibility address different aspects of complexity. In the case of architectural robustness and flexibility, it is the relationship between architecture (which might be the architecture of a system, system type, state or behaviour) and function that we are concerned with. By contrast, in the case of behavioural robustness and flexibility, we are concerned only with the architecture itself (characterised as regularities in behaviour).

The different aspects of complexity introduced in this primer are useful for structuring our understanding of the complexity literature, allowing it to be more easily understood and compared (Table 2).

Table 2. Different notions of complexity mapped to the different aspects of complexity introduced in this primer. The extracts are taken from a special issue on Complex Systems published by the journal *Science* (1999) and other texts found in Section 2 of Ladyman et al.'s (2013) paper, which sought to identify the features of complex systems

Extract	*Open systems characterisation*	*Multi-structural function realisation*	*Context-dependent multi-functionality*	*Architectural robustness / flexibility*	*Heterarchy*	*Behavioural robustness, emergence, self-organisation*
"To us, complexity means that we have structure with variations." (Goldenfeld & Kadanhoff, 1999: p.87)		×		×		
"In one characterization, a complex system is one whose evolution is very sensitive to initial conditions or to small perturbations, one in which the number of independent interacting components is large, or one in which there are multiple pathways by which the system can evolve. Analytical descriptions of such system typically require nonlinear differential equations." (Whitesides & Ismagilov, 1999, p: 89)						×
"A second characterization is more informal; that is, the system is "complicated" by some subjective judgement and is not amenable to exact description, analytical or otherwise." (Whitesides & Ismagilov, 1999: p. 89)	×			×		
"In a general sense, the adjective "complex" describes a system or component that by design or function or both is difficult to understand and verify… complexity is determined by such factors as the number of components and the intricacy of conditional branches, the degree of nesting, and the types of data structures." (Weng et al., 1999: p.92)				×		
"Complexity theory indicates that large populations of units can self-organize into aggregations that generate pattern, store information, and engage in collective decision-making." (Parrish & Edelstein-Keshet, 1999: p.99)						×
"Complexity in natural landform patterns is a manifestation of two key characteristics. Natural patterns form from processes that are non-linear, those that modify the properties of the environment in which they operate or that are strongly coupled; and natural patterns form in systems that are open, driven from equilibrium by the exchange of energy, momentum, material, or information across their boundaries." (Werner, 1999: p.102)	×					×
"A complex system is literally one in which there are multiple interactions between many different components." (Rind, 1999: p.105)				×		
"Common to all studies on complexity are systems with multiple elements adapting or reacting to the pattern these elements create." (Arthur, 1999: p.107)	×		×	×		
"In recent years the scientific community has coined the rubric 'complex system' to describe phenomena, structure, aggregates, organisms, or problems that share some common theme: (i) They are inherently complicated or intricate…; (ii) they are rarely completely deterministic; (iii) mathematical models of the system are usually complex and involve non-linear, ill-posed, or chaotic behaviour; (iv) the systems are predisposed to unexpected outcomes (so-called emergent behaviour)." (Foote, 2007: p.410)	×			×	×	×
"Complexity starts when causality breaks down" (Nature Editorial, 2009).	×			×	×	

5. CONCLUSIONS

This primer has introduced a domain-neutral framework for understanding the relationships between different aspects of complexity and modularity in different systems characterisations (see Section 2). We defined three core aspects of modularity (structural encapsulation, function-structure mapping, and interfacing) and two further abstractions from them (function-driven encapsulation and interface compatibility) (Section 3). These were then explicitly related to different aspects of complexity (Section 4).

Table 3 summarises how different aspects of complexity relate to more fundamental systems constructs and to the different aspects and abstractions of modularity. The extent to which an entity is considered to be a 'complex system' or a 'modular system' depends on how the entity is characterised. Systematically relating different aspects of complexity to different aspects of modularity permits complex systems problems to be characterised and re-characterised in different ways to find suitable solutions. It also allows methods from different domains to be applied to similar problems that might otherwise seem unrelated to each other. In particular, we point to the following opportunities for system design to leverage existing methods (some drawn from the design context, others from scientific contexts).

If we look back through our notes about the table lamp and the rainforest ecosystem examples, we can now see that entities which we might have at first thought of as inherently modular or complex can actually be characterised as either or both.

- Methodologies from Design permitting the systematic characterisation of the relationship between architectural variety and functional variety in a product family at different levels (e.g. 'design for variety', Martin & Ishii, 2002) could be

Table 3. Different complex systems characterisations related to different aspects and abstractions of modularity (non-complex systems characterisations). The relevant sections of the present primer are listed in the columns to the right.

Aspects of complexity	Section	System characterisations, Aspect(s) and abstraction(s) of modularity	Section
Open systems characterisation, shifting system boundaries, ill-defined interfaces.	4.2	Structural encapsulation, interfacing, interface compatibility.	3.11 3.1 3.2.2
Multi-structural function realisation, architectural robustness, evolvability.	4.1.1	Function-structure mapping, function-driven encapsulation.	3.1.2 3.2.1
Context-dependent multi-functionality, architectural flexibility	4.1.2	Function-structure mapping, function-driven encapsulation.	3.1.2 3.2.1
Heterarchy, multi-level representations	2.1.2 4.3.1	Composition, classification, levels, hierarchy.	2.1.1 2.1.2
Endogenous and exogenous functions	4.3.2	Composition, classification, levels, hierarchy, function-structure mapping, function-driven encapsulation.	2.1.1 2.1.2 3.1.2
Behavioural robustness, emergence, self-organisation.	4.3.3	Composition, classification, levels, hierarchy, architecture, functions, properties, behaviours, states	2.1.1 2.1.2 2.2.1 2.2.2 2.2.3 4.3.2

used to analyse the relationship between architectural variety and functional variety of non-designed entities. By generalising the notion of types, architectures and functions, we would be able to include both designed and non-designed system elements within the same characterisation.

- Techniques for exploring system states in the Complex Systems sciences, such as agent-based modelling (Bonabeau, 2002; Axelrod, 2006) could be used to understand the costs and benefits of different architectures with respect to different functions. When a large number of architectural configurations are possible, being able to simulate them and analyse the functional implications of certain family groupings would provide more solid justification for making architectural decisions at product, product family and even product portfolio levels. In addition, for systems with both designed and non-designed elements, we would be able to make better decisions about initialisation states and interventions that would help 'guide' the system into adopting certain architecturally characterised states with desirable properties.
- Community detection and clustering techniques (Palla, 2005 ; Newman, 2006; Lancichinetti & Fortunato, 2009; Fortunato & Castellano, 2012) applied in the Complex Systems sciences could be used to discover different potential product or component 'family' groupings with respect to different functions.
- Static architectural 'patterns' and dynamic 'behavioural motifs' could be shared across domains and application contexts.[29] The domain-neutral nature of our framework would provide a basis for analysing dynamic architectures structurally to identify further trends and commonalities between them. These could be generalised to higher-level design principles and guidelines for designers working on products and problems with complex systems characterisations. In turn, these might be further specialised and adapted for different application contexts.

In both design and scientific contexts, the challenge posed by 'complex systems' comes from having to integrate multiple overlapping characterisations. Those engineering new and emerging technologies are often tackling systems with ill-defined mappings between architectures and their functionally relevant properties. Similarly, those working in the complex systems sciences often struggle to integrate multiple models of a system with overlapping hierarchies, resulting in heterarchical characterisations. The domain-neutral framework we have introduced here allows complex systems problems to be expressed in multiple ways so that the insights, methods and techniques drawn from different domains and application contexts can be appropriately applied to the problems they are most suited to. More fundamentally, being able to characterise and re-characterise entities in different ways encourages the development of innovative solutions that arise from adopting and adapting the methods and techniques of other disciplines and problem domains.

Notes

1. We use the term 'stance' here in the same way that Dennett (1987) has previously used that term. Dennett describes the way in which people take different stances towards entities when predicting their behaviour (e.g. the physical stance, the design stance and the intentional stance).

2. SysML standards are open source and are periodically revised. See http://www.sysml.org

3. CML is developed as part of the Compass project, whose goal is to integrate different engineering notations and methods to support the building of Systems of Systems. See http://www.compass-research.eu/index.html

4. In (Checkland, 1988; Colombo & Cascini, 2014), the term 'holon' is used.

5. We are aware that the deeper semantics, ontological status and metaphysical implications of these two relationships is not uncontroversial (see, for example (Chisholm, 1973; Cleve, 1986) on the composition relationship, and (Tait, 1967; Zalta, 1983; Zemach, 1992) on the super-type-subtype and type-'instance' relationship); our definitions in this case serve simply as pragmatic working definitions to keep the discussion closer to everyday discourse. They do not imply a formal, ontological or metaphysical distinction between types and instances (instances can be seen simply as the entities at the bottom of type hierarchies). However, it should be emphasised that while instances and types 'point to' entities in the world and characterisations, they should not themselves be identified with the entities and characterisations. We can therefore say that a given type is associated with a particular characterisation or set of characterisations (e.g. a particular architecture or a particular set of functional requirements), but it is not the characterisation itself (in the case of instances, it should be obvious that the sequence of words 'an instance of a chair (type)' is not the chair itself).

6. In domain mapping matrix terminology, these different aspects are also known as different "domains".

7. The term "domain" is also used to refer to these different aspects of systems, e.g. domain mapping matrices represent mappings (e.g. Danilovic & Sandkull, 2005) between two different aspects of a system.

8. At the same time, since we make no assumptions about the nature of the elements themselves, if these are functions, then the system architecture will define relationships between them. In this case though, in the system architecture we would not then map these functions (the elements) to other functions just as we would not map physical components to functions or a system composed of physical components. The practice in design domains of relating functions through function decomposition and function commonality (Jiao & Tseng, 1999a; Jiao et al., 2007) can be seen as examples of giving functions architectural characterisations. Similarly, in scientific domains such as neuroscience, functions are often realised by different physical structures or spatio-temporal activation patterns (Coltheart, 1999; Bishop & McArthur, 2005). In such cases, scientists talk about two distinct architectures – a 'physical' architecture (equivalent in this case to the system architecture), which relates the components and subsystems, and a functional architecture, which may map to different physical architectures. As we shall discuss in Section 4, distinguishing between different architectures and being able to relate them to each other gives us a basis for precisely characterising certain forms of complexity (architecturally-based forms of complexity). For example, we can think of a system architecture as being 'degenerate' (non-modular) with respect to the system's functional architecture but still allow that the functional architecture is modular with respect to some other functional architecture (or indeed another system architecture).

9. We are aware of other definitions of 'architecture' that do include references to function, such as those found in (Ulrich, 1995; Baldwin & Clark, 2000; Mikkola & Gassmann, 2003; Chen & Liu, 2005), where the product architecture refers to the scheme by which functions of a product are allocated to its physical components. In the manufacturing literature, there are also definitions of architecture that include reference to the entire product portfolio (a product portfolio consists of a set of product families), which consists of the union of the product architectures of all members in the product family; this defines the function-component mapping of the entire product family (Zamirowski & Otto, 1999; Dahmus et al., 2001).

10. See also (Pahl & Beitz, 1996; Umeda and Tomiyama, 1997; Hubka, 1982) for more details on function decomposition. We are also aware of discussions about the formal validity of functional decomposition. For example, it has been shown that the composition relation does not always meet all the formal requirements of the composition part-whole relationship given by mereology (Vermaas, 2013). However, since our framework does not define the deep semantics of such relationships, we consider this debate outside the scope of this primer. Indeed, without making formal semantic assumptions, we can even permit dependencies and flows between functions such as those found between information processing functions in the model in (Smedt et al., 1996) or the function 'chain' for a screwdriver in (Stone et al., 2000).

11. Note that saying a property is statically or atemporally expressed does not mean that it is itself static or does not have temporal extension, only that its characterisation does not include a dynamic aspect. For example, a system can be said to be 'in a state of change', which obviously refers to a property which is dynamic, but does not include the dynamic aspect in the characterisation. By contrast, saying that a system 'went from one state of change to another state of change' (as in the case of 'epoch shifts' in product lifecycles (Ross & Rhodes, 2007; Ross et al., 2008) or 'regime shifts' in ecosystems (Gunderson & Holling, 2001)) would count as a behaviour since the dynamic aspect is included in the characterisation.

12. For example, in (Sanchez, 2000), the following types of interfaces are distinguished: (i) attachment interfaces, which define how one component physically attaches to another (this is similar to the snap-to-fit perspective taken above); (ii) spatial interfaces that define the physical space (dimensions and position) that a component occupies in relation to other components; (iii) transfer interfaces that define the way one component transfers electrical or mechanical power, fluid, a bitstream, or other primary flow; (iv) control and communication interfaces that define the way that one component informs another of its current state and the way that that other component communicates a signal to change the original component's current state; (v) environmental interfaces that define the effects, often unintended, that the presence or functioning of one component can have on the functioning of another (e.g., through the generation of heat, magnetic fields, vibrations, corrosive vapors, and so forth); (vi) ambient interfaces that define the range of ambient use conditions (ambient temperature, humidity, elevation, and so on) in which a component is intended to perform. In (Sanchez, 2000), there are also user interfaces that define specific ways in which users will interact with a product, but we exclude this

seventh type here because it involves a system-level rather than component-level of description (i.e. it concerns the interface between the system type and user rather than between component types within the system type). Of course, we could treat the system type and user as component types of the user-product supersystem, but this brings us back to talking about within-system interactions. At the same time, we are sensitive to the subtler issues that arise when addressing systems involving both human and 'technical' components (Kroes et al., 2006).

13. We also acknowledge the fact that the distinction between function and property is not always straightforward, e.g. a function might be precisely to deliver a particular property or behaviour.

14. Of course, in most cases, it is likely that function-driven encapsulation also implies property-driven encapsulation (since it is by virtue of realising certain properties that structures map to particular functions), but they can still be considered independently.

15. Component types with the same interface compatibilities are also referred to as 'module variants', 'module types' or even simply 'modules' (Galsworth, 1994).

16. The distinction between types and instances introduced in Section 2 becomes important in discussions of 'sharing'. Sharing between component instances equates to a particular component interacting with several other components, while sharing between component types refers to a particular type of component being able to exist in many different system types.

17. Some firms may even have product 'portfolios', where different families might share either or both architectures and component types (Zamirowski & Otto, 1999; Dahmus et al., 2001). In (Mikkola & Gassmann, 2003), a 'substitutability factor' is introduced which quantifies the impact of substitutability of component types by estimating the number of product families made possible by the average number of interfaces of components for a function.

18. In design domains, methodologies and indices have been introduced to quantify the adaptability, flexibility and robustness of product lines (see e.g. Gu et al., 2009) by analysing the potential for architectural variety. Similarly, in scientific domains, methods and techniques exist to conduct analyses of the similarities and differences between different viable entities (e.g. genotypes of a species).

19. Flexibility can also mean fragility if the majority of functions to which the architectures map are architectures with negative consequences.

20. In the product design context, a 'design for variety' (DFV) framework (Martin & Ishii, 2002) has been introduced which permits a more systematic treatment of the relationship between architectural and functional variety. Within this framework, the 'flexibility'/'robustness' axis is represented by the Generational variety index (GVI), which is a measure of the amount of redesign effort required for future designs of the product while the Coupling index (CI) represents the degree of coupling among product elements (how 'modular' the architecture is).

21. Agent- and equation-based models are used to explore the different possible system behaviours.

22. Open systems characterisations are those where the system itself can change structure, i.e. not only do dependencies exist between elements, but which elements depend on each other can change. In (Giavitto & Michel, 2001), such open systems characterisations are said to be 'dynamical systems with a dynamical structure'. 'Non-linear time variant systems' and 'stochastic non-linear time variant systems' are also means of characterising open systems.

23. Using function in one or other of these ways has precedent in the earliest works of design theory (see review in Winsor & MacCallum, 1994: pp. 166–167). More recently, many variants of this conceptual distinction have been proposed, including device-centric functions and environment-centric functions (Chandrasekaran & Josephson, 2000), action functions and purpose functions (Deng, 2002) and internal functions and external functions (Gzara et al., 2003).

24. We are not denying the fact that often, changes in a system in response to changes in its environment also alter the system's capabilities with respect to future changes in requirements (e.g. a firm that was agile in the past may be unable to handle today's rapidly changing technological landscape because it is now a global conglomerate organisation that is no longer agile). Rather, we are separating out the issue of being able to handle different requirements from a system's identity. For example, we do not make the distinction between the 'spatial' and temporal dimension of the environment made in (Heydari & Dalili, 2014).

25. In the Function Behaviour Stucture (FBS) framework defined in (Gero, 1990; Gero & Mc Neill, 1998) and the Structure Behaviour Function framework (SBF) defined in (Goel & Chandrasekaran, 1989; Goel et al., 2009; Vattam et al., 2011), 'structure' can refer to a state type, architecture and/or their concrete realisation.

26. The term 'transition' is general enough so that it need not require change in system state, but it does require that change can be observed somewhere; this might be change in the system's environment or the passing of time. We also try to avoid reference to time as a dimension in its own right so as to accommodate different interpretations of time, such as the Newtonian (the passing of time is itself a behaviour) versus the relativistic (time realised through behaviours, see, e.g. Callender, 2011).

27. Many sophisticated techniques exist for specifying state transition rules, such as petri nets (Weyns & Holvoet, 2002) or state charts (Kimiaghalam et al., 2002; Stamatopoulou et al., 2007), but a detailed review is outside the scope of this primer.

28. For example, see (Ford & Lerner, 1992; West-Eberhard, 2003; Bar-Yam, 2004; Schlosser & Wagner, 2004; Powell et al., 2005; Hornberg et al., 2006; Roth & Cointet, 2010).

29. Recently, there have been significant efforts in both systems engineering (Ingram et al., 2014) and synthetic biology (Agapakis & Silver, 2009; Agapakis, 2011) to find appropriate representations of such 'patterns' so that they might be better shared within the domain.

References

Agapakis, C. M. (2011) *Biological Design Principles for Synthetic Biology*. Unpublished doctoral dissertation, Harvard University, The Division of Medical Sciences.

Agapakis, C. M. & Silver, P. A. (2009) Synthetic biology: exploring and exploiting genetic modularity through the design of novel biological networks, *Molecular bioSystems* 5 (7), 704–713.

Alexander, C. (1964) *Notes on the Synthesis of Form.* Harvard University Press, Boston, MA.

Allen, K. R. & Carlson-Skalak, S. (1998) Defining product architecture during conceptual design. In *ASME Design Engineering Technical Conference 1998, Atlanta, GA*.

Alvarez Cabrera, A. A., Erden, M. S. & Tomiyama, E. T. (2009) On the Potential of Function-Behavior-State (FBS) Methodology for the Integration of Modeling Tools. In *Proceedings of the 19th CIRP Design Conference - Competitive Design*, Cranfield University.

Arthur, W. B. (1999) Complexity and the economy. *Science* 284 (5411), 107–109.

Ashby, W. R. (1962) Principles of the self-organising system. In H. von Foerster & G. W. Zopf (ed.), *Principles of Self-Organisation* (pp. 108–118). Pergamon, New York, NY.

Axelrod, R. (2006) "Agent-Based Modeling as a Bridge Between Disciplines", In L. Tesfatsion & K.L. Judd (ed.), *Handbook of Computational Economics, Volume 2: Agent-Based Computational Economics* ed. North-Holland, Amsterdam (pp. 1565-1584).

Babaoglu, O., Jelasity, M., Montresor, A., Fetzer, C., Leonardi, S., van Moorsel, A. & van Steen, M., (ed.). (2005) *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations. (Lecture Notes in Computer Science / Theoretical Computer Science and General Issues), Vol. 3460.* Springer, Berlin.

Baldwin, C. Y. & Clark, K. B. (2000) *Design Rules, Vol. 1: The Power of Modularity.* The MIT Press, Cambridge, MA.

Baldwin, C. Y. & Clark, K. B. (1997) Managing in an Age of Modularity. *Harvard Business Review* 75, 84–93.

Bar-Yam, Y. (2004) A mathematical theory of strong emergence using multiscale variety. *Complexity* 9 (6), 15-24.

van Beek, T. J., Erden, M. S. & Tomiyama, T. (2010) Modular design of mechatronic systems with function modeling. *Mechatronics* 20 (8), 850–863.

Belle, R. A. & Kissinger, P. J. (1999) *Bridging the globe: Engineering and construction solutions for sustainable development in the twenty-first century.* Berkeley-Stanford CE&M Workshop, Stanford University, CA.

Bentley, P. J. (2002) *Digital Biology: How Nature Is Transforming Our Technology and Our Lives.* Simon & Schuster, New York.

Bishop, D. V. & McArthur, G. M. (2005) Individual differences in auditory processing in specific language impairment: a follow-up study using event-related potentials and behavioural thresholds, *Cortex* 41 (3), 327–341.

Bonabeau, E. (2002) Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences* 99 (suppl 3), 7280–7287.

Bunge, M. A. (1977) *Ontology I: The furniture of the world – Treatise on basic philosophy.* Reidel, Dordrecht.

Bunge, M. A. (1979) *Ontology II: A world of systems – Treatise on basic philosophy.* Reidel, Dordrecht.

Callender, C., (ed.) (2011) *The Oxford Handbook of Philosophy of Time.* Oxford University Press, Oxford.

Carey, M. (1997) Modularity times three. *Sea Power* 40 (4), 81–84.

Chandrasekaran, B. & Josephson, J. R. (2000) Function in device representation. *Engineering with computers* 16 (3-4), 162–177.

Chang, T. S. & Ward, A. C. (1995) Design-in-modularity with conceptual robustness. In *Proceedings of the 1995 ASME Design Engineering Technical Conferences, 21st International Conference on Advances in Design Automation, Boston, MA*. The American Society of Mechanical Engineers, New York.

Checkland, P. (1988) The case for 'holon'. *Systemic Practice and Action Research* 1 (3) 235–238.

Chen, C.-C. & Crilly, N. (2014) Modularity, redundancy and degeneracy: Cross-domain perspectives on key design principles. In *8th Annual IEEE Systems Conference (SysCon)* (pp. 546–553).

Chen, C.-C. & Crilly, N. (2016) Describing complex design practices with a cross-domain framework: learning from Synthetic Biology and Swarm Robotics. *Research in Engineering Design* 27 (3), 291-305.

Chen, C.-C., Nagl, S. & Clack, C. (2009) Complexity and Emergence in Engineering Systems. In A. Tolk (ed.), *Complex Systems in Knowledge-based Environments: Theory, Models and Applications* (pp. 99–128). Springer, Berlin.

Chen, K.-M. & Liu, R.-J. (2005) Interface strategies in modular product innovation. *Technovation* 25 (7), 771–782.

Chen, W. (1987) A Theory of Modules Based on Second-Order Logic. In *Proceedings of IEEE 1987 Symposium on Logic Programming,* San Francisco, CA (pp. 24–33).

Chisholm, R. (1973) Parts as Essential to Their Wholes. *Review of Metaphysics* 26, 581–603.

Cleve, J. (1986) Mereological Essentialism, Mereological Conjunctivism, and Identity Through Time. *Midwest Studies In Philosophy* 11 (1), 141–156.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E. & Bone, M. (2010) The Concept of Reference Architectures. *Systems Engineering* 13 (1), 14–27.

Colombo, E. F. & Cascini, G. (2014) Complexity as information content and its implications for systems design. In *International Design Conference - DESIGN 2014*, Dubrovnik, Croatia (pp. 1249–1260).

Coltheart, M. (1999) Modularity and cognition. *Trends in Cognitive Sciences* 3 (3), 115–120.

Crilly, N. (2010) The roles that artefacts play: technical, social and aesthetic functions. *Design Studies*, 31 (4), 311–344.

Crilly, N. (2013) Function propagation through nested systems. *Design Studies* 34 (2), 216–242.

Crilly, N. (2015) The proliferation of functions: Multiple systems playing multiple roles in multiple supersystems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 29 (1), 83–92.

Dahmus, J. B., Gonzalez-Zugasti, J. P. & Otto, K. N. (2001) Modular product architecture. *Design Studies* 22 (5), 409–424.

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

Dauscher, P. & Uthmann, T. (2005) Self-Organized Modularization in Evolutionary Algorithms. *Evolutionary Computation* 13 (3), 303–328.

de Weck, O. L., Roos, D. & Magee, C. L. (2011) *Engineering Systems: Meeting Human Needs in a Complex Technological World*. The MIT Press, Cambridge, MA.

De Jong, K. A. (2002) *Evolutionary Computation.* MIT Press (Bradford Books), Cambridge, MA.

Deng, Y. (2002) Function and behaviour representation in conceptual mechanical design. *AI EDAM* 16 (5), 343-362.

Dennett, D. C. (1987) *The intentional stance*. Cambridge, MA: The MIT Press.

Di Marco, P., Eubanks, C. F. & Ishii, K. (1994) Compatibility analysis of product design for recyclability and reuse. *Computers in Engineering* 1, 105–112.

Dressler, F. & Akan, O. B. (2010) A survey on bio-inspired networking. *Computer Networks* 54 (6), 881–900.

Edelman, G. M. & Gally, J. A. (2001) Degeneracy and complexity in biological systems. *Proceedings of the National Academy of Sciences* 98 (24), 13763–13768.

Endy, D. (2005) Foundations for engineering biology. *Nature* 438 (7067), 449–453.

Erden, M. S., Komoto, H., van Beek, T. J., D'Amelio, V., Echavarria, E. & Tomiyama, T. (2008) A review of function modeling: Approaches and applications. *AI EDAM*, 22 (2), 147–169.

Foote, R. (2007) Mathematics and complex systems. *Science* 318 (5849), 410–412.

Ford, D. H. & Lerner, R. M. (1992) *Developmental Systems Theory: An integrative approach.* Sage Publications, Thousand Oaks, CA.

Forrest, S., Balthrop, J., Glickman, M. & Ackley, D. (2005) Computation in the wild. In E. Jen (ed.), *Robust design: Repertoire of biological, ecological, and engineering case studies* (pp. 207–230). Oxford University Press, Oxford.

Fortunato, S. & Castellano, C. (2012) Community structure in graphs. In R. A. Meyers (ed.), *Computational Complexity: Theory, Techniques, and Applications* (pp. 490–512). Springer, Berlin.

Fricke, E. & Schulz, A. P. (2005) Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle. *Systems Engineering* 8 (4), 279–295.

Galsworth, G. D. (1994) *Smart, Simple Design: Using Variety Effectiveness to Reduce Total Cost and Maximize Customer Selection.* Omneo, Essex Junction, VT.

Gao, L. (2000) On Inferring Autonomous System Relationships in the Internet. In *IEEE/ACM Transactions on Networking* (pp. 733–745).

Garud, R. & Kumaraswamy, A. (1993) Changing competitive dynamics in network industries: An exploration of Sun Microsystems' open systems strategy. *Strategic Management Journal* 14 (5), 351–369.

George, G. & Leathrum, J. F. (1985) Orthogonality of concerns in module closure. *Software: Practice and Experience* 15 (2), 119–130.

Gero, J. S. (1990) Design prototypes; a knowledge representation schema for design. *AI Magazine* 11 (4), 26–36.

Gero, J. S. & Mc Neill, T. (1998) An approach to the analysis of design protocols. *Design Studies* 19 (1), 21–61.

Gershenson, J. K., Prasad, G. J. & Allamneni, S. (1999) Modular Product Design : A Life-Cycle View. *Journal of Integrated Design and Process Science* 3 (4), 13–26.

Gershenson, J. K., Prasad, G. J. and Zhang, Y. (2003) Product modularity: Definitions and benefits. *Journal of Engineering Design* 14 (3), 295-313.

Giavitto, J.-L. & Michel, O. (2001) MGS: a rule-based programming language for complex objects and collections 59 (4), 286–304.

Goldstone, R. L. and Sakamoto, Y. (2003) The transfer of abstract principles governing complex adaptive systems. *Cognitive psychology*, 46 (4), 414–466.

Goel, A. & Chandrasekaran, B. (1989) Functional Representation of Designs and Redesign Problem Solving. Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, MI: Morgan Kaufmann Publishers (pp. 1388–1394).

Goel, A., Rugaber, S. & Vattam, S. (2009) Structure, behaviour and function of complex systems: the SBF modelling language. *International Journal of AI in Engineering Design, Analysis and Manufacturing* 23 (1): 23–35.

Goldenfeld, N. & Kadanoff, L. P. (1999) Simple lessons from complexity. *Science* 284 (5411), 87–89.

Gu, P., Xue, D. & Nee, A. Y. C. (2009) Adaptable design: Concepts, methods, and applications. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 223 (11), 1367–1387.

Gunderson, L. & Holling, C. S. (2001) *Panarchy: Understanding Transformations in Systems and Nature.* Island Press, Washington, DC.

Gunji, Y.-P. & Kamiura, M. (2004) Observational heterarchy enhancing active coupling. *Physica D: Nonlinear Phenomena* 198 (1-2), 74–105.

Gzara, L., Rieu, D. & Tollenaere, M. (2003) Product information systems engineering: an approach for building product models by reuse of patterns. *Robotics and Computer-Integrated Manufacturing* 19 (3), 239–261.

Heydari, B. & Dalili, K. (2014) Emergence of Modularity in System of Systems: Complex Networks in Heterogeneous Environments. *IEEE Systems Journal*, 1–9.

Heylighen, F. & Joslyn, C. (2001) Cybernetics and Second-Order Cybernetics. In R. A. Meyers (ed.), *Encyclopedia of Physical Science and Technology.* Academic Press, New York (pp. 155–169).

Holtta, K. M. M. & Salonen, M. P. (2003) Comparing three different modularity methods. In *Proceedings of the ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. ASME, Chicago, Illinois, USA.

Hornberg, J. J., Bruggeman, F. J., Westerhoff, H. V. & Lankelma, J. (2006) Cancer: A Systems Biology disease. *Biosystems* 83 (2–3), 81–90.

Houkes, W. & Vermaas, P. E. (2010) T*echnical Functions: On the Use and Design of Artefacts.* Springer, Berlin.

Huang, C.-C. & Kusiak, A. (1998) Modularity in design of products and systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 28 (1), 66–77.

Hubka, V. & Eder, W. E. (1982) *Principles of engineering design*. Butterworth Scientific, London.

Ingram, C., Payne, R., Perry, S., Holt, J., Hansen, F. O. & Couto, L. D. (2014) Modelling Patterns for Systems of Systems Architectures. In *Proceedings of the 8th Annual IEEE International Systems Conference*, Ottawa, ON.

Ishii, K., Juengel, C. & Eubanks, C. F. (1995) Design for product variety: key to product line structuring. *ASME Design Engineering Division* 83 (2), 499–506.

Jiao, J. & Tseng, M. M. (1999a) Fundamentals of product architecture. *Integrated Manufacturing Systems* 11 (7), 469–483.

Jiao, J. & Tseng, M. (1999b) A methodology of developing product family architecture for mass customization. *Journal of Intelligent Manufacturing* 10 (1), 3–20.

Jiao, J. R., Simpson, T. W. & Siddique, Z. (2007) Product family design and platform-based product development: a state-of-the-art review. *Journal of intelligent Manufacturing* 18 (1), 5–29.

Johnson, J. (2007) Multidimensional Events in Multilevel Systems. In S. Albeverio, D. Andrey, P. Giordano & A. Vancheri (ed.),*The Dynamics of Complex Urban Systems: An Interdisciplinary approach* (pp. 311–334). Physica-Verlag, Heidelburg.

Jose, A. & Tollenaere, M. (2005) Modular and platform methods for product family design: literature analysis. *Journal of Intelligent Manufacturing* 16 (3), 371–390.

Kam, N., Cohen, I. & Harel, D. (2001) The immune system as a reactive system - Modelling T cell activation with statecharts. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments. Symposia on Visual Languages and Formal Methods (VLFM '01), Stresa, Italy* (pp. 15–22).

Kimiaghalam, B., Homaifar, A. & Esterline, A. (2002) A Statechart Framework for Agent Roles that Captures Expertise and Learns Improved Behavior. In *Formal Approaches to Agent-Based Systems, Vol. 2699* (pp. 28–36). Springer Berlin.

Knight, T. F. (2005) Engineering novel life. *Molecular Systems Biology* 1 (1).

Kroes, P., Franssen, M., Poel, I. & Ottens, M. (2006) Treating socio-technical systems as engineering systems: some conceptual problems. *Systems Research and Behavioral Science.* 23 (6), 803–814.

Ladyman, J., Lambert, J. and Wisener, K. (2013) What is a complex system? *European Journal for Philosophy of Science* 3 (1), 33-67.

Lancichinetti, A. & Fortunato, S. (2009) Community detection algorithms: A comparative analysis. *Physical Review E* 80 (5), 056117+.

Luzeaux, D., Ruault, J.-R. & Wippler, J.-L., (ed.). (2013) *Complex Systems and Systems of Systems Engineering.* ISTE/Wiley, London.

Maier, M. W. & Rechtin, E. (2009) *The Art of Systems Architecting.* CRC Press, Boca Raton, FL.

Maier, M. W. (1998) Architecting principles for systems-of-systems. *Systems Engineering* 1 (4), 267–284.

Marshall, R., Leanrey, P. G. & Botterell, O. P. (1998) Enhanced Product Realisation through Modular Design: An Example of Product Process Integration. In *Proceedings of Third Biennial World Conference on Integrated Design and Process Technology*, Berlin.

Martin, M. & Ishii, K. (2002) Design for variety: developing standardized and modularized product platform architectures. *Research in Engineering Design* 13 (4), 213–235.

Maturana, H. & Varela, F. J. (1980) *Autopoiesis and cognition: The realization of the living.* Reidel, Boston.

McCulloch, W. S. (1945) A heterarchy of values determined by the topology of nervous nets 7 (2), 89–93.

McManus, H. & Hastings, D. (2006) A framework for understanding uncertainty and its mitigation and exploitation in complex systems. *IEEE Engineering Management Review* 34 (3), 81-94.

Meadows, D. H. & Wright, D. (2008) *Thinking in systems: A primer.* Chelsea Green Publishing, White River Junction, VT.

Mikkola, J. H. & Gassmann, O. (2003) Managing Modularity of Product Architectures: Toward an Integrated Theory. *IEEE Transactions on Engineering Management* 50 (2), 204–218.

Miozzo, M. & Grimshaw, D. (2005) Modularity and innovation in knowledge-intensive business services: IT outsourcing in Germany and the UK. *Research Policy* 34 (9), 1419–1439.

Mitchell, M. (2009) *Complexity: A guided tour.* Oxford University Press.

Nature Editorial (2009) No man is an island (Editorial). *Nature Physics* 5, 1.

Newcomb, P. J., Bras, B. & Rosen, D. W. (1998) Implications of modularity on product design for the life cycle. *Journal of Mechanical Design* 120 (3), 482–490.

Newman, M. E. J. (2006) Modularity and community structure in networks. *Proceedings of the National Academy of Sciences* 103 (23), 8577–8582.

Newman, M. (2010) *Networks: An Introduction.* Oxford University Press, Oxford.

Ottino, J. M. (2004) Engineering complex systems. *Nature* 427 (6973), 399+.

Otto, K. & Sudjianto, A. (2001) Modularization to support multiple brand platforms. In *Proceedings of the ASME Design Engineering Technical Conferences*, Pittsburgh, PA.

Otto, K. N. & Wood, K. L. (2001) *Product design: techniques in reverse engineering and new product development.* Prentice Hall, Upper Saddle River, NJ.

Pahl, G. & Beitz, W. (1996) Developing size ranges and modular products. In K. Wallace (ed.) *Engineering Design* (pp. 405–453). Springer, Berlin.

Palla, G., Derényi, I., Farkas, I. & Vicsek, T. (2005) Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435 (7043), 814–818.

Parnas, D. L. (1972) On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM* 15 (12), 1053-1058.

Parrish, J. K. & Edelstein-Keshet, L. (1999) Complexity, pattern, and evolutionary trade-offs in animal aggregation. *Science* 284 (5411), 99–101.

Pimmler, T. U. & Eppinger, S. D. (1994) Integration analysis of product decompositions. In *Proceedings of the ASME Conference on Design Theory and Methodology. Minneapolis, MN* (pp. 343–351).

Powell, W. W., White, D. R., Koput, K. W. & Owen-Smith, J. (2005) Network Dynamics and Field Evolution: The Growth of Interorganizational Collaboration in the Life Sciences. *American Journal of Sociology* 110 (4), 1132–1205.

Preston, B. (2009) Philosophical theories of artifact function. In A. Meijers (ed.), *Philosophy of technology and engineering sciences* (pp. 213–234). Elsevier, Amsterdam, The Netherlands.

Rind, D. (1999) Complexity and climate. *Science* 284, 105-107.

Ross, A. M. & Rhodes, D. H. (2008) Using Natural Value-Centric Time Scales for Conceptualizing System Timelines through Epoch-Era Analysis. INCOSE International Symposium. Utrecht, The Netherlands.

Ross A. M., Rhodes D. H. & Hastings D. E. (2008) Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. *Systems Engineering* 11 (3): 246–262.

Roth, C. & Cointet, J.-P. (2010) Social and semantic coevolution in knowledge networks. *Social Networks* 32 (1), 16–29.

Ryan, A. J. (2007) Emergence is coupled to scope, not level. *Complex* 13 (2), 67–77.

Ryan, E. T., Jacques, D. R. & Colombi, J. M. (2013) An ontological framework for clarifying flexibility-related terminology via literature survey. *Systems Engineering* 16(1), 99–110.

Sanchez, R. (2000) Modular architectures, knowledge assets and organisational learning: new management processes for product creation. *International Journal of Technology Management* 19 (6), 610–629.

Sargut, G. and McGrath, R. (2011) Learning to Live with Complexity. *Harvard Business Review* 89 (September), 68-76.

Sarkar, S., Dong, A., Henderson, J. A. & Robinson, P. A. (2013) Spectral characterization of hierarchical modularity in product architectures. *Journal of Mechanical Design* 136 (1), 011006+.

Sasai, K. & Gunji, Y.-P. (2008) Heterarchy in biological systems: A logic-based dynamical model of abstract biological network derived from time-state-scale re-entrant form. *Biosystems* 92 (2), 182–188.

Schlosser, G. & Wagner, G. P. (2004) *Modularity in Development and Evolution.* University Of Chicago Press.

Schoettl, F. & Lindemann, U. (2014) Design for System Lifecycle Properties – A Generic Approach for Modularizing Systems. *Procedia Computer Science* 28, 682–691.

Simon, H. A. (1962) The architecture of complexity. *Proceedings of the American Philosophical Society* 106 (6) 467–482.

Skyttner, L. (2005) *General Systems Theory: Problems, Perspectives, Practice.* World Scientific, London.

Smedt, K., Horacek, H. & Zock, M. (1996) Architectures for natural language generation: Problems and perspectives. In G. Adorni & M. Zock (ed.), *Trends in Natural Language Generation: An Artificial Intelligence Perspective* (pp. 17–46). Springer, Berlin.

Software Engineering Standards Committee (2000) *IEEE recommended practice for architectural description of software-intensive systems (Technical Report IEEE Std 1471-2000).* IEEE Computer Society.

Sosale, S., Hashemian, M. & Gu, P. (1997) Product modularization for reuse and recycling, *Concurrent Product Design and Environmentally Conscious Manufacturing, Proceedings of the 1997 ASME International Mechanical Engineering Congress and Exposition*, Dallas, Texas, pp.195–206.

Stamatopoulou, I., Kefalas, P. & Gheorghe, M. (2007) Modelling the dynamic structure of biological state-based systems. *Biosystems* 87 (2-3), 142–149.

Stone, R. B., Wood, K. L. & Crawford, R. H. (2000) A heuristic method for identifying modules for product architectures. *Design Studies* 21 (1), 5–31.

Tait, W. W. (1967) Intensional interpretations of functionals of finite type. *Journal of Symbolic Logic* 32 (2), 198–212.

Tomiyama, T., Umeda, Y., Ishii, M., Yoshioka, M. & Kirayama, T. (1993) A CAD for functional design. *Annals of the CIRP* 42 (1), 143–146.

Tononi, G., Sporns, O. & Edelman, G. M. (1999) Measures of degeneracy and redundancy in biological networks. *Proceedings of the National Academy of Sciences* 96 (6), 3257–3262.

Ulrich, K. (1995) The role of product architecture in the manufacturing firm. *Research Policy* 24 (3), 419–440.

Ulrich, K. T. & Eppinger, S. D. (1995) *Product design and development.* McGraw-Hill, New York, NY.

Ulrich, K. & Tung, K. (1991) Fundamentals of Product Modularity. In *Proceedings of the 1991 ASME Winter Annual Meeting Symposium on Issues in Design Manufacture/ Integration, Vol. 39* (pp. 73–79).

Umeda, Y. & Tomiyama, T. (1997) Functional reasoning in design. *IEEE Expert* 12 (2), 42–48.

Vattam, S. S., Goel, A. K., Rugaber, S., Hmelo-Silver, C. E., Jordan, R. & Gray, S. (2011) Understanding Complex Natural Systems by Articulating Structure-Behaviour-Function Models. *Journal of Educational Technology and Society* 14 (1).

Veeke, H. P. M., Ottjes, J. A. & Lodewijks, G. (2008) *The Delft Systems Approach: Analysis and Design of Industrial Systems*. Springer, London, UK.

Vermaas, P. E. (2013) On the formal impossibility of analysing subfunctions as parts of functions in design methodology. *Research in Engineering Design* 24 (1), 19–32.

Vermaas, P. E. & Dorst, K. (2007) On the conceptual framework of John Gero's FBS-model and the prescriptive aims of design methodology. *Design Studies* 28 (2), 133–157.

Walz, G. A. (1980) Design tactics for optimal modularity. In *Proceedings of IEEE Autotestcon 1980* (pp. 281–284), Washington, DC.

Wand, Y. and Weber, R. (1990) Mario Bunge's ontology as a formal foundation for information systems concepts, In P. Weingartner & G.J.W. Dorn (eds.), *Studies on Mario Bunge's Treatise*, Rodopi, Atlanta, 1990 (pp. 79–107).

Weng, G., Bhalla, U. S. & Iyengar, R. (1999) Complexity in biological signaling systems. *Science* 284 (5411), 92–96.

Werner, B. T. (1999) Complexity in natural landform patterns. *Science* 284 (5411), 102–104.

West-Eberhard, M. J. (2003) *Developmental Plasticity and Evolution.* Oxford University Press, Oxford.

Weyns, D. & Holvoet, T. A. (2002) A Colored Petri Net for a Multi-Agent Application. In *Proceedings of the Second Workshop on Modeling of Objects, Components, and Agents* (pp. 121–140), Aaarhaus, Denmark.

Whitacre, J. (2010) Degeneracy: a link between evolvability, robustness and complexity in biological systems. *Theoretical Biology and Medical Modelling* 7 (1), 6+.

Whitehead, A. N. (1919) *An Enquiry concerning the Principles of Natural Knowledge.* Cambridge University Press, Cambridge, UK.

Whitesides, G. M. & Ismagilov, R. F. (1999) Complexity in chemistry. *Science* 284 (5411), 89–92.

Winsor, J. & MacCallum, K. (1994) A review of functionality modelling in design. *The Knowledge Engineering Review* 9 (2), 163–199.

Yamamoto, L., Schreckling, D. & Meyer, T. (2007) Self-replicating and self-modifying programs in fraglets. In *Proceedings of the Second International Conference on Bio-inspired Information; Bio-Inspired Models of Network, Information and Computing Systems (Bionetics 2007)* (pp. 159–167). IEEE, Budapest.

Zalta, E. (1983) *Abstract Objects: An Introduction to Axiomatic Metaphysics.* Springer, Berlin.

Zamirowski, E. & Otto, K. (1999) Identifying product portfolio architecture modularity using function and variety heuristics. In *Proceedings of the 1999 ASME Design Enginee ring Technical Conference*. ASME, Las Vegas, NV.

Zemach, E. (1992) *Types: Essays in Metaphysics.* Brill, Leiden.