Engineering Self-Designing Enterprises as Complex Systems Using Extended Axiomatic Design Theory

Hadi Kandjani, Peter Bernus
School of ICT, Griffith University, Brisbane, Australia
(Tel: +61-737355002) e-mail: {H.Kandjani, P.Bernus}@griffith.edu.au

Abstract: The paper proposes an approach for designing complex enterprises as systems of systems that on each level of hierarchy have a (recursive) self-designing property. A definition of self-evolving / self-designing systems is given using the notion of life cycle relationships as defined in ISO 15704:2000 and is presented in a form for practical use in enterprise engineering. The approach is based on an extension of Nam Suh's axiomatic design, by introducing a third design axiom, the 'axiom of recursion'. It is argued that (under certain circumstances) by distributing the design authority between systems on each level, it may be possible to limit the complexity of the system that any design authority needs to see.

Keywords: Complex Systems, Enterprise Modelling, Socio-technical systems, Systems engineering

1. INTRODUCTION

1.1 Uncontrollability of Human-Engineered Systems

One way to look at the history of homo-sapiens is to consider it as the history of inventing, building, using, continuously improving and reinventing tools to support human endeavour. This history starts with the creation of simple tools, such as weapons for hunting and warfare, through to today’s complex engineering objects and production, transport, etc. systems.

With the invention and application of computers humankind has created the means to design and build systems of unprecedented complexity, solving problems and providing services that were impossible before. Early in the use of computers it was realised that the creation of ever more complex software systems has limits (Brooks 1982). This is a serious problem, because humankind came to rely on systems of which the complexity makes them harder and harder to invent, specify, design, build, operate and control.

Various disciplines have experienced the problem of having to design and construct more and more complex systems and built tools to handle ever more complex models. Our observation is that while improved design methodologies, modelling languages and analysis tools can decrease the designer’s problem, they only extend the complexity barrier that a designer (or group of designers) can deal with – they do not remove the barrier. This is because the desired functionality of the system may be intrinsically complex (i.e. the complexity can only be avoided by giving up on some desired system characteristics). Therefore any designer who needs to model the complete system faces a problem.

Our hypothesis is that perhaps the system (or system of systems) and the designer should not be separated: systems should design themselves, out of component systems that have the same self-designing property. This means that while the system of systems may have an intrinsically complex nature (by some significant complexity measure). This complexity would only have to be seen by an omniscient external observer, but not by any involved design authority.

1.2 Measures of Complexity

Gershenson (2007) points out, that the complexity of the representation of a system depends on the observer (stakeholder). The explanation of this fact is as follows.

When modelling any system, different elements (levels of detail, properties, dimensions, states, and probabilities of events etc.) are considered relevant by different stakeholders of that system (such as user, designer, manager, …). An omniscient observer ‘O’ would have to be aware of all such elements (etc), as may be represented by O’s model of the system (M_O). However, individual stakeholders (S_i) only need to see appropriate views of this model. Each such view is a model M_{S_i} of the system seen by the respective stakeholder S_i. Therefore, while it may be possible to define an intrinsic complexity measure of a system (based on M_O) this may not have to be seen by any real stakeholder at all.

Contemporary researchers in architecture, biology, computer science, dynamic systems, engineering, finance, game theory, etc., have defined different measures of complexity for each field. Lloyd (2001) observes that these researchers were asking the same or similar questions and that their answers have much in common. Following Lloyd: there are three questions that are posed when attempting to quantify the complexity of an entity (which Lloyd calls a ‘thing’):
a) How hard is it to describe the entity?

b) How hard is it to create the entity?

c) What is its degree of organization of the entity?

(Lloyd 2001) lists the following measures for each:

a. The difficulty involved in completely describing the entity (typically quantified in bits): Information; Entropy; Algorithmic complexity or algorithmic information content; Minimum description length; Fisher information; Rényi entropy; Code length, Chernoff information; Dimension; Fractal dimension; Lempel-Ziv complexity.

b. The difficulty involved in constructing or duplicating the entity; Time/space Computational complexity; Information-based complexity; Logical depth; Thermodynamic depth...

c. The degree of what Lloyd calls 'organization of the entity' (Lloyd does not elaborate on the dimensions used to quantify these), whereupon this measure is subdivided into:

i) The difficulty involved in describing the organizational structure: Effective complexity; Metric entropy; Fractal dimension; Excess entropy; Stochastic complexity; Sophistication; Effective measure complexity...

ii) The amount of information shared between the parts of a system as the result of its organizational structure: Mutual information; Channel capacity; Correlation; ...

1.3 Complexity as an Important Cause of the Problem

What groups (a) and (c) of complexity measures above have in common is: they measure the difficulty an observer encounters when describing the entity as a system (for analysing, designing or controlling it). Gershenson (2007) introduces a relative measure of complexity (see Def 1).

Def 1. “The complexity of a system $C_{sys}$ scales with the number of its elements $#E$, the number of interactions $#I$ between them, the complexities of the elements $C_{e}$, and the complexities of the interactions $C_{i}$” (Gershenson 2002).

(Gershenson 2004) shows that in Random Boolean Networks (Kauffman 1993) the chance of chaotic behaviour increases with the number of nodes (#E) and the number of interactions (determined by the number of connections (#I)) between nodes: with the increase in the number of interactions between nodes slightly different starting states tend to produce increasingly different outcomes.

Although this helps us show how system complexity contributes to the difficulty of predicting system behaviour (Melvin 2003) argues that we need large and complex systems to be able to satisfy all functional requirements of a system. However, it is precisely the complexity of a system that makes it difficult to predict that the system will always do what is required! Indeed, Axiomatic Design Theory (Suh 2001) defines a 'complex' system as one that can not be predicted to always satisfy its functional requirements.

Gershenson’s and Suh’s complexity definitions are relative notions, because depending on the concerns of the observer the same entity may be viewed as having different number of elements / interactions (and the complexity of elements and their interactions may in turn be viewed differently by each observer). We conclude that ‘being complex’ is not an inherent class of properties of an entity, but it is a property seen by an observer considering the entity as a system.

Smarr (1985) explains that the concept of complexity is multidimensional and multi-disciplinary and there is no single way to define and measure it. A mathematician may define it by the number of degrees of freedom in computational operations, while a physicist may define it by the number and frequency of interactions in a system.

Abraham (2002) analysed the twentieth century history of complexity theory and describe its three roots, their interactions and bifurcations as a complex dynamic system itself: these roots are cybernetics, general systems theory (theoretical biology), and system dynamics.

Castellani (2009) also identifies three roots of complexity science: systems theory, cybernetics and artificial intelligence, and defines derivations of complexity science from these roots, giving a historical perspective of the area.

Complex Adaptive Systems Theory (Holland 1992; Gell-Mann 1994) looks at the problem of complex system as a system that is created from the interactions of systems – possibly by itself – and which interaction produces emergent properties (without there being a designer at all). Still the problem exists for the observer who encounters such a system without claiming to have ‘designed’ it: how is it possible to control the system (at least to some desired extent)?

1.4 Disciplines that Studied how to Tackle the Complexity Problem

Software Engineering has been trying to devise methods and to reduce apparent complexity. Efforts include the use of design principles and patterns, such as Encapsulation in Object Oriented software development (Snyder 1986), Object Oriented Analysis and Design (OOAD) (Meyer 1988), Component Oriented Development (Nierstrasz, Gibbs et al. 1992), Service Oriented Architecture (Krafzig, Banke et al. 2004; Erl 2005) and Multi Agent Systems (Wooldridge and Jennings, 1995), Software Process Improvement concentrates on the system that develops software (Osterweil, 1987).

Traditional Systems Engineering (SE) has been successful at satisfying complex ‘design, develop, and deploy’ requirements. However, Bar-Yam (2003) argues that there is a transition from procedure-driven to capability-based engineering processes and systems engineering must rely on the self-organising ability of engineering – something traditional system engineering was never proposed to handle.

Systems engineering (SE) grew out of the need to tackle the complexity of designing large scale systems and developed various approaches and methods (‘multistage analysis’, ‘evolutionary engineering’ (Bar-Yam, 2003)). SE attempts to overcome the limitations of ‘decomposition based’ engineering and focuses on planned methods to analyse and
design a system of systems rather than a way to remove the complexity of the designed system(s) (Bar-Yam & Kuras, 2003). Bar-Yam (2003) thinks: if the task is intrinsically complex for usual (systems) engineering processes, then it is better to use an evolutionary process to create an environment within which continuous innovation can be carried out.

Note, that Axiomatic Design (AD) theory (Section 1.3) developed a theory of complex systems (with its definition of ‘complex’), explaining reasons of emerging complexity in design. AD is also a formal design theory, defining principles that designers of systems need to follow to produce systems with minimal complexity (in Suh’s sense).

Enterprise architecture (or ‘enterprise engineering’) addresses the same problem, because enterprises are unquestioningly complex systems, but while they can be partly characterised as ‘designed systems’ they are also ‘complex adaptive’ in nature because the system being changed/designs is usually a ‘living system’ (a property also studied in General Systems theory (Bertalanffy, 1968)). Enterprise architecture frameworks e.g., GERAM (IFIP-IFAC Task Force, 1999; ISO15704:2000), aim at advising on a complete collection of tools, methods and models to be used by enterprise engineering/change efforts. GERAM is a “toolkit of concepts for designing and maintaining enterprises for their entire life history” (ISO, 1999a; Bernus, Nemes et al. 2003) therefore we expect that it may be used to systematise various contributions of the field that address the creation and sustenance through life of the enterprise as a complex system.

Although other frameworks can be applied to the context of the current paper, we chose GERAM because of its distinct features: a) complete life-cycle coverage of enterprise engineering, b) considering equally important the human and technical views in engineering a system, c) its ability to cover hybrid systems engineering like engineering socio-technological systems, and d) ability to demonstrate relationships between life cycle and life history of entities of an enterprise as a complex system (an important distinction not provided by any other systems engineering frameworks).

1.5 A Pragmatic Measure of Complexity

As we try to solve the difficulty of having to use complex design descriptions, we turn to Axiomatic Design Theory’s complexity measures. Note that AD proposes techniques for reducing complexity in multiple engineering domains (incl. software development (Suh and Do, 2000)). Some complexity definitions are easy to understand but hard to measure: e.g., Chaitin (1966) and Kolmogorov (1965) proposed a definition of complexity of a software calculated as the length of its most compact description. However, these measures can not be explicitly computed (Suh 1999).

What we propose as the measure of complexity is a proxy of this measure. For a system to always satisfy its functional requirements it must have ‘enough knowledge’ of the state of its surroundings (e.g. for software systems this includes the operating system, storage, processor, communication network…), the states of the control information received from other systems, and the possible states of the system’s inputs (in each case two states are deemed different if the system must respond differently). To function correctly, the system has to know the relevant states of its surroundings. By encoding these relevant states the minimum length of system description grows with the number of these states: so we can use the number of these relevant states as a complexity measure (and call it ‘information content’ (IC)).

The chance of success of a design (Suh, 1999) is low if the designer must use too much information to satisfy the FRs. This is equivalent to having to consider too many states in which a system must operate. Therefore our Information Content (IC) as a complexity measure is a proxy of what Suh calls ‘information content’ ($I_{CSuh} =$ the negative logarithm of the probability that the system always satisfies its FRs). The advantage of IC is that it is a measurable quantity.

2. ENTERPRISE ENGINEERING AND COMPLEXITY

2.1. Complexity Addressed By Axiomatic Design

We interpret Lloyd’s complexity measure classification as:

- a. Complexity of the system’s function (difficulty to describe the function, behaviour, and states of the system).
- b. Complexity of the process creating the system (difficulty to describe and execute the process creating the system).
- c. Complexity of architecture (difficulty to describe the relationship between physical and functional structure, e.g. due to number of relationships / dynamics of relationships).

First consider a) and c). Both measures talk about a ‘target system’ (in our case an ‘enterprise’). As enterprise engineering is a special case of systems engineering, we can try to apply principles of AD, given that AD claims to codify (in a discipline-independent way) what a ‘best design’ is.

Suh’s two axioms of design are as follows:

**Axiom I: Independence Axiom.** ‘The independence of FRs must always be maintained.’

Explanation: a FR, is independent of other FRs if there exists a solution (a list of ‘design parameters’ [DP]) such that if changing one FR, only one DP needs to be changed.

According to Suh, design is a process to find DPs such that $[FR] = [[A]] * [DP]$, where $[FR]$ is the vector of FRs, $[DP]$ is the vector of DPs and $[[A]]$ is mapping DPs to FRs.

- If $[[A]]$ is a diagonal matrix the design is called uncoupled (full independence of FRs is achieved).
- If $[[A]]$ is triangular then the design is decoupled (the implementation process of DPs is ‘serialisable’).
- If $[[A]]$ can not be rearranged to be triangular then the design is coupled (the implementation is not ‘serialisable’).

Therefore, if an uncoupled design solution is not possible then the number of dependencies must be minimised, through trying to achieve decoupled solutions. If decoupled solutions do not exist then the number of couplings must be minimised.
We see Axiom I as the intent to minimise the complexity of the system’s architecture – complexity type ‘c’ above (examples of this are found in many engineering disciplines).

**Axiom II: Information Axiom** ‘Out of the designs that satisfy Axiom I that design is best that has the minimal information content.’

Explanation: while Suh defined information content as the negative logarithm of the ‘probability of success’, in section 1.5 we proposed an alternate measure (a proxy) for this property, due to pragmatic advantages of this measure. We see Axiom II as the intent to minimise the difficulty to describe the system’s function – complexity type ‘a’ above.

According to these two ‘axioms of design’ an enterprise as a system must be specified and designed for its FRs to be satisfied by design parameters as independent as possible, where the combined information content is minimal.

Observe that complexity type ‘b’ is not automatically addressed by AD in enterprise engineering. How to avoid this third type of complexity is described in Section 2.4.

An entity’s life history (cf GERAM) is the representation in time of tasks carried out on the particular entity during its life. E.g., several concurrent design and implementation processes may be executed in one enterprise engineering process, and these work in parallel with the enterprise’s operation. The same can be applied to the life history of the system that changes the target system (the ‘change system’): dependencies exist between the current states of a change system, its next states, and the target system’s state.

According to the study of Random Boolean Networks (Kauffman 1993) the more dependency links exist between consequent states, the higher the probability of chaos in the life of a system. E.g., if the architect produces a draft architectural design, and the designer of a subsystem can work without having to depend on design specifications of other subsystems, progression is predictable. However, if a complete design specification of all subsystems is needed for designers to progress, change in one subsystem’s specification propagates (causing unpredictability).

The example indicates that there are inseparable dependencies between states of different life-cycle activities throughout the life-history of a system. We suppose that if there are too many dependencies and these dependencies are not controlled, then possibly the evolution of the system during its life history becomes chaotic. Williams and Li (1994) attempted to use AD in enterprise engineering, and demonstrated its use to the development of a master planning methodology. In a way their work is an early attempt to address complexity type ‘b’ of Section 2.1, i.e. to minimise the complexity of the system that creates the system.

While Li and Williams demonstrated the application of AD to the development of a methodology of master planning, typically (in addition to the instantiation of such a methodology in form of a master planning project) there exist *multiple other change processes*, performed at the same time. E.g. a long term master planning effort may be parallel with multiple shorter term change processes. These processes all change the entity (system of interest) but on various horizons.

What we propose below is to define methodology-agnostic principles of change to be satisfied when co-ordinating multiple change projects / processes.

The set of change projects operating on a system could use different methodologies or processes, but their target is the same entity, so there will be *dependencies among these projects*. The change projects themselves form a ‘change system’ (a set of processes which could be considered as a complex system by itself). As a consequence, the complexity of the process that creates the system (complexity type ‘b’ of Section 2.1) is the same as the complexity of type ‘a’ and ‘c’ of this change system. We therefore propose that the two AD axioms must also be applied to the change system itself.

This is expressed as the recursion axiom, meaning that change projects (as a system of systems) not only have to use axiomatic design but they must be ‘axiomatically designed’.

**Axiom III. Recursion Axiom:** The system that designs a system must satisfy Axioms I & II of design.

Explanation: A system that satisfies Axioms I and II does not necessarily satisfy Axiom III and while at a given moment in time in its life history a system may be considered moderately complex, the same system may be very hard to create or change. E.g., denote three consecutive stages of a system S as S1, S2 and S3. In S1 system S is operating and has a design that either intentionally (or by pure chance) satisfies Axioms I & II. S2 is the stage of change, where S2 is the original system extended by a change project P. The task of P is to create S3. When S1 creates P it can mandate that P uses Axioms I & II to design S3. However, whether P (and thereby S3) satisfy Axioms I and II remains undetermined.

Thus, P can be more complex than necessary and even if its mandate is to design S3, the likelihood of success of this endeavour may be less than desired, i.e. P does not satisfy Axioms I & II (even if it applies them to design S3). Axiom III states that S2 not only has to mandate that P use Axioms I & II, but S2 has to design P using Axioms I & II (to improve the probability of successful evolution).

As demonstrated, systems at one stage of life may satisfy Axioms I & II but may lose this design quality and through reducing the likelihood of success in the change process this quality may even be lost permanently. To prevent such state of affairs we had to introduce Axiom III. Accordingly Axiom III is independent of Axioms I & II. Pragmatically: a large and complex system is created by complex systems to the design of which axiomatic design needs to be applied (the change system (of systems) is called the set of ‘supporting systems’ in ISO 15288:2000).

Consequently, among those design processes that apply the first and second axioms to design a system, that design process is best which itself satisfies the 1st and 2nd axioms.

11946
3. SELF-EVOLUTION AND COMPLEXITY

A way to minimise the amount of information that a designer needs to have about the states of a system $S$ (and its subsystems $S_i$) is to minimise the amount of information the designer has to have about the set of states $T_i$ of each subsystem (where "$n$" is the number of subsystems). There are two ways to minimise the observed information content:

(i) The designer of a system elaborates the dependencies of subsystems for each system state in $\Pi_{i=1,...,n}T_i$ and defines the desired system behaviour for each such state. Note that in large systems multiple parallel change processes of subsystems may create very large values of $T_i$. The designer may instead, stipulate a set of relevant states $t_i$ of each subsystem (where a $|t_i| << |T_i|$) such that dependencies are artificially restricted to avoid chaotic life history. E.g., the designer of a system may stipulate $t_1$ and request a ‘controlled release of change’ from the subsystem’s designer. This technique reduces the chance of undesired system behaviour but does not alleviate the designer’s problem, because the designer still has to consider all of $T_i$.

(ii) It is known from practice that for the same pragmatic purpose a designer situated outside of the system needs a more detailed model of the system than a designer that is part of the system itself. This is because a designer who is part of the system will have developed tacit knowledge and will have already have the knowledge of how to collapse $T_i$ into $t_i$ – without actually having to enumerate $T_i$ at all: the designer who is part of the system knows what the relevant distinctions are regarding the system’s ability to satisfy its requirements. Thus the apparent information content of the system from this ‘embedded designer’ point of view is less than the apparent information content.

Our hypothesis is that perhaps the system (or system of systems) and the designer (group of designers or design authorities) should not be separated and systems and subsystems should design themselves, out of component systems with the same self-designing property.

Def 2: Self-evolving System. A self-evolving system is a system which can perform its own life-cycle activities (by using its own resources or acquiring them from others).

To minimise apparent complexity of a self-evolving system (as appears to any one designer) axiomatic design principles may be used. All of Axioms I, II and III must be satisfied (as the system designs its own change system). Self-evolution can be illustrated by life cycle relationships (see Fig.1).

By satisfying all three axioms the system must attempt to make its life cycle activities as independent, controlled and uncoupled as possible so that the designer can predict the next relevant states of the life history and avoid a chaotic change, i.e. change of those states of which the size and direction must be predictable. (This still allows behaviour that an external observer may characterise as chaotic.)

According to (i) and (ii) self-design results in minimised information content (IC) and thus creates a ‘best design’, from the point of view of likelihood of success. Applying this logic to the change system we derive the conjecture:

Self-evolution Conjecture: Among those change systems which satisfy and apply the first two design axioms, that “change system” is the best which is designed by the stakeholders of the system of interest itself.

In other words that change system (change project/ change program) is most likely to succeed whose model used to create and control it is the simplest. The argument behind this conjecture is that management (the designer of the system’s future) needs to be able to make decisions and plan in light of information about the change environment, and it is the stakeholders themselves (with tacit knowledge) who have the least need for explicit information in order to make predictions about and control the change process.

In other words, models used by these ‘embedded designers’ can be simpler because they know the important distinctions between relevant states. Conversely, embedded designers can recognise two states as identical from the point of view of relevance to the change process, whereupon the same two states of the change environment would be considered different by ‘outsiders’ (external observers) who do not have tacit knowledge of this environment.

As a consequence, the requisite variety of the change system (Ashby, 1957; p202) can be minimised through the change process being designed by embedded designer agents.

4. CONCLUSION

This article reviewed a number of complexity measures that can help characterise how hard it is to design and change a system so that it all the time satisfies all its requirements. By applying Axiomatic Design theory two types of complexity can be avoided, but we have demonstrated that the third type of complexity (defined by Lloyd) is actually the repetition of the other two types of complexity, only targeting the change system not the system of interest. As a consequence, we extended the collection of design axioms with Axiom III – the Recursion Axiom. To reduce the apparent complexity of systems (such as enterprises) we concluded that a way to achieve this if systems design themselves and the designer is part of the system being (re)designed.
We derived, by logical argument, a conjecture which states that the best change system (‘best’ by the likelihood of success) is a change system designed by the system itself. We introduced the notion of self-evolving systems, a category of evolving systems (those deliberately designing themselves).

REFERENCES


Li, H., T. Williams (1994) A formalization and extension of the Purdue enterprise reference architecture and the Purdue methodology. TR 158 Purdue Lab. of Applied Industrial Control. West Lafayette : Purdue University.


