Modularity and Complexity: A domain-neutral systems framework

Chih-Chun Chen and Nathan Crilly

Engineering Design Centre, Department of Engineering, The University of Cambridge

Although much work on modularity and complexity exists within different domains (e.g. Computer Science, Biology, Economics), this work is seldom shared between domains due to the domain-specificity of the representations and terminology adopted. This article seeks to address these missed opportunities for cross-domain exchange by introducing a domain-neutral systems characterisation framework and diagrammatic scheme that relates different aspects of modularity (structural encapsulation, function-structure mapping and interfacing) to different aspects of complexity. This framework gives researchers and practitioners a common basis for distinguishing between different types of complex system characterisation, allowing them to connect the discourse, methods and findings from different domains.

1 Introduction

This article introduces a domain-neutral framework that relates different aspects of modularity to different aspects of complexity. Distinguishing between different types of complex systems characterisations gives researchers and practitioners a basis for navigating and relating the discourse, methods and techniques drawn from different domains.

A systems perspective is widely adopted in both Design and Science to characterise the entities being designed or studied. Some of these systems are given the label ‘complex’, referring to the fact that they exhibit properties seen to arise through ‘self-organisation’ or ‘emergence’. The ‘complex system’ label also applies to systems with elements and

interactions that need to be understood at different levels or from different perspectives. When the relationships between these different levels and perspectives are not well-defined, the system can be seen as exhibiting unexpected behaviours.

While 'complexity' in the design context has traditionally been cast in a rather negative light, attempts have also been made to harness complexity in engineering (e.g. 'complexity engineering' (Ottino, 2004), 'learning from nature' (Dressler and 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 can be found in (Fricke and Schulz, 2005; McManus and Hastings, 2006; Ross et al., 2008; Ryan et al., 2013; Schoettl and 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 in today's design and engineering problems which often cut across traditional domain boundaries and involve both designed and non-designed entities. 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 differences in the space of design solutions (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).

For designers, modular architectures permit a system to be divided into more manageable parts which can be designed, produced and modified relatively independently. At the same time, sharing of modules between different systems reduces the resources required for developing the components and subsystems that comprise the system, providing a more efficient means of delivering variety and adaptability. 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 so that different phenomena are modelled as arising through interactions between system elements.

Although many domains have worked on understanding how modularity relates to complexity, they rarely benefit from each others' methods, tools or insights due to domain-specific terminology and a lack of explicitness or precision. The lack of an idealised 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 and 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 solutions and methods. To make the theories, methods and findings from one domain accessible to other domains, we need to consider what modularity and complexity really are in domain-neutral terms, how they relate to each other in systems characterisations.

Unlike existing ontologies (e.g. Bunge, 1977, 1979; Gero, 1990; Goel, 2009; Tomiyama et. al., 1993) and systems modelling frameworks (e.g. SysML[1], CML[2]), our framework is not tied to strict semantics, but serves as a reference language for the discussion of modularity, complexity, and the ways in which they are related. To ensure conceptual explicitness, we give domain-neutral definitions and diagrammatic representations of the key terms introduced. We also use concrete examples drawn from diverse domains where appropriate. The objective is not to comprehensively review the literatures relating to systems characterisations, modularity or complexity but rather to provide an accessible means for researchers and practitioners working in different domains to navigate each other's literature. Therefore we do not endeavour to cite all the 'classic' resources from different domains. Instead, we use citations mainly to refer the reader to more details on the examples or to illustrate terminological discrepancies. For pointers to domain-specific reviews, the reader is advised to consult introductory texts, on modularity in design (e.g. Baldwin and Clark, 2000; Gershenson et. al., 2003; Ulrich and Eppinger, 2003); on modularity in science (Newman, 2006); on complexity in design (Luzeaux et, al, 2011); on complexity in science (Ladyman et. al., 2013; Mitchell, 2009); and on system characterisations generally (Meadows and Wright, 2008). The two appendices, which illustrate application of the framework to modularity and complexity literature also contain references to resources on these topics.

The article is structured as follows. Section 2 introduces a framework for characterising systems, focusing on characterisations that are particularly pertinent to design 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 *interfaces*. Based on these, two abstractions are introduced: *function-driven encapsulation*, which relates structural

encapsulation to function-structure mapping; and *interface compatibility*, which relates modularity to architectural variety. 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 article by summarising the relationships between the different aspects of modularity and complexity.

2 Characterising systems

To discuss modularity and complexity without being tied to the assumptions made by particular domains about systems,[3] we need to have a set of domain-neutral constructs and terms for talking about systems. We use the term '**characterisation**'[4] to refer to any representation, model, specification or description of an entity. Indeed, calling an entity a "system" itself assumes a particular kind of characterisation, which we call a "systems characterisation".

For the purposes of this article, we define a **system** as a set of entities and relationships, where the relationships are connections or interactions between the entities. We call the entities in the system the **elements** of the system, where those elements might themselves be considered systems.[5]

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)[6] so that a particular characterisation of an entity is treated as the entity itself.

In order to avoid confusion between cases where we are referring to an entity 'in the world'[7] 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 the term 'type' to refer to the latter.[8] For example, 'Boeing-747 instance' would be used to refer to a particular Boeing-747 aircraft, while 'Boeing-747 type' would be used to refer to the characterisation of Boeing-747 aircrafts, which would include their 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. 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*.

In addition to defining subsystems, components and supersystems, with respect to a given system, we define an **environment** 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 (See Figure 1).

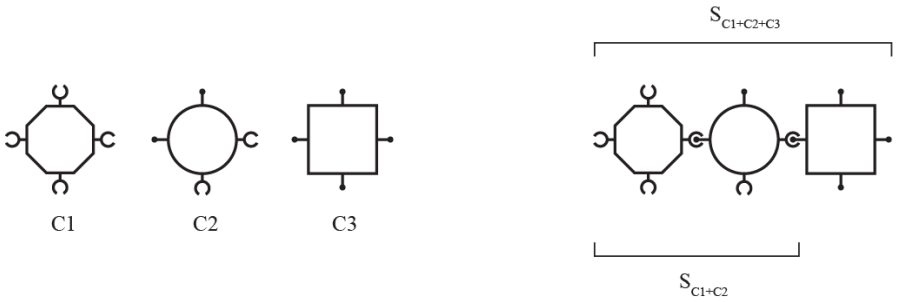

Figure 1: In this diagrammatic scheme, there are different types of entity (represented by different shapes and interfaces). Here, the entities C1, C2 and C3 can be combined to make a system, $S_{C1+C2+C3}$. System S_{C1+C2} is a subsystem of $S_{C1+C2+C3}$. Entity C3 is a component of $S_{C1+C2+C3}$ but is the environment of system S_{C1+C2}. These basic aspects of composition apply to types of entities and also to instances of entities.

Entities can 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. 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 (also see Figure 2):

- 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*.

Figure 3 shows multiple characterisations of an entity which differ in abstraction and composition.

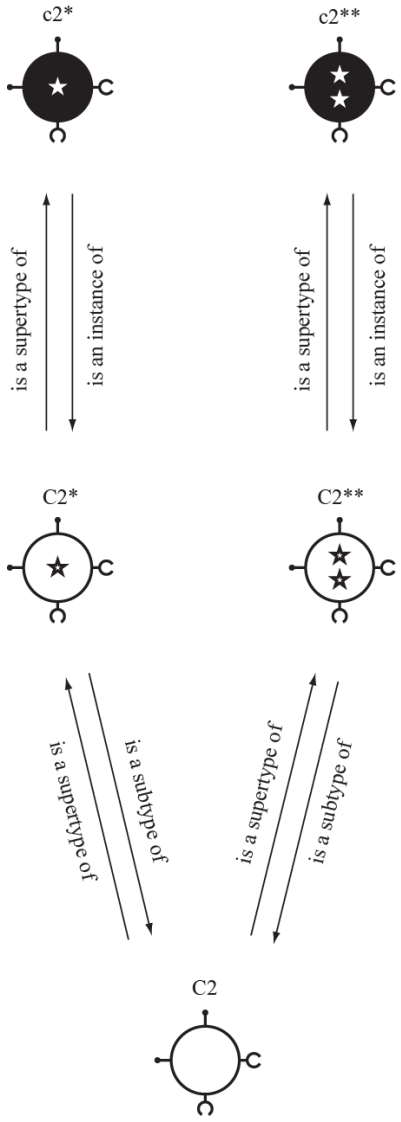

Figure 2: In this diagrammatic scheme, component types (outlined shapes) can be represented at two levels of abstraction: with stars or without stars, where stars represent some feature of the component. These types can also be instantiated (solid shapes). Where components are viewed at a level of abstraction that makes stars visible, there are two options: there might be one 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.**

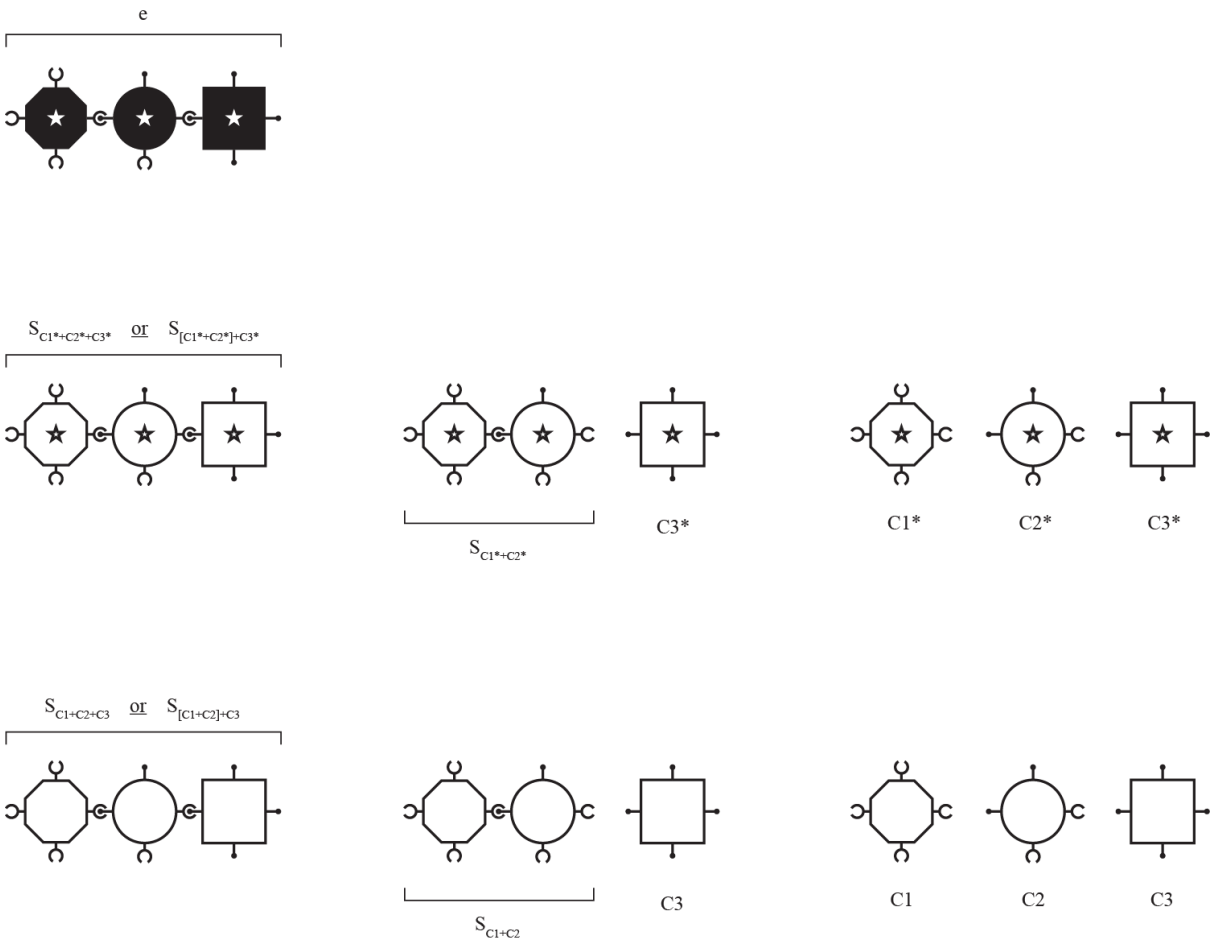

Figure 3: Multiple characterisations of the entity e with different compositions and levels of abstraction. Entity e can be characterised as an instance of the system type $S_{C1*+C2*+C3*}$, composed of C1*, C2* and C3* but it might also be characterised as $S_{[C1*+C2*]+C3*}$, composed of $S_{[C1*+C2*]}$ and C3*. It might also be characterised as an instance of the more abstract system types $S_{C1+C2+C3}$ or $S_{[C1+C2]+C3}$.

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 *composition* relationship is what is known as the ‘scope’ of the characterisation, which is the set of elements involved. Implicit in the *classification* relationship is the ‘resolution’ of the characterisation (also known as ‘granularity’ or ‘level of abstraction’), which is the set of distinctions that can be made between the elements.[9]

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 can not also be its supertype. A compositional hierarchy is a structure in which, if one element is the part of another element, it can not be the whole with respect to that element. 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 C1 in a classification hierarchy (see Figure 4).

In the case of complex systems characterisations, the constraint that levels do not overlap in a single characterisation is not observed. This is what is referred to as a ‘heterarchy’ (McCulloch, 1945, Gunji and Kamiura, 2004; Sasai and Gunji, 2008), ‘panarchy’ (Gunderson and Holling, 2001) or ‘entangled hierarchy’ (Palla, 2005) and can be represented by

hypernetworks (Johnson, 2007; Chen, 2009).[10] Figure 5 depicts a heterarchy that contrasts with the hierarchy described above. We discuss heterarchy further in Section 4.4.

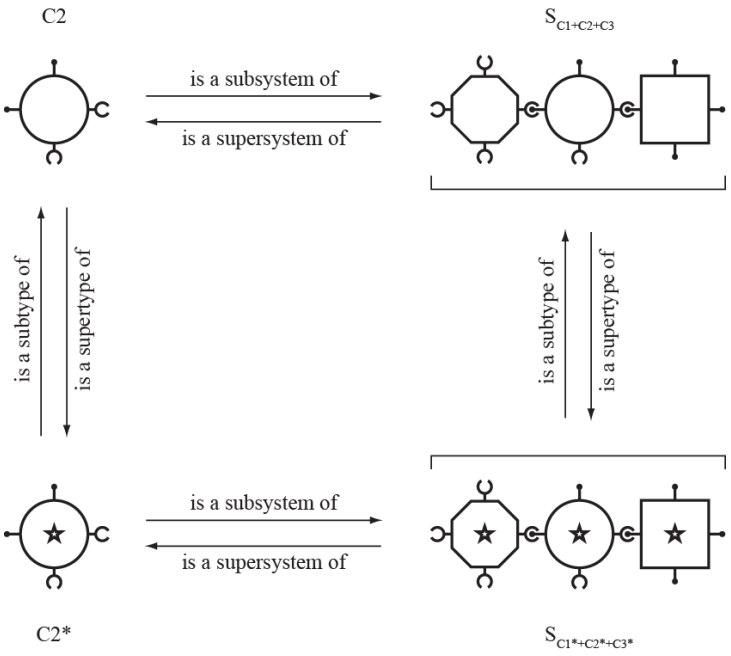

Figure 4: An example of a (clean) hierarchy. C2 is related to $S_{C1+C2+C3}$ in compositional hierarchy, and C2*is related to $S_{C1*+C2*+C3}$ in compositional hierarchy. C2* is related to C2 in classificatory hierarchy, and $S_{C1*+C2*+C3*}$is related to $S_{C1+C2+C3}$ in classificatory hierarchy.

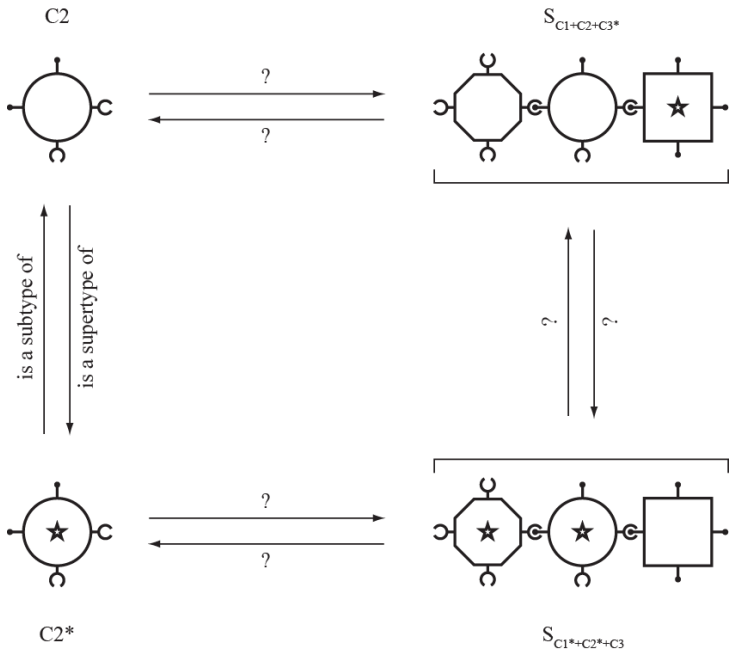

Figure 5: An example of a heterarchy. Although there is a classificatory hierarchical relationship between C2 and C2*, the relationship between $S_{C1*+C2*+C3}$ and $S_{C1+C2+C3*}$ cannot be characterised by classificatory hierarchy alone. The relationship between C2 and $S_{C1*+C2*+C3}$, and between C2* and $S_{C1+C2+C3*}$ cannot be characterised by compositional hierarchy alone.

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*[11] of the system, e.g. a function mapped to an architecture.[12] This section considers three aspects of systems that are most pertinent in Design and Science: 'architecture', 'functions' and 'properties'. The pervasiveness of these three concepts is evidenced by the existence of several ontologies relating them, e.g. (Goel et. al., 2009; Gero, 1990; Tomiyama, 1993) from design domains, and (Bunge, 1977; 1979; Wand and Weber, 1990) in scientific domains.

2.2.1 Architecture

We define an **architectural characterisation** as a description of the relationships between a set of entities. We define a **system architecture** as a characterisation of a system in terms of relationships between its elements.[13] The separation of the *structural* relationships between elements from the *mapping* relationships between elements and functions is consistent with several definitions and discussions of architecture in the literature (e.g. IEEE, 2000; Simon, 1962; Alexander, 1964; Maier, 2009).[14] [15]

2.2.2 Functions

The term "function" is much discussed across various literatures on how systems operate (see reviews in Crilly, 2010; Erden et al., 2008; Houkes & Vermaas, 2010; Preston, 2009; Vermaas & Dorst, 2007), but it is not always easy to see how a general definition can apply across domains (e.g. to both artefacts and organisms). Generally, however, functions describe what a system *should do*, whether that is to satisfy the goals of some agent (e.g. users, designers) or to permit 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, which in turn might be related to its architecture, the functional characterisation of the entity can be considered independently of these other aspects.[16]

2.2.3 Properties

As well as functional and architectural characterisations, a system can also be given characterisations which might broadly be referred to as property-based. Although properties can be dependent on architecture and associated with particular functions, this is not always the case. Therefore, like architectural and functional characterisations, property-based characterisations can be considered independently. 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[17]), we call the property a **state** (Tomiyama et al., 1993). When it is expressed dynamically (in temporally extended terms), we use the term **behaviour**, or more precisely, **state transitions** (Gero, 1990; Kam et al., 2001) and **state transition rules** (see also Section 4.3.2).

3 Aspects and abstractions of modularity

A system characterisation with a compositional hierarchy describes components and subsystems as interacting (or interfacing) with each other in well-defined, well-understood ways and is said to "modular".

Three core aspects of modularity pervade the modularity literature: structural encapsulation, function-structure mapping, and interfacing (Section 3.1). Appendix 1 illustrates how these aspects consolidate different definitions. From these core aspects, we can derive two further abstractions, function-driven encapsulation and interface compatibility (Section 3.2).

3.1 Three core aspects of modularity

The three core aspects of modularity are represented diagrammatically in Figure 6. For a set of system elements to be characterised as a **module**, the following are required:

- Structural encapsulation so they can collectively be treated as a component.
- One-to-one function-structure mapping.
- Well-defined characterisation of their collective interactions with other system elements, interfacing.

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 a set of related system elements to be considered independently from its interactions with other system elements.

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 of its association with the function, we refer to the encapsulation as 'function-driven' (see Section 3.2.1 below).

3.1.3 Interfacing

For a given element, we define its 'interface' as the aspect(s) of the element that allow it to interact with another element or set of 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. Standards, protocols, languages, signals and processes, can all be treated as interfaces.

Which aspect(s) of a system element is treated as its interface depends on the characterisation adopted, which defines the set of elements with respect to which interaction takes place. It is also possible for a characterisation to have multiple types of interfaces, each defined with respect to a different aspect of the system.[18] [19]

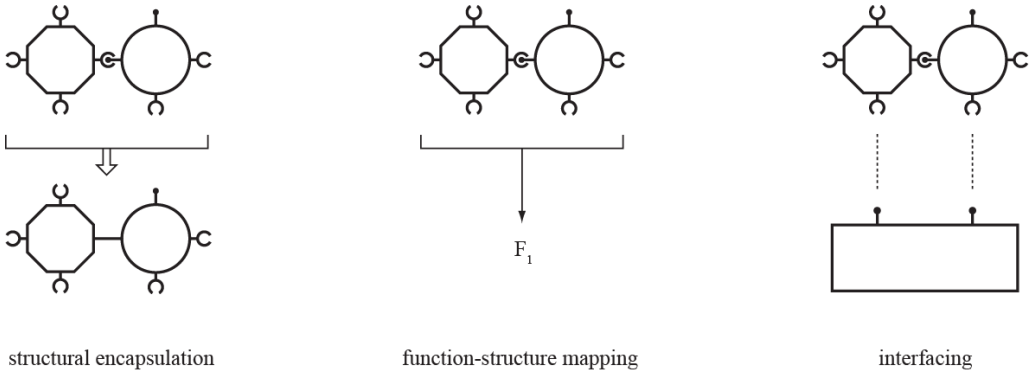

Figure 6: Three aspects of modularity: structural encapsulation (grouping of a set of elements), function-structure mapping (collective mapping of a structured set of elements to a function), and interfacing (how a set of elements interacts with other systems).

3.2 Two abstractions from modularity

From the three core aspects outlined above, we can derive two further abstractions that also pervade the 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 criteria for encapsulation come from function fulfilment (see Figure 7). With function-driven encapsulation, what makes elements within a group strongly connected to each other is that they collectively map to a function, and what makes this set of elements disconnected from or only weakly connected to other elements is the fact that these other elements do not participate in the fulfilment of this function (being 'connected' or 'disconnected' might also be a matter of degree). Function-driven encapsulation can be seen as one of a set of 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 encapsulation where elements are 'connected' when they collectively realise a particular property.[20] However, since it is the relationship between elements and functions that modularity is concerned with, we only consider *function*-driven encapsulation in detail.[21]

In the case of modularity, there is a one-to-one mapping between a given set of connected system elements and a function. This means that every element contributes to the fulfilment of only one function. If every element in a system belongs to such a functionally grouped set, and fulfilment of the system's overall function is completely accounted for by the function-structure mappings to these sets, then the architecture can be said to be completely modular (see Figure 8). 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.

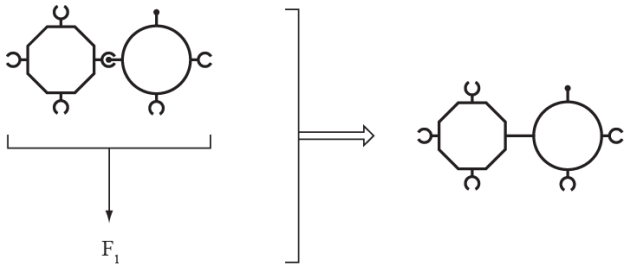

Figure 7. An example of function-driven encapsulation: the structural encapsulation of the module is determined by function-structure mapping.

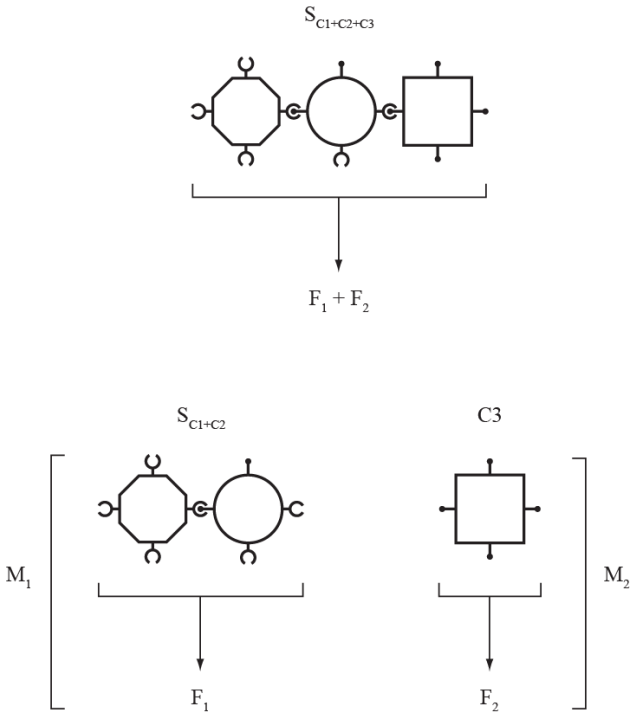

Figure 8: An example of a modular architecture. The system type $S_{C1+C2+C3}$ has a completely modular architecture since all its elements (both S_{C1+C2} and C3) belong to or constitute modules (M_1 and M_2, respectively). In this case, the modules are defined by function-structure mapping.

3.2.2 Interface compatibility

Interface compatibility refers to the compatibility between a component type and other elements of the system it is part of. This compatibility might be a matter of degree and characterised as interaction strength. Interface compatibilities determine which system elements are able to interact with each other, thus providing a characterisation of the system's architectural constraints. In a modular architecture, all elements would be modules or would belong to modules, and hence, interactions between elements in different modules would always be via their interfaces. If all modules in a system had the same mutually compatible interfaces with each other, there would be no architectural constraints i.e. architectural degrees of freedom would be maximised, and every element could be 'repositioned'. This is known as 'sectional' modularity (Ulrich and Tung, 1991; Ulrich, 1995), where every element has the same set of interfaces. At the other extreme, where interfaces minimise architectural degrees of freedom and each element has a specific 'position' or 'role' in the system, we have 'slot' modularity (Ulrich and Tung, 1991; Ulrich, 1995). In 'slot' modularity, each element has a unique set of interfaces, which implies that it has a unique set of interactions with other elements in the system and hence can only be located in one position with respect to them. These two extremes are shown in Figure 9, together with the intermediate case of 'platform modularity', where there is one element which interacts with all the others, and with respect to which the other elements can be repositioned since it interacts with them through identical interfaces.

At the same time, 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), so long as they have the same interface compatibilities,[22] can interact with the same set of other elements. In a modular architecture,

interface compatibilities determine which component types can be swapped or substituted for each other. The terms ‘component-sharing’, (Ulrich, 1995) ‘substitution’ (Garud and Kumaraswamy, 1993, Mikkola 2003) and ‘standardisation’ (Miozzo and Grimshaw, 2005) can be found in the literature to refer to cases where, at a particular level of abstraction, different component types are mapped to the same function (i.e. they are the same module). This ‘component-sharing’, together with overall architectural similarity between products, can be the basis for establishing product ‘families’ (Ulrich, 1995; Galsworth, 1994; Jose and Tollenaere, 2005). The term ‘component-swapping’ (Ulrich, 1995) is used to refer to cases where, at a particular level of abstraction, the component types are mapped to different functions but have the same interface (see Figure 10). If these differences in function have implications for a product’s overall function (i.e. functionally relevant at the system level), they provide the basis for the different variants in product ‘families’.[23]

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 (e.g. a USB bus and the devices plugged into it, an organism and the other organisms in its food web, a protein with many domains engaged in multiple reactions at the same time), while sharing between component *types* refers to a particular type of component being able to exist in many different system types, as in different products in a product family or different species in a genus.

Interface compatibility provides us with a systematic means of characterising and analysing architectural variety as elements distinguishable by their compatibilities with each other being combined.

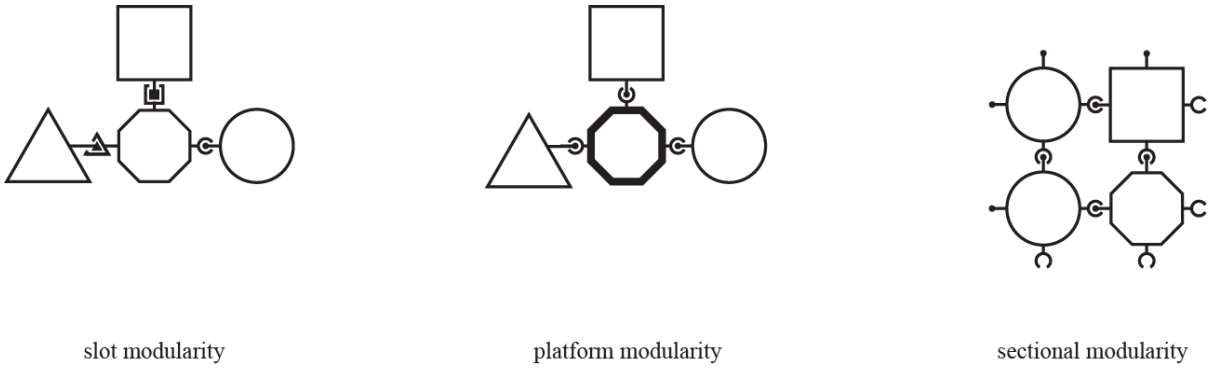

Figure 9: Architectural variety from interface compatibility. In slot modularity, each component type has a unique set of interfaces and can hence only interface with a particular set of component types (in the diagram, the different interfaces are represented by different shaped connectors). In platform modularity, a single component type (the octagon in the diagram) interfaces with all the others with the same kind of interface. In sectional modularity, all component types can interface with all the others via the same kind of interface.

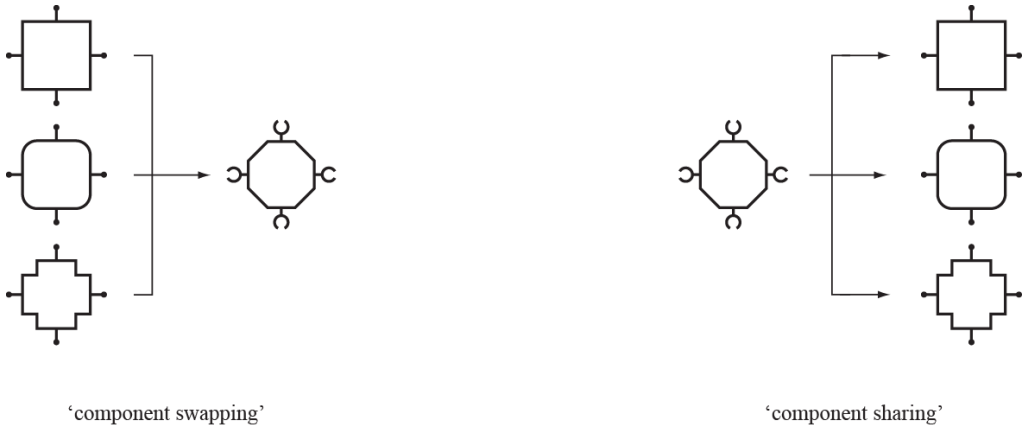

Figure 10: ‘Component-swapping’ always implies ‘component-sharing’, and vice versa. When we are taking the perspective of an element (here, the octagon) that can interact with a variety of other elements, the architecture is characterised as ‘component-swapping’ (different components can be swapped ‘in or out’ of the octagon). When we are taking the perspective of different elements that can interact with the same element (the octagon), the architecture is characterised as ‘component-sharing’ (the octagon is a component that can be shared ‘between’ different elements). Here, we are representing types not instances, and so it is really a component type that is being swapped or shared.

4 Aspects of complexity

The two abstractions of modularity and three aspects of modularity on which they are based can be used to distinguish between different aspects of complexity.

4.1 Function-structure mappings

Function-driven encapsulation ensures one-to-one mapping between function and architecture. Complexity arises when the mapping is not one-to-one.

4.1.1 Multi-structural function realisation

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 at some level). In Design and Engineering, the term 'principle redundancy' (Pahl and Beitz, 1996) describes cases in which multiple architectures can realise the same function. In Biology, the term 'degeneracy' describes cases where, when a particular element is not able to fulfil the function, other means of fulfilling that function are possible (Tononi et al., 1999; Edelman and Gally, 2001; Whitacre, 2010). A functions that was previously associated with a single element might also become distributed among multiple elements. Figure 11 contrasts modular and multi-structural function realisation. To emphasise the level-dependent nature of function-structure mappings, these differ only in the distinguishability of the functions F_X and F_Y.

When different instances of a type can have different architectures, they can have different degrees of multi-structural function realisation (where 'degree' means the number of structures that can realise a function), even with the same mappings. [24] As an example, given the mapping shown at the top of Figure 12, there can be two different ways of achieving redundancy in the function F_X. Similarly, a system instance might be more or less degenerate

when in different states.[25] Indeed the two architectures in Figure 12 might equally represent two different states (see Section 4.4 for architectural characterisations of state and behaviour).

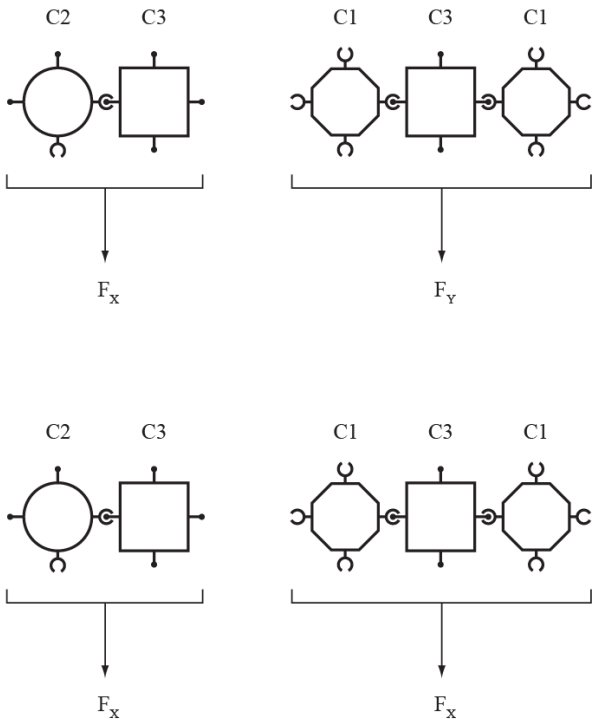

Figure 11: Modular and multi-structural function realisation depending on whether functions are characterised as being equivalent. Top: modular function realisation. If $F_Y \neq F_X$, then [C2+C3] and [C1+C3+C1] map to different functions and are hence different modules. Bottom: multi-structural function realisation. If $F_Y=F_X$, [C2+C3] and [C1+C3+C1] map to the same function are hence alternative architectures for realising that function.

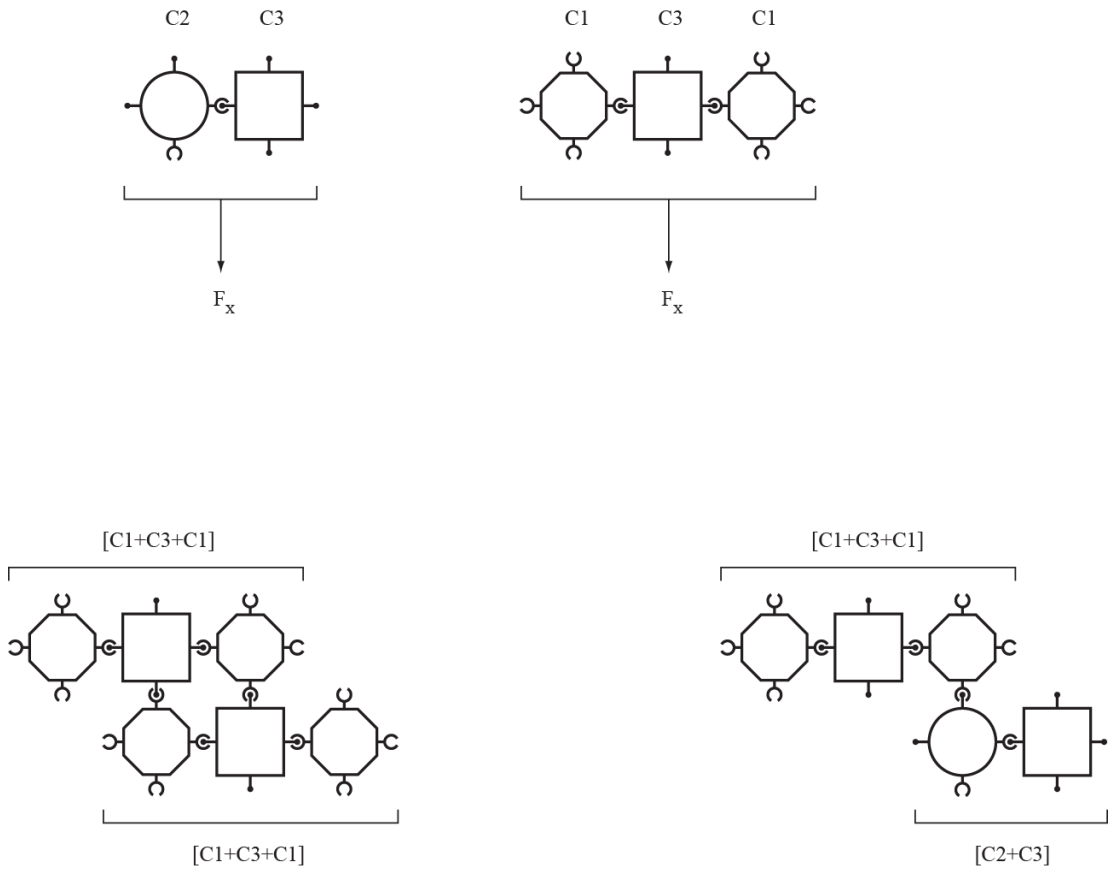

Figure 12. Redundancy through duplicated architectures and distinct architectures. Top row: [C1+C2] maps to F_X and [C1+C3+C1] maps to F_X. Bottom left: redundancy in F_X is provided by an architecture with duplication of [C1+C3+C1]. Bottom right: redundancy in F_X is provided by two distinct architectural realisations, [C1+C3+SC1] and [C1+C2].

4.1.2 Context-dependent multi-functionality

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, or in systems terms, where 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 13 shows how C3 can be characterised as context-dependently multi-functional. When it is connected to C2, it realises F_{X1}, and when it is connected to two instances of C1, it realises F_{X2}.

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 a purchase by a consumer depends on the purchasing activities of other consumers.

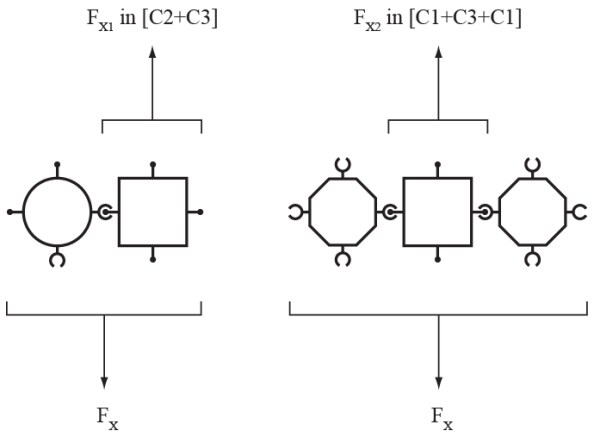

Figure 13: An example of context-dependent multifunctionality: the same component (in this case, the square, C3) realises different functions by participating in different architectures, even if those architectures realise the same function.

4.2 System boundaries and functions

A modular system has subsystems (the modules) with well-defined boundaries (interfaces) and perfect compositional hierarchy. “Complexity” arises when boundaries are ill-defined or changing.

4.2.1 Open and closed systems

Open systems are systems whose elements are interchangeable with the elements of their environment, while closed systems are those whose elements are not.

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.[26] 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.[27]

4.2.2 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 and 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 (e.g. product requirements; organism viability), which are defined with respect to the system in question, and 'exogenous' functions (e.g. user preferences, ecosystem role), which are defined with respect to the supersystem in which it operates (see Crilly, 2013).[28]

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 them in terms of function fulfilment (e.g. the elements of the product associated with user preferences would have greater uncertainty associated with them than those associated with specified product functional requirements; biological elements associated with organism-level behaviour would have greater uncertainty than those associated with core metabolic functions).

Figure 14 shows how different architectures might map to the same endogenous function but to different exogenous functions. In many cases, endogenous functions and exogenous 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.

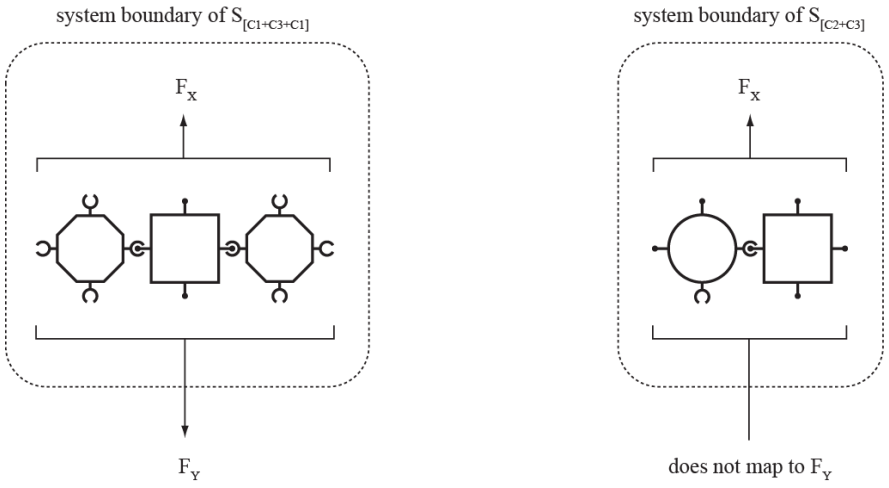

Figure 14: Endogenous and exogenous functions: F_x is an endogenous function with respect to the system types $S_{[C1+C3+C1]}$ and $S_{[C2+C3]}$ while F_Y is an exogenous function with respect to these two system types. The architectures [C1+C3+C1] and [C2+C3] map to the same endogenous function F_X but only [C1+C3+C1] maps to the exogenous function F_Y.

4.3 Variety and change

Interface compatibility and function-driven encapsulation in modular architectures imply well-defined relationships between functional variety and architectural variety, making functional variety straightforward to analyse and manage. In non-modular architectures, the relationship between functional variety and architectural variety is less well-defined.

4.3.1 Architectural robustness and flexibility

In modular architectures, function-driven encapsulation and interface compatibility mean that functional variety is proportional to architectural variety.

In the case of multi-structural function realisation, architectural variety is high with respect to function. 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 15). On the other hand, it makes the function-structure mappings more difficult to analyse , and when there is failure, it can be difficult to identify the elements involved.

In the case of muti-functionality, functional variety is high with respect to an architecture. When it is not well-understood which contexts different functions are realised in, 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 define the terms ‘architecturally robust’ and ‘architecturally flexible’ as follows:

- 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).
- A system is **architecturally flexible**[29] if variety in function is high with respect to architectural variety (in the limit, every architectural variation would be functionally relevant and the ratio would be 1).[30]

Architectural robustness is positively associated with evolvability (Whitacre, 2010). 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 14, having both [C2+C3] and [C1+C3+C1] available as possibilities would make the system both *architecturally robust* with respect to F_X (see Figure 15) 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_Y to be realised, there would be a selection pressure in favour of the architecture [C1+C3+C1]. 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_Y. 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 the 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.[31]

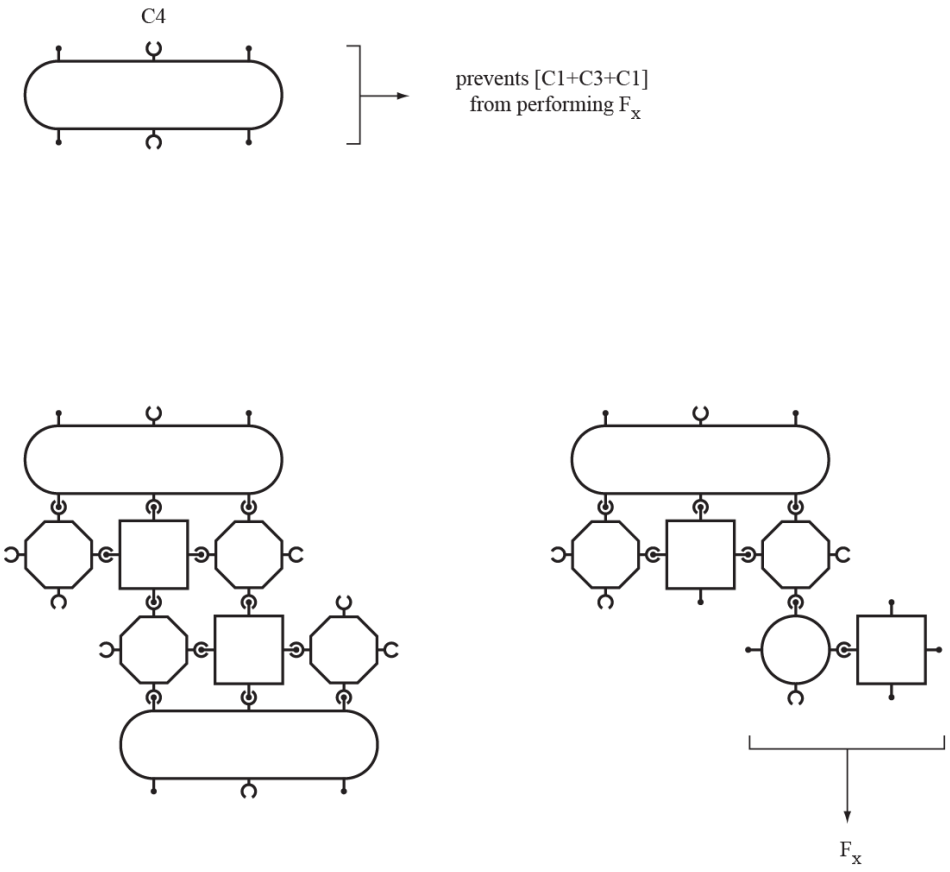

Figure 15. An example of how multi-structural function realisation provides robustness. As in Figure 13, [C2+C3] and [C1+C3+C1] are both mapped to F_X but a new component type, C4 is introduced, which prevents the architecture [C1+C3+C1] from realising F_X. In the bottom left architecture, the presence of C4 prevents $S_{[C1+C3+C1]+[C1+C3+C1]}$ from performing F_X. In the bottom right architecture, the presence of C4 does not prevent the system type $S_{[C1+C3+C1]+[C2+C3]}$ from performing F_X because $S_{[C2+C3]}$ is unaffected by C4. The multi-structural function realisation architecture of $S_{[C1+C3+C1]+[C2+C3]}$ allows it to be more robust than $S_{[C1+C3+C1]+[C1+C3+C1]}$ with respect to performing F_X since it can do so in the presence of C4.

4.3.2 Behavioural robustness and flexibility

Although change and 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.[32]

While state *transitions* describe the behaviour of a system *instance*, state transition *rules* describe the behaviour of a system *type*.[3334] 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' (see Figure 16 and Figure 17).[35] 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, consequently regularities in the architecture of the behaviour 'emerge'.[36] 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 and understand.

Terms such as 'positive feedback' and 'negative feedback' are used to describe the mechanisms which constrain or 'guide' behaviour (Ashby, 1962; Heylighen and Joslyn, 2001; Babaoglu et al., 2005; Dauscher and 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), and homeostasis or 'autopoiesis', the ability of the system to maintain itself in a viable condition (Maturana and 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 (see Figure 18), thus making different behavioural trajectories available. For example, one of an aircraft's engines may fail as a result of exposure to airborne volcanic ash, which might put a heavier load on the other engines and make them more likely to fail. Similarly, how a stem cell differentiates might be determined by chemical factors in its environment but once differentiated, it multiplies and 'locks' a part of the system into developing in a particular way, i.e. it makes the subsystem associated with this developmental trajectory more robust (Bateson and Gluckman, 2012).

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.[3738]

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).

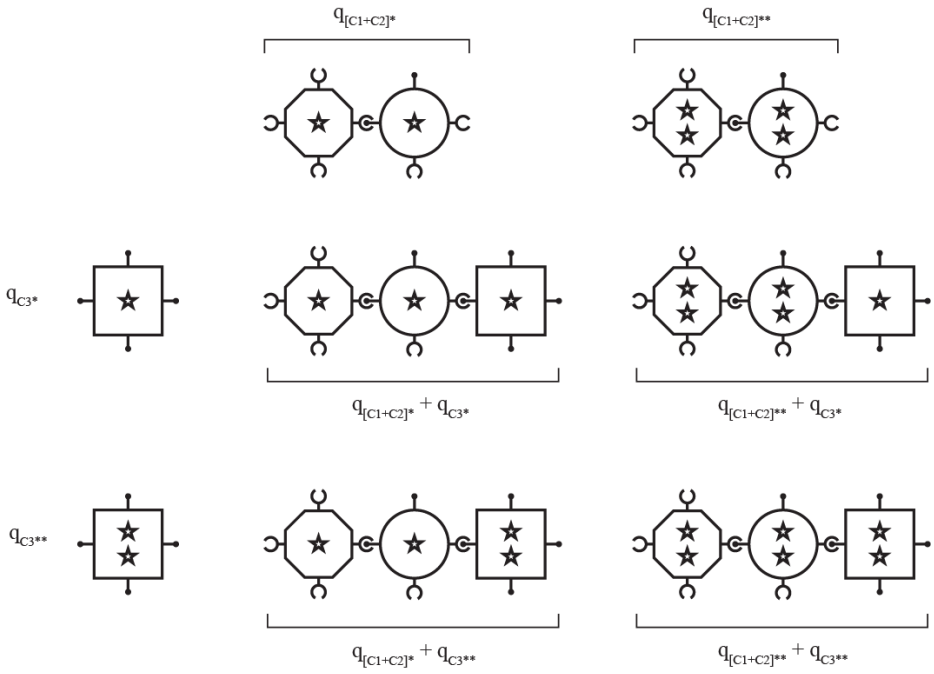

Figure 16. An example of a state space of a system type. The state space $Q_{[C1+C2]+C3}$ of the system type $S_{[C1+C2]+C3}$ is defined as the product $Q_{[C1+C2]} \times Q_{C3}$. This is every possible combination of the states that can be taken by S_{C1+C2} and C3. $Q_{C1+C2}=\{q_{[C1+C2]*}, q_{[C1+C2]**}\}$ and $Q_{C3}=\{q_{C3*}, q_{C3**}\}$.

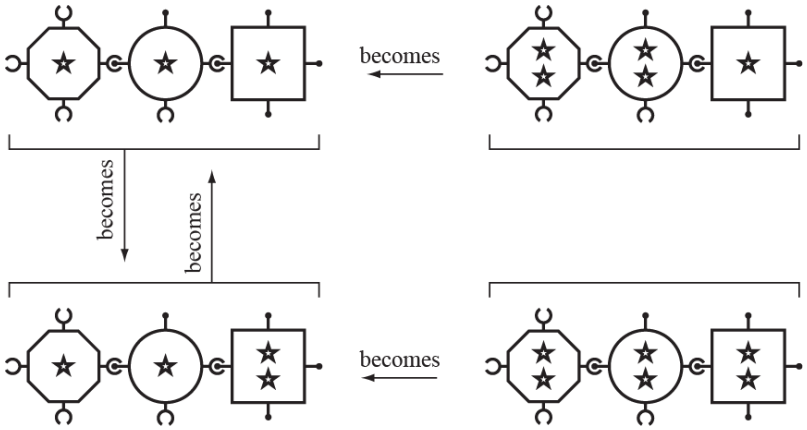

Figure 17. Behavioural rules when system is considered on its own. The state transition rules result in the system iterating between the two left-hand states, irrespective of the initial state.

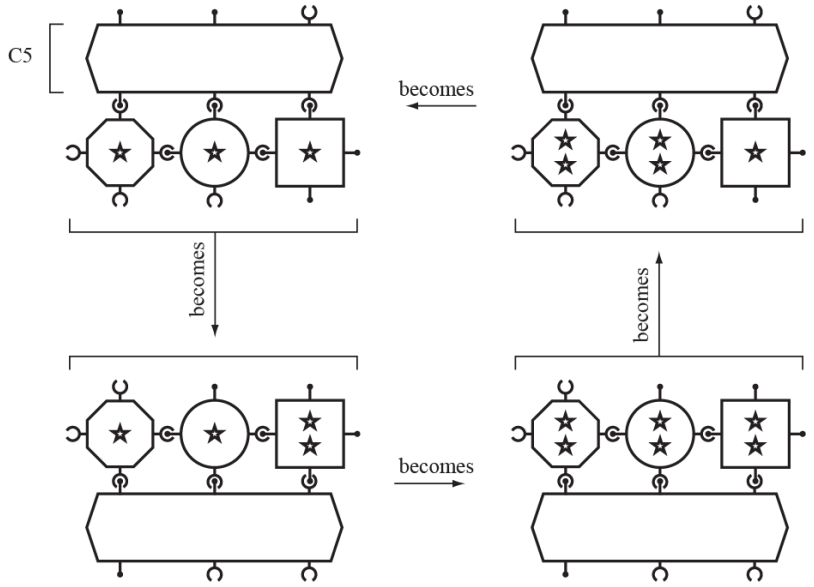

Figure 18. Environmentally influenced behavioural rules. The presence of C5 changes the state transition rules so that the system iterates cyclically through all four states, irrespective of the initial state.

4.4 Multi-level representations 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, 1962b, 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 19 shows an example 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 associated with different domains, e.g. programmers, software architects, business analysts working on the same software; cognitive psychologists, neuroscientists, cell biologists and molecular biologists studying the brain.

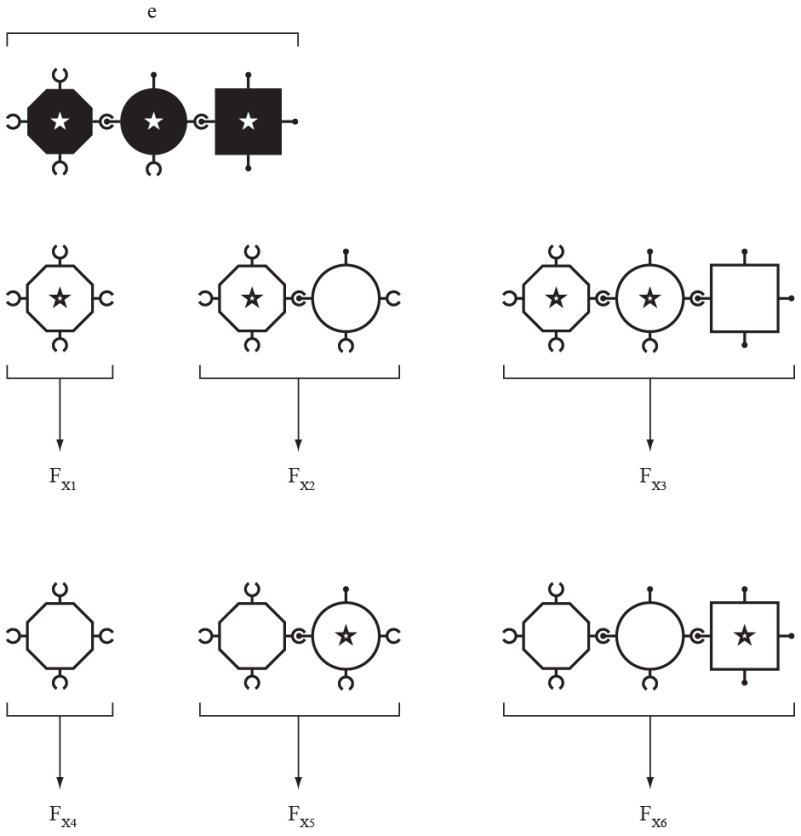

Figure 19. A Complex systems characterisation of entity *e* where functions are mapped to architectures specified at different levels. The characterisation is a complex systems characterisation because in order for the realisations of all the functions to be characterised, different levels of abstraction and scope overlap, i.e. heterarchy.

5 Conclusions

In this article we introduced a domain-neutral framework for understanding the relationships between different aspects of system complexity and modularity in different systems characterisations (see Section 2). We defined three core aspects of modularity 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 1 summarises these, and Appendix 2 further illustrates how they can be used to characterise complex systems discourse.

Types of complexity	Section	System characterisations, Aspect(s) and abstraction(s) of modularity	Section
Open systems characterisation	4.2.1	Structural encapsulation, interfacing	3.1.1, 3.1.3
Multi-structural function realisation.	4.1.1	Function-structure mapping.	3.1.2
Context-dependent multi-functionality	4.1.2	Function-structure mapping.	3.1.2
Architectural robustness.	4.3.1	Function-driven encapsulation, interface compatibility	3.2.1, 3.2.2
Heterarchy.	2.1.2, 4.4	Composition, classification, levels, hierarchy.	2.1.1, 2.1.2
Behavioural robustness, emergence, self-organisation.	4.3.2	Architecture, functions, properties, behaviours, states	2.2.1, 2.2.2, 2.2.3, 4.3.2

Table 1. Different complex systems characterisations related to different aspects and abstractions of modularity (non-complex systems characterisations). The relevant sections of the present paper are listed in the columns to the right.

As noted throughout this article, the extents to which an entity is considered to be a ‘complex system’ or a ‘modular system’ depend on how the entity is characterised. Systematically relating different aspects of complexity to different aspects of modularity permits complex systems problems to be (re-)characterised in different ways to find suitable solutions. It also allows methods from different domains to be applied to similar problems. In particular, we

point to the following opportunities for the engineering design community to leverage existing methods (some drawn from the design context, others from scientific contexts):

- 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 and Ishii, 2002) could be 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 or numerical simulation 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 applied in the Complex Systems sciences could be used to discover different potential 'family' groupings with respect to different functions.
- Both static architectural 'patterns' and dynamic 'behavioural motifs' could be shared across domains and application contexts.[41] 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 problems comes from having to integrate multiple overlapping characterisations. Many emerging technologies are said to involve ‘complex systems’ due to ill-defined mappings between architectures to 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 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 freely applied to the problems they are most suited to.

Acknowledgements

This work was funded by the UK’s Engineering and Physical Sciences Research Council (EP/K008196/1). The authors wish to thank Eloise Taysom for her constructive comments on this paper.

Appendices

Appendix 1: Mapping modularity definitions to aspects of modularity

Table A1 illustrates how the aspects of modularity introduced in this article can be used to characterise different notions of modularity found in the literature. For example, the sources in the first row define modularity in terms of structural encapsulation (or in the case of discourse, this is the aspect emphasised) while the sources in the second row emphasise function-structure mapping.

	Aspects of modularity		
Source of definition	Structural encapsulation	Function-structure mapping	Interfacing
(George and Leathrum, 1985) (Gershenson et. al., 1998) (Jiao and Tseng, 1999b) (Newman, 2010) (Ulrich and Eppinger, 1995)	X		
(Ishii et. al., 1995) (Otto and Wood, 2001)		X	
(Allen and Carlson-Skalak, 1998) (Baldwin and Clark, 1997) (DiMarco et. al., 1994) (Huang and Kusiak, 1998) (Newcomb et. al., 1996) (Sarker et. al., 2013) (Spencer, 1998) (Ulrich and Tung, 1991) (Ulrich, 1995)	X	X	
(Galsworth, 1994) (Walz, 1980)			X
(Chang and Ward, 1995) (Chen, 1987) (Jablan, 1997) (Sosale et. al., 1997)		X	X
(Carey, 1997) (Civil Engineering Research Foundation, 1996) (Pimmler and Eppinger, 1998)	X		X
(Hollta and Salononen, 2003) (Marshall et. al., 1998)	X	X	X

Table A1: Aspects of modularity found in different definitions of modularity.

The list below gives the definitions represented in the table, including the domain-specific definitions reviewed in (Gershenson et. al., 2003):

- In (Allen and Carlson-Skalak, 1998), a module is defined as 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. Modularity is then defined as the degree to which a product's architecture is composed of modules with minimal interactions between modules.
- In (Baldwin and Clark, 1997), modularity refers to the 'building of complex product or process from smaller subsystems that can be designed independently yet function together as a whole).
- In (Carey, 1997), modularity is defined as design with subsystems 'that can be assembled and tested prior to integration… to reduce the time and cost of manufacturing'.
- In (Chang and Ward, 1995), a modular product as 'a function-oriented design that can be integrated into different systems for the same functional purpose without (or with minor) modifications'.
- In (Chen, 1987), modularity refers to 'tools for the user to build large programs out of pieces'.
- In (Civil Engineering Research Foundation, 1996), modularity is defined as using sets of units designed to be arranged in a variety of ways.
- In (DiMarco et. al., 1994), a clump (module) is a collection of 'components and/or subassemblies that share a physical relationship and some common characteristic based on the designer's intent'.
- In (Galsworth, 1994), a module is defined as a group of standard and interchangeable components.

- In (George and Leathrum, 1985), it is required that for a software module, for a given function, there is no access to, informational flow to, or inter-activity between modules.
- In (Gershenson et. al., 1999), it is stipulated that modules contain a high number of components that have minimal dependencies upon and similarities to other components not in the module.
- In (Holtta and Salonen, 2003), a module is defined as a structurally independent building block of a larger system with fairly loose connections to the rest of the system. It is also required that they have well-defined interfaces which allow independent development of the module as long as the interconnections at the interfaces are retained.
- In (Huang and Kusiak, 1998), modularity requires similarity of functional interactions and suitability of inclusion of components in a module.
- In (Ishii et. al., 1995), the term 'modular' refers to the minimisation of the number of functions per component.
- In (Jablan, 1997), modularity is considered as the use of several basic modules for constructing a large collection of different structures.
- In (Jiao and Tseng, 1999b), a module is defined as a physical or conceptual grouping of components.
- In (Marshall et. al., 1998), modules are defined as 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.

- In (Newcomb et. al., 1996), a module is described as a set of components grouped together in a physical structure and by some characteristic based on the designer's intent.
- In (Newman, 2010), a module is defined as a subsystem in which the associations between elements within the subsystem are stronger than the associations between these elements and other elements in the system. This is expressed in network terms. 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.
- In (Otto and Wood, 2001), 'conceptual' modules are introduced. Each of these perform the same functions even if they have different physical compositions.
- In (Pimmler and Eppinger, 1994), the discussion centres around the interactions between elements. Four types of interaction are identified: (i) spatial, the need for adjacency or orientation between elements; (ii) energy, the need for energy transfer between two elements; (iii) information, the need for information or signal transfer between two elements; and (iv) material, the need for material exchange between two elements.
- In (Sarker et. al., 2013), a module is defined as 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.
- In (Sosale et al., 1997), modules are groups of components that can easily be re-used or re-manufactured, also considering material compatibility.

- In (Spencer, 1998), module refers to a ‘manageable portion’ of the code, with minimum interaction between modules (cohesion) and a high degree of interaction within module (high cohesion).
- In (Ulrich and Eppinger, 1995), 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.
- In (Ulrich and Tung, 1991), 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.’
- In (Ulrich, 1995), 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’.
- In (Walz, 1980), modularity is defined as ‘constructed of standardised units of dimensions for flexibility and use’.

Appendix 2: Mapping complex systems discourse to types of complexity

Table A2 illustrates how the types of complexity identified in this article can be used to characterise discourse on complex systems. The list below consists of extracts from a special issue of the journal 'Science' on Complex Systems (Science 2 April, 1999) and other texts found in Section 2 of (Ladyman et. al., 2013), which sought to identify the features of complex systems. For example, extract 1 is concerned with complexity as structural variation, implying multi-structural function realisation and architectural robustness with respect to a particular function.

Extract	Open systems characterisation	Multi-structural function realisation	Context-dependent multi-functionality	Architectural robustness/flexibility	Heterarchy	Behavioural robustness, emergence, self-organisation
1		X		X		
2a						X
2b	X				X	
3					X	
4						X
5	X					X
6					X	
7	X		X		X	
8	X		X		X	X
9	X			X	X	

Table A2. Characterisation of complex systems discourse identified in (Ladyman et. al., 2013, Section 2).

The extract indexes in the first column correspond to the following:

1. “To us, complexity means that we have structure with variations.” (Goldenfeld and Kadanhoff, 19999, p.87)
2. a. “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 ”

   b. “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 and Ismagilov, 1999, p. 89)
3. “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)
4. “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 and Edelstein-Keshet, 1999, p.99)
5. “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)

6. "A complex system is literally one in which there are multiple interactions between many different components." (Rind, 1999, p.105)
7. "Common to all studies on complexity are systems with multiple elements adapting or reacting to the pattern these elements create." (Brian Arthur, 1999, p.107)
8. "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)
9. "Complexity starts when causality breaks down" (Editorial, 2009).

References

Agapakis, C. M. 2014. "Biological Design Principles for Synthetic Biology". PhD thesis, Harvard University.

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

Agarwal, S., Deane, C. M., Porter, M. A. and Jones N. S. 2010. "Revisiting Date and Party Hubs: Novel Approaches to Role Assignment in Protein Interaction Networks". *PLoS Comput Biol* 6(6):e1000817+.

Akao, Y. 2004. *Quality Function Deployment*. Taylor and Francis.

Alexander, C. 1964. *Notes on the Synthesis of Form*. Harvard University Press, new edition, 1964.

Alizon, F., Shooter, S. B. and Simpson, T. W. 2006. "Improving an existing product family based on commonality/diversity, modularity, and cost". *Proceeding of the 11th Design for Manufacturing and the Lifecycle Conference*. ASME.

Allen, K. R. and Carlson-Skalak, S. 1998. "Defining product architecture during conceptual design". *Proceedings of the 1998 ASME Design Engineering Technical Conference*, Atlanta, GA..

Allen, R., Douence, R. and Garlan, D. 1998. "Specifying and analyzing dynamic software architectures". In *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science (1382), edited by Egidio Astesiano, 21–37. Springer Berlin Heidelberg.

Alvarez Cabrera, A. A., Erden, M. S. and Tomiyama, 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.

Aral, S., Muchnik, L. and Sundararajan A. 2009. "Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks". *Proceedings of the National Academy of Sciences* 106(51): 21544–21549.

Arnheiter, E. D. and Harren, H. 2005. "A typology to unleash the potential of modularity". *Journal of Manufacturing Technology Management* 16(7/8): 699–711.

Ashby, W. R. *Principles of the self-organising system*. 1962. Pergamon, New York.

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

Baldwin, C. Y. and Clark, K. B. 1997. "Managing in an age of modularity". *Harvard Business Review* 75, 84-93.

Baldwin, C. Y. and Clark, K. B. 2000. *Design Rules, Vol. 1: The Power of Modularity*. MIT Press.

Bateson, P. and Gluckman, P. 2012. "Plasticity, Robustness, Development and Evolution". *International Journal of Epidemiology* 41(1): 218.

Beckers, R., Holland, O. E. and Deneubourg, J. L. 1994. "From local actions to global tasks: Stigmergy and collective robotics". *Prerational intelligence: Adaptive Behaviour and Intelligent Systems*. Springer.

Bedau, M. A. 2003. "Downward causation and the autonomy of weak emergence". *Principia* 3: 5–50.

Bentley, K. and Clack, C. 2005. “Morphological Plasticity - Environmentally Driven Morphogenesis”. In *Proceedings of the Eighth European Conference on Artificial Life (ECAL ’05), Advances in Artificial Life* 3630: 118–127, Lecture notes in AI series.

Bentley, P. J. 2002. *Digital Biology: How Nature Is Transforming Our Technology and Our Lives*. Simon and Schulster.

Bhattacharyya, R. P., Remenyi, A., Yeh, B. J. and Lim, W. A. 2006. “Domains, Motifs, and Scaffolds: The Role of Modular Interactions in the Evolution and Wiring of Cell Signaling Circuits”. *Annual Review of Biochemistry* 75(1): 655–680.

Bishop, D. V. and 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: a journal devoted to the study of the nervous system and behavior* 41(3): 327–341.

Blackenfelt, M. 2001 “Managing complexity by product modularization”. PhD thesis, Department of Machine Design, Royal Institute of Technology.

Bolker, J. A. 2000. “Modularity in Development and Why It Matters to Evo-Devo”. *American Zoologist* 40(5): 770– 776..

Bonabeau, E. and Dessalles, J. L. 1997. “Detection and emergence”. *Intellectica* 2(25): 85–94.

Bongulielmi, L., Henseler, P., Puls, C. and Meier, M. 2001. “The K- and V-matrix method - an approach in analysis and description of variant products”. In *Proceedings of the International Conference on Engineering Design*.

Boschetti, F. and Gray, R. 2007. “Emergence and Computability”. *Emergence: Complexity and Organisation*.

Brandon, R. N. 1999. “The Units of Selection Revisited: The Modules of Selection”. Biology and Philosophy 14(2): 167–180.

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

Cabigiosu, A., Zirpoli, F. and Camuffo, A. 2013. "Modularity, interfaces definition and the integration of external sources of innovation in the automotive industry". *Research Policy* 42(3): 662–675.

Cai, Y. L., Nee, A. Y. C and Lu, W. F. 2009. "Optimal design of hierarchic components platform under hybrid modular architecture". *Concurrent Engineering* 17(4).

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

Camuffo, A. 2004. *Rolling out a World Car: Globalization, Outsourcing and Modularity in the Auto Industry*. Technical report, University of Venice.

Carey, M. 1997. *Modularity times three*. Sea Power, Navy League of the United States.

Cariani, P. 1992. "Emergence and Artificial Life". In *Artificial life II*, edited by C. G. Langton, C. Taylor, J. D. Farmer, S. Rasmussen. Sante Fe Institute of Studies in Science of Complexity 10: 775-798. Reading, MA: Addison-Wesley.

Carter, G. W., Rush, C. G., Uygun, F., Sakhanenko, N. A., Galas, D. J. and Galitski, T. 2010. "A systems-biology approach to modular genetic complexity". *Chaos* 20(2): 026102+.

Chandrasekaran, B. and Josephson, J. R. 2000. "Function in device representation". *Engineering with Computers* 16(3/4): 162–177.

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

Checkland, P. 2000. "Soft systems methodology: a thirty year retrospective". *Systems Research and Behavioural Science* 17.

Chen, C-C. 2009. “Complex Event Types for Agent-Based Modelling”. PhD thesis, University College London.

Chen, C-C., Nagl, S. and Clack, C. 2009. “Complexity and Emergence in Engineering Systems”. *Complex Systems in Knowledge-based Environments: Theory, Models and Applications* 168: 99–128.

Chen, K-M. and 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 the IEEE Logic Programming Symposium*, 24-33.

Chiriac, N., Holtta-Otto, K., Lysy, D. and Suh, E. S. 2011. “Level of Modularity and Different Levels of System Granularity”. *Journal of Mechanical Design* 133(101007).

Chisholm, R. 1973. “Parts as Essential to Their Wholes”. *Review of Metaphysics* 26: 581–603.

Christensen, C. M. and Rosenbloom, R. S. 1995. “Explaining the attacker’s advantage: Technological paradigms, organizational dynamics, and the value network”. *Research Policy* 24(2): 233–257.

Civil Engineering Research Foundation. 1996. “Bridging the globe: engineering and construction solutions for sustainable development in the twenty-first century”. Highlight of the Engineering and Construction for Sustainable Development in the Twenty-first Century: An International Research Symposium and Technology Showcase.

Clements, P. C. 1996. “A Survey of Architecture Description Languages”. In *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD ’96, Washington, DC, USA. IEEE Computer Society.

Cleve, J. 1996. “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. and Bone, M. 2010. “The Concept of Reference Architectures”. *Systems Engineering* 13(1): 14–27.

Collins, R., Bechler, K. and Pires, S. 1997. “Outsourcing in the automotive industry: From JIT to Modular Consortia”. *European Management Journal* 15(5): 498–508.

Colombo, E. F. and Cascini, G. 2014. “Complexity as information content and its implications for systems design”. In *International Design Conference* - DESIGN 2014.

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

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

Cross, N. 2000. *Engineering design methods: strategies for product design*. Wiley.

Crosson, S., McGrath, P. T., Stephens, C., McAdams, H. H. and Shapiro, L. 2005. “Conserved modular design of an oxygen sensory/signaling network with species-specific output”. *Proceedings of the National Academy of Sciences of the United States of America* 102(22): 8018–8023.

Cummins, R. 1975. “Functional analysis”. *The Journal of Philosophy* 72(20): 741–765.

da Silveira, G., Borenstein, D. and Fogliatto, F. S. 2001. “Mass customization: Literature review and research directions”. *International Journal of Production Economics* 72(1): 1–13.

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

Danilovic, M and 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.

Darley, V. 1994. “Emergent phenomena and complexity”. *Arificial Life* 4: 411–416.

Dasgupta, D., Yu, S. and Nino, F. 2011. “Recent Advances in Artificial Immune Systems: Models and Applications”. *Applied Soft Computing*, 11(2): 1574–1587.

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

de Castro, L. N. and Timmis, J. 2002. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer.

de Jong, K. A. 2002. *Evolutionary Computation*. A Bradford Book

de Weck, O. L., Roos, D. and Magee, C. L. 2011. *Engineering Systems: Meeting Human Needs in a Complex Technological World*. MIT Press.

De Wolf, T. and Holvoet, T. 2005. “Emergence versus self-organisation: different concepts but promising when combined”. In *Engineering Self Organising Systems: Methodologies and Applications*, Lecture Notes in Computer Science 3464: 1–15. Springer Verlag.

Deguet, J., Demazeau, Y. and Magnin, L. 2006 “Elements about the Emergence Issue - A Survey of Emergence Definitions”. *ComPlexUs* 3: 24–31.

Del Vecchio, D., Ninfa, A. J. and Sontag, E. D. 2008. “Modular cell biology: retroactivity and insulation”. *Molecular systems biology* 4.

Delis, I., Panzeri, S., Pozzo, T. and Berret, B. 2014. “A unifying model of concurrent spatial and temporal modularity in muscle activity”. *Journal of Neurophysiology* 111(3): 675–693.

DiMarco, P., Eubanks, C. F. and Ishii, K. 1994. "Compatibility analysis of product design for recyclability". In *Proceedings of the 1994 ASME Design Engineering Technical Conferences – 14th International Conference on Computers in Engineering*.

Doran, D. 2003. "Supply Chain Implications of Modularization". *International Journal of Operations and Production Management* 23(3): 233–257.

Dorigo, M. and Stutzle, T. 2004. *Ant colony optimization*. MIT Press.

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

Du, X., Jiao, J. and Tseng, M. M. 2001. "Architecture of Product Family: Fundamentals and Methodology". *Concurrent Engineering* 9(4): 309–325.

Duray, R., Ward, P. T., Milligan, G. W. and Berry, W. L. 2000. "Approaches to mass customization: configurations and empirical validation". *Journal of Operations Management* 18(6): 605–625.

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

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

Ellis., G. F. R. 2011. "Top-down causation and emergence: some comments on mechanisms". *Interface Focus*.

Endy, D. 2005. "Foundations for engineering biology". Nature 438(7067): 449-453.

Ericsson, A. and Erixon, G. 1999. *Controlling design variants: Modular product platforms*. ASME Press.

Erixon, G. von Yxkull, A. and Arnstrom, A. 1996. “Modularity the Basis for Product and Factory Reengineering”. *CIRP Annals - Manufacturing Technology* 45(1): 1–6.

Eugster, P., Felber, P., Guerraoui, P. and Kermarrec, A. 2003. “The many faces of Publish/Subscribe”. *ACM Computing Surveys* 35(2): 114-131.

Fodor, J. A. 1983. *The Modularity of Mind: An Essay on Faculty Psychology*. A Bradford Book/MIT Press.

Foote, R. 2007. “Mathematics and complex systems. *Science* 318: 410-412.

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

Forrest, S., Balthrop, J., Glickman, M. and Ackley, D. 2005. “Computation in the wild”. In *Robust Design: A Repertoire of Biological, Ecological, and Engineering Case Studies* 207-230. Edited by E. Jen. Oxford University Press.

Fortunato, S. and Castellano, C. 2012. “Community Structure in Graphs”. *Computational Complexity* 490-512.

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

Frigant, V. and Talbot, D. 2005. “Technological Determinism and Modularity: Lessons from a Comparison between Aircraft and Auto Industries in Europe”. *Industry and Innovation* 12(3): 337–355.

Galea, C. A., Wang, Y., Sivakolundu, S. G. and Kriwacki, R. W. 2008. “Regulation of cell division by intrinsically unstructured proteins: intrinsic flexibility, modularity, and signaling conduits”. *Biochemistry* 47(29): 7598–7609.

Galsworth, G. D. 1994. *Smart, Simple Design: Using Variety Effectiveness to Reduce Total Cost and Maximize Customer Selection*. Wiley.

Gao, L. 2000. "On Inferring Autonomous System Relationships in the Internet". In *IEEE/ACM Transactions on Networking* 733-745.

Gallos, L. K., Sigman, M. and Makse, H. A. 2012. "The conundrum of functional brain networks: smallworld efficiency or fractal modularity". *Frontiers in physiology* 3.

Garud, R. and Kumaraswamy, A. 1993. "Changing competitive dynamics in network industries: An exploration of sun microsystems' open systems strategy". *Strat. Mgmt. J.* 14(5): 351–369.

George, G. and Leathrum, L. F. 1985. "Orthogonality of concerns in module closure". *Software Practice and Experience* 15: 119-130.

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

Gero, J. S. and McNeill, Thomas. 1998. "An approach to the analysis of design protocols". *Design Studies* 19(1): 21–61.

Gershenson, J. K., Prasad, G. J. and Allamneni, S. 1999. "Modular Product Design: A Life-Cycle View". *J. Integr. Des. Process Sci.* 3(4): 13–26.

Giavitto, J-L. and Michel, O. 2001. "MGS: A Rule-Based Programming Language for Complex Objects and Collections". In *Proceedings of RULE 2001*, Second International Workshop on Rule-Based Programming.

Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., Clarkson, J. 2009. "Change Propagation Analysis in Complex Technical Systems". *Journal of Mechanical Design* 131. ASME.

Goel, A., Rugaber, S. and 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).

Goldenfeld, N. and Kadanoff, L. P. 1999. “Simple lessons from complexity”. *Science* 284: 87-89.

Goswami, M. and Tiwari, M. K. 2014. “A predictive risk evaluation framework for modular product concept selection in new product design environment”. *Journal of Engineering Design* 25(1-3): 150–171.

Gu, P., Xue, D. and Nee, A. Y. C. 2009. “Adaptable design: Concepts, methods, and applications”. *Journal of Engineering Manufacture* 223(11): 1367-1387.

Gunderson, L. and Holling, C. S. 2001. *Panarchy: Understanding Transformations in Systems and Nature*. Island Press.

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

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

Hartwell, L. H., Hopfield, J. J., Leibler, S. and Murray, A. W. 1999. “From molecular to modular cell biology”. *Nature* 402(6761 Suppl): C47–C52.

Hata, T., Kato, S. and Kimura, F. 2001. “Design of product modularity for life cycle management”. In *Environmentally Conscious Design and Inverse Manufacturing*, Proceedings of Second International Symposium on EcoDesign 2001: Second International Symposium on, pages 93–96, IEEE, 2001.

Hauser, J. R. and Clausing, D. 1988. *The House of Quality*. Harvard Business Review.

He, D., Kusiak, A. and Tseng, T-L B. 1998. “Delayed product differentiation: a design and manufacturing perspective”. *Computer-Aided Design* 30(2): 105–113.

Henderson, R. M. and Clark, K. B. 1990. “Architectural Innovation: The Reconfiguration of Existing Product Technologies and The Failure of Established Firms”. *Administrative Science Quarterly* 35(1): 9–30.

Henson, R. N. 2011. “How to discover modules in mind and brain: the curse of nonlinearity, and blessing of neuroimaging. A comment on Sternberg”. *Cognitive neuropsychology* 28(3-4): 209–223.

Hesse, M. 1963. *Models and Analogies in Science*. Sheed and Ward, London.

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

Heylighen, F. and Joslyn, C. 2001. *Cybernetics and Second-Order Cybernetics*. Academic Press, New York, 3 edition.

Hmelo-Silver, C. E. and Pfeffer, M. G. 2004. “Comparing expert and novice understanding of a complex system from the perspective of structures, behaviors, and functions”. *Cognitive Science* 28(1): 127–138.

Hoetker, G. 2006. “Do modular products lead to modular organizations”. In *Strategic Management Journal*: 501–518.

Holland, J. 2000. *Emergence - from chaos to order*. Oxford University Press.

Holt, J and Perry, S. 2008. “SysML for Systems Engineering”. Professional Applications of Computing. The Institution of Engineering and Technology.

Holtta, K. M. M. and Salonen, M. P. 2003. “Comparing three different modularity methods”. In *Proceedings of DETC: ASME Design Engineering Technical Conferences*. ASME.

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

Hsuan, J. 1999. "Impacts of supplierbuyer relationships on modularization in new product development". *European Journal of Purchasing & Supply Management* 5(3-4): 197–209.

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

Ingram, C., Payne, R., Perry, S., Holt, J., Hansen, F. O. and Couto, L. D. 2014. "Modelling Patterns for Systems of Systems Architectures". In *8th Annual IEEE International Systems Conference*. IEEE Systems Council, IEEE.

Ishii, K., Juengel, C. and Eubanks, C. F. 1995. "Design for product variety: key to product line structuring". In *Proceedings of the 1995 ASME Design Engineering Technical Conferences – 7th International Conference on Design Theory and Methodology*. The American Society of Mechanical Engineers.

Jablan, S. V. 1997. *Modularity in Art*.

Jagannathan, V., Dodhiawala, R. and Baum, L. S. *Blackboard architectures and applications*. 1989.

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

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

Jiao, J., Simpson, T. W. and 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, C. W. 2006. “What are Emergent Properties and How Do They Affect the Engineering of Complex Systems? Reliability Engineering and System Safety”. In *Proceedings of Complex Systems: Reliability Engineering and System Safety* 91(12): 1475–1481.

Johnson, J. 2006. “Hypernetworks for reconstructing the dynamics of multilevel systems”. In *Proceedings of European Conference on Complex Systems*.

Johnson, J. 2007. “Multidimensional Events in Multilevel Systems”. *The Dynamics of Complex Urban Systems*: 311–334. Physica-Verlag HD.

Jose, A. and 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. and Harel, D. 2001. “The immune system as a reactive system - Modelling t-cell activation with statecharts”. In *Proc. Visual Languages and Formal Methods (VLFM ’01)*: 15–22.

Kasperek, D., Maisenbacher, S. and Maurer, M. 2014. “Structure-based Analysis of Dynamic Engineering Process Behaviour”. In 8*th Annual IEEE International Systems Conference*. IEEE.

Kimiaghalam, B., Homaifar, A. and Esterline, A. 2002. “A Statechart Framework for Agent Roles that Captures Expertise and Learns Improved Behavior”. In *Formal Approaches to Agent-Based Systems*, Lecture Notes in Computer Science 2699: 28–36. Springer Berlin/Heidelberg.

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

Kogut, B. and Kulatilaka, N. 1994. “Operating Flexibility, Global Manufacturing, and the Option Value of a Multinational Network”. *Manage. Sci.* 40(1): 123–139.

Kogut, B. and Zander, U. 2009. “Knowledge of the Firm, Combinative Capabilities, and the Replication of Technology”. *Social Science Research Network Working Paper Series*, November 2009.

Kong, F., Ming, X., Wang, L., Wang, P., Zuo, H. and He, L. 2011. “An integrated modularity approach for green product development”. *International Journal of Environmental Technology and Management*: 397–416.

Kotonya, G. and Sommerville, I. 1998. *Requirements engineering: Processes and Techniques*. Worldwide Series in Computer Science. Wiley.

Kroes, P., Franssen, M., Poel, I. and Ottens, M. 2006. “Treating socio-technical systems as engineering systems: some conceptual problems”. *Syst. Res.* 23(6): 803–814.

Kubik, A. 2003. “Toward a formalization of emergence”. *Artificial Life* 9:41–66.

Lai, X. 2008. “Design structure matrix-based product representation for life-cycle process-based modularity”. PhD thesis, Michigan Technological University.

Lancichinetti, A. and Fortunato, S. 2009. “Community detection algorithms: A comparative analysis”. *Physical Review E* 80.

Langlois. R. N. 2000. “Modularity in Technology, Organization, and Society”. *Social Science Research Network Working Paper Series*, February 2000.

Lee, H. L. and Tang, C. S. 1997. “Modelling the Costs and Benefits of Delayed Product Differentiation”. *Management Science* 43(1).

Li, X., Cui, D., Jiruska, P., Fox, J. E., Yao, X. and Jeffreys, J. G. R. 2007. “Synchronization measurement of multiple neuronal populations”. *Journal of neurophysiology* 98(6): 3341–3348.

Limsoonthrakul, S., Dailey, M. N., Srisupundit, M., Tongphu, S. and Parnichkun, M. 2008. “A modular system architecture for autonomous robots based on blackboard and

publish-subscribe mechanisms". In *IEEE International Conference on Robotics and Biomimetics, 2008* (ROBIO 2008), 633–638. IEEE.

Lindemann, U., Maurer, M. and Braun, T. 2009. *Structural Complexity Management: An Approach for the Field of Product Design*. Springer.

Luzeaux, D., Renault, J-R. and Wippler, J-L., eds. 2011. *Complex Systems and Systems of Systems Engineering*. Wiley. 2011.

Maier, M. W. 1998. "Architecting principles for systems-of-systems". *Syst. Engin*. 1(4): 267–284.

Maier, M. W. 2009. *The Art of Systems Architecting*, Third Edition (Systems Engineering). CRC Press, 3.

Marshall, R., Leanrey, P. G. and 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*.

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

Marx, R., Zilbovicius, M. and Salerno, M. S. 1997. "The 'modular Consortium' in a New VW Truck Plant in Brazil: New Forms of Assembler and Supplier Relationship". *Computer Integrated Manufacturing* 8(5): 292–298.

Mason, P. H. 2014. "Degeneracy: Demystifying and destigmatizing a core concept in systems biology". *Complexity*.

Maturana, H. and Varela, F. J. 1980. *Autopoiesis and cognition*. Reidel, Boston.

McAdams, D. A., Stone, R. B. and Wood, K. L. 1999. “Functional Interdependence and Product Similarity Based on Customer Needs”. *Research in Engineering Design* 11(1): 1–19.

McConnell, S. 2004. *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press.

McCulloch, W. S. 1945. “A heterarchy of values determined by the topology of nervous nets”. *Bulletin of Biophysics* 7(2): 89–93.

McManus, H. L., Richards, M. G., Ross, A. M. and Hastings, D. E. “A framework for incorporating “ilities” in tradespace studies”. *A/AA Space 2007 Conference* 1: 941-954.

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

Meadows, D. H. 2008. *Thinking in Systems: A Primer*. Chelsea Green publishing.

Meehan, J., Duffy, A. and Whitfield, R. 2007. “Supporting ‘design for re-use’ with modular design”. *Concurrent Engineering* 15(2): 141–155.

Meunier, D., Lambiotte, R. Fornito, A., Ersche, K. D. and Bullmore, E. T. 2010. “Hierarchical modularity in human brain functional networks”. *Frontiers in Neuroinformatics* 3.

Meyer, B. “Applying ’design by contract’”. 1992. *Computer* 25(10): 40–51.

Mikkola, J. H. 2000. “Product architecture design: implications for modularization and interface management”. In *LINK Workshop, Organizing processes of Learning*, May 2000.

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

Miller, T. D. and Elgard, P. 1998. “Defining modules, modularity and modularization”. In *Proceedings of the 13th IPS Research Seminar*.

Milroy, L. 1987. *Language and social networks*. Wiley-Blackwell.

Mitchell, M. 2009. *Complexity: A Guided Tour*. Oxford University Press.

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

Morrison, M. 1998. “Modelling Nature: Between Physics and the Physical World”. *Philosophia Naturalis* 35: 65–85.

Nakano, T., Moore, M. J., Wei, F., Vasilakos, A. V. and Shuai, J. 2012. “Molecular communication and networking: opportunities and challenges”. *IEEE transactions on nanobioscience* 11(2): 135–148.

Navarro-serment, L. E., Grabowski, R., Paredis, C. J. J. and Khosla, P. K. 1999. “Modularity in Small Distributed Robots”. In *Proceedings of the SPIE conference on Sensor Fusion and Decentralized Control in Robotic Systems II*: 297–306.

Newcomb, P. J., Bras, B. and Rosen, D. W. 1998. “Implications of modularity on product design for the life cycle”. *Journal of Mechanical Design*.

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

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

Oizumi, K., Aruga, K. and Aoyama, K. “Common module product family design in view of compromise in function and integrity”. In *International Design Conference - DESIGN 2014*.

Onnela, J. P., Saramaki, J., Hyvonen, J., Szabo, G., Lazer, D., Kaski, K., Kertesz, J. and Barabasi, A. L. 2007. “Structure and tie strengths in mobile communication networks”. *Proceedings of the National Academy of Sciences* 104(18): 7332–7336.

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

Otto, K. N. and Sudjianto, A. 2001. “Modularization to support multiple brand platforms”. In *Proceedings of DETC: ASME Design Engineering Technical Conferences*.

Otto, K. N. and Wood, K. L. 2001. *Product design: techniques in reverse engineering and new product development*. Prentice Hall.

Pahl, G. and Beitz, W. 1984. *Developing size ranges and modular products*, 315–361.

Pahl, G., Beitz, W. and Wallace, K. 1996. *Engineering Design: Systematic Approach*. Springer-Verlag.

Palla, G. 2005. “Uncovering the overlapping community structure of complex networks in nature and society”. *Nature* 435: 814-818.

Parrish, J. K. and Edelstein-Keshet, L. 1999. “Complexity, pattern, and evolutionary trade-offs in animal aggregation”. *Science* 284: 99-101.

Payne, R. and Fitzgerald, J. 2011. “Contract-based interface specification language for functional and non-functional properties”. Technical report, Newcastle University.

Pil, F. K. and Cohen, S. K. 2006. “Modularity: Implications for imitation, innovation, and sustained advantage”. *Academy of Management Review* 31(4): 995–1011.

Pilot, F. and Lecuit, T. 2005. “Compartmentalized morphogenesis in epithelia: From cell to tissue shape”. *Dev. Dyn.* 232(3): 685–694.

Pimmler, T. U. and Eppinger, S. D. 1994. “Integration analysis of product decompositions”. In *Proceedings of ASME Conference on Design Theory and Methodology*: 343-351. Minneapolis, MN.

Pires, S. 1998. “Managerial Implications of the Modular Consortium Model in a Brazilian Automotive Plant”. *International Journal of Operations and Production Management* 18(3).

Pires, S. 2002. “New Productive System in the Auto Industry: the Current Situation of Three Innovative Plants in Brazil”. *International Journal of Automative Technology and Management* 2(1).

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

Price, C. J. J. and Friston, K. J. J. 2002. “Degeneracy and cognitive anatomy”. *Trends in cognitive sciences* 6(10): 416–421.

Rescher, N. 1955. “Axioms for the part relation”. *Philosophical Studies: An International Journal for Philosophy in the Analytic Tradition* 6(1): 8–11.

Ronald, E. and Sipper, M. 1999. “Design, observation, surprise! A test of emergence”. *Artificial Life* 5: 225–239.

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

Roth, C. and 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. and Colombi, J. M. 2013. “An ontological framework for clarifying flexibilityrelated terminology via literature survey”. *Syst. Engin.* 16(1): 99–110.

Sanchez, R. and Mahoney, J. T. 1996. “Modularity, flexibility, and knowledge management in product and organization design”. *Strategic Management Journal* 17.

Sanchez, R. 1995. “Strategic flexibility in product competition”. *Strat. Mgmt. J.* 16(S1): 135–159.

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.

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

Sasai, K. and 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.

Schilling, M. A. 2002. *Modularity in multiple disciplines*: 203–214.

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

Schoettl, F. and Lindemann, U. 2014. “Design for System Lifecycle Properties A Generic Approach for Modularizing Systems”. *Procedia Computer Science* 28: 682–691.

Shannon, C. E. 1951. “Prediction and entropy of printed English”. *Bell System Technical Journal* 30(1).

Simon, H. A. 1962. “The Architecture of Complexity”. *Proceedings of the American Philosophical Society* 106(6): 467–482.

Simpson, T. W. 2004. “Product Platform Design and Customization: Status and Promise”. *Artif. Intell. Eng. Des. Anal. Manuf.* 18(1): 3–20.

Simpson, T. W., Seepersad, C. C. and Mistree, F. 2001. “Balancing Commonality and Performance within the Concurrent Design of Multiple Products in a Product Family”. *Concurrent Engineering* 9(3): 177– 190.

Skyttner, L. 2005. *General Systems Theory: Problems, Perspectives, Practice*. World Scientific Press.

Smedt, K., Horacek, H. and Zock, M. 1996. “Architectures for natural language generation: Problems and perspectives”. In *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, Lecture Notes in Computer Science 1036: 17–46. Edited by G. Adorni and M. Zock. Springer Berlin Heidelberg.

Sole, R. V., Ferrer-Cancho, R., Montoya, J. M. and Valverde, S. 2002. “Selection, tinkering, and emergence in complex networks”. *Complexity* 8(1): 20–33.

Sosa, M. E. and Rowles, C. M. 2007. “Network approach to define modularity of components in complex products”. In *ASME Journal of Mechanical Design*: 1118–1129.

Sosale, S., Hashemian, M. and Gu, P. 1997. “Product modularization for reuse and recycling”. *Design Division* publication 94: 195-206. American Society of Mechanical Engineers.

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

IEEE Standards. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 2000.

Sternberg, S. 2011. "Modular processes in mind and brain". *Cognitive neuropsychology* 28(3-4): 156–208.

Stone, R. B., Wood, K. L. and 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.

Taysom, E. and Crilly, N. 2014. "Diagrammatic Representation of System Lifecycle Properties". In *4th International Engineering Systems Symposium* (CESUN 2014).

Theraulaz, G. and Bonabeau, E. "A brief history of stigmergy". *Artificial life* 5(2): 97–116, 1999.

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

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

Tseng, H., Chang, C. and Li, J. 2008. "Modular design to support green life-cycle engineering". *Expert Systems with Applications* 34(4): 2524–2537.

Tseng, M. M. and Jiao, J. 1998. "Computer-aided requirement management for product definition: a method and implementation". *Concurrent Engineering Research and Applications* 6: 145-160.

Ulieru, M. and Doursat, R. 2011. "Emergent Engineering; a Radical Paradigm Shift". *Int. J. Auton. Adapt. Commun. Syst.* 4(1): 39–60, December.

Ulitsky, I. and Shamir, R. 2007. “Identification of functional modules using network topology and high-throughput data”. *BMC systems biology* 1(1): 8+.

Urich, K. and Eppinger, S. D. 1995. *Product Design and Development*. McGraw-Hill.

Ulrich, K. and Tung, K. 1991. “Fundamentals of Product Modularity”. In *ASME Winter Annual Meeting Symposium on Issues in Design Manufacture/Integration* 39: 73–79.

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

Ulrich, K. and Eppinger, S. D. 2003. *Product Design and Development*. McGraw-Hill Higher Education.

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

van Beek, T. J., Erden, M. S., and Tomiyama, T. 2010. “Modular design of mechatronic systems with function modelling”. *Mechatronics* 20(8): 850–863.

Van Wie, M. J., Rajan, P., Campbell, M. I., Stone, R. and Wood, K. L. 2003. “Representing Product Architecture”. In *ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference* 3.

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

Vermaas, P. E. 2012. “On the formal impossibility of analysing subfunctions as parts of functions in design methodology”. *Research in Engineering Design* 24: 18-32.

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.

Wagner, G. P. 1996. “Homologues, natural kinds and the evolution of modularity”. *American Zoologist* 36(1).

Wagner, G. P., Pavlicev, M. and Cheverud, J. M. 2007. “The road to modularity”. *Nature reviews: Genetics* 8(12): 921–931.

Walz, G. A. 1980. “Design tactics for optimal modularity”. *Proceedings of Autotestcon*. IEEE.

Weng, G., Bhalla, U. S. and Iyengar, R. 1999. “Complexity in biological signalling systems”. *Science* 284: 92-96.

Werner, B. T. 1999. “Complexity in natural landform patterns”. *Science* 284: 102-104.

Whitesides, G. M. and Ismagilov, R. F. 1999. “Complexity in Chemistry”. *Science* 284: 89-92.

Weng, L., Menczer, F. and Ahn, Y. Y. 2013. “Virality prediction and community structure in social networks”. *Scientific Reports* 3.

West-Eberhard, M. J. 2003. *Developmental Plasticity and Evolution*. OUP USA.

Weyns, D. and Holvoet, T. A. 2002. “A Colored Petri Net for a Multi-Agent Application”. In *Second Workshop on Modeling of Objects, Components, and Agents*: 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+.

Whitacre, J. and Bender, A. 2010. “Degeneracy: A design principle for achieving robustness and evolvability”. *Journal of Theoretical Biology* 263(1):143–153.

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

Wilhelm, B. 1997. "Platform and Modular Concepts at Volkswagen Their Effects on the Assembly Process". In *Transforming Automobile Assembly*: 146– 156. Edited by K. Shimokawa, U. Jurgens, and T. Fujimoto. Springer Berlin Heidelberg.

Winsor, J. and MacCallum, K. 1994. "A Review of Functionality Modelling in Design". *The Knowledge Engineering Review* 9(02): 163-199.

Yamamoto, L., Schreckling, D. and Meyer, T. 2007. "Self-replicating and self-modifying programs in fraglets". In *BioInspired Models of Network, Information and Computing Systems*, 2nd Bionetics: 159–167. IEEE.

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

Zamirowski, E. and Otto, K. 1999. "Identifying product portfolio architecture modularity using function and variety heuristics". In *ASME Design Engineering Technical Conference*.

Zemach, E. 1992. "Types: Essays in Metaphysics".

Zimmer, R., Daskalopulu, A. and Hunter, A. 1999. "Verifying Inter-Organisational Trade Procedures by Model Checking Synchronised Petri Nets". *South African Computer Journal* 24.

[1] SysML standards are open source and are periodically revised, see: http://www.sysml.org/

[2] 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: http://www.compass-research.eu/index.html

[3] Within a given context or domain, a lack of explicit precision in how terms are used tends to matter less because all those concerned tend to share similar assumptions (e.g. designers belonging to the same organisation designing the same or similar product, scientists in the same team studying the same system).

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

[5] For a review of systems definitions see (Skyttner, 2005: pp. 57-58; Veeke, Ottjes, & Lodewijks, 2008: p. 9).

[6] There is also a tendency to see terms such as 'system', 'system of systems', 'complex system' or 'emergence' as describing the intrinsic nature of the entity. However, whether or not an entity is deemed to be a system, a system of systems, a complex system or to be seen as exhibiting emergence depends on our description or model of the system (although of course, it can not be ruled out that there are some entities for whom such characterisations would not be applicable).

[7] By 'entity in the world' we mean a concrete realisation, but this need not be physical. For example, a process in execution or a procedure that is adopted would count as entities in the world within the context of certain system characterisations.

[8] 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 supertype-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).

[9] For a more detailed discussion of scope and resolution, see Ryan, 2007.

[10] An example of overlapping classification would be an action, e.g. kicking a ball, which might on the one hand be characterised as the subtype of an action type, e.g. initiating a ball game, but on the other hand be seen as defining this action type, e.g. initiating a ball game is a subtype of kicking a ball. An example of overlapping composition would be a relationship between human beings, e.g. a friendship, which might on the one hand be characterised as part of the individuals involved but on the other hand be seen as consisting of these individuals (and therefore the 'whole' in which the individuals participate). Although these examples are simple, it should already be obvious that characterisations integrating them would be complex.

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

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

[13] The simplest possible architecture is a single component type.

[14] 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 and 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 components. Our definition is therefore general enough to accommodate architectures defined at high levels of abstraction with respect to their applications, such as reference architectures (Holtta and 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 languages defined by these grammars).

On the other hand, 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 and Tseng, 1999a; Jiao et al., 2007) can be seen as examples 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 and 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).

[15] We are aware of other definitions of 'architecture' that *do* include references to function, such as those found in (Ulrich, 1995; Baldwin and Clark, 2000; Mikkola, 2000, Mikkola and Gassman 2003; Chen and 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 and Otto, 1999; Dahmus et al., 2001).

[16] This does not preclude the functions themselves making reference to these other aspects. 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 (which 'subfunctions' it is composed of or decomposes into) and classificatory characterisations (which functions it is seen to be a variant of and which variants it itself has). See also (Pahl and Beitz, 1984; 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 (Vermaas, 2012) (e.g. 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)) but since our framework does not define the deep semantics of such relationships, we consider this debate outside the scope of this article. 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).

[17] 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 and Rhodes, 2007; Ross et. al., 2008) or 'regime shifts' in ecosystems (Gunderson, 2001)) would count as a behaviour since the dynamic aspect is included in the characterisation.

[18] 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 to another; (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 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 userproduct 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).

[19] This might 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.

[20] 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.

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

[22] Component types with the same interface compatibilities are also referred to as ‘module variants’, ‘module types’ or even simply ‘modules’ (Galsworth, 1994).

[23] Some firms may even have product ‘portfolios’, where different families might share either or both architectures and component types (Zamirowski and Otto, 1999; Dahmus et al., 2001). In (Mikkola, 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.

[24] This has also been discussed in relation to a system’s susceptibility to risk and the application of modularity to mitigate risk, e.g. (Goswami and Tiwari, 2014)

[25] A concrete example of this scenario would be when, even though several different types of molecules could potentially catalyse a particular chemical reaction, at a given point in time, only molecules of one particular type are free from participating in other reactions to catalyse the reaction. Conversely, in another state, there may be many molecules of many different types free to participate in the reaction. Some of these molecules will then be ‘redundant’ with respect to the function. Thus, whether a particular element is essential or redundant with respect to a particular function is dependent on the states, behaviours and function realisations of other elements in the system (which are its environment), i.e. multi-structural function realisation implies context-dependency in function when scope is reduced (this might be context-dependent multi-functionality, but it can also include cases where the structure only realises one function in the system, but where realisation of this single function is context-dependent).

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

[27] Open systems characterisations are those where the system itself can change structure, i.e. not do dependencies exist between elements, but *which* elements depend on each other can change. In (Giavitto and Michel. al., 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.

[28] 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, Rieu, & Tollenaere, 2003).

[29] Flexibility can also mean fragility if the majority of functions to which the architectures map are ones with negative consequences.

[30] In the product design context, a 'design for variety' (DFV) framework (Martin and 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).

[31] 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).

[32] 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 rapidchanging 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. E.g., we do not make the distinction between the 'spatial' and temporal dimension of the environment made in (Heydari and Dalili, 2014).

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

[34] 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).

[35] Many sophisticated techniques exist for specifying state transition rules, such as petri nets (Zimmer et al., 1999; Weyns and Holvoet, 2002) or state charts (Kimiaghalam et al., 2002; Stamatopoulou et al., 2007), but a detailed review is outside the scope of this article.

[36] This constrained set might be mapped to the system type itself, i.e. a system type can be defined by a set of possible behaviours that we see as being essential for instances of the type to be 'viable'. For example, to be a living human being *is* the realisation of certain biological functions which are mapped to architectures with particular commonalities; at a given point in time, some of these commonalities can be seen as 'guiding' the system toward realising other commonalities. Similarly, we might say that a system instance is a product because the mutually constraining properties and behaviours it realises throughout its lifetime map to the set of functions associated with the product's functional requirements.

[37] E.g. Ford and Lerner, 1992; West-Eberhard, 2003; Yam, 2004; Schlosser and Wagner, 2004; Powell et al., 2005; Hornberg et al., 2006; Roth and Cointet, 2010.

[38] The environment might also determine the wider implications of the 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 enviromental 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, a different set of environments (which might be characterised as different environmental states of a single environment), flexibility might conversely make the system resilient because the functional variety afforded allows it to realise viable possibilities in.

[41] Recently, there have been significant efforts in both systems engineering Ingram et al. [2014] and synthetic biology Agapakis and Silver [2009], Agapakis [2014] to find appropriate representations of such ‘patterns’ so that they might be better shared within the domain.