Authors' copy of
Kandjani, H., Tavana, M., Bernus, P., Wen, L. , Mohtarami, A. (2015). Using Extended Axiomatic Design Theory to Reduce Complexities in Global Software Development Projects. *Computers in Idustry*. 67(February 2015):86-96.

Using Extended Axiomatic Design Theory to Reduce Complexities in Global Software Development Projects

Hadi Kandjani[a,b]
[a]Department of Information Technology Management
Shahid Beheshti University, Tehran, Iran

[b]Centre for Enterprise Architecture Research and Management (CEARM)
School of Information and Communication Technology, Griffith University, Brisbane, Australia
E-mail: h.kandjani@griffith.edu.au

Madjid Tavana[c,d*]
[c]Business Systems and Analytics Department
Lindback Distinguished Chair of Information Systems and Decision Sciences
La Salle University, Philadelphia, PA 19141, USA
E-mail: tavana@lasalle.edu
Web: http://tavana.us/

[d]Business Information Systems Department
Faculty of Business Administration and Economics
University of Paderborn, D-33098 Paderborn, Germany

Peter Bernus[e]
[e]Centre for Enterprise Architecture Research and Management (CEARM)
School of ICT, Griffith University, Brisbane, Australia
E-mail: p.bernus@griffith.edu.au

Lian Wen[f]
[f]Institute for Integrated and Intelligent Systems (IIIS)
School of ICT, Griffith University, Brisbane, Australia
E-mail: l.wen@griffith.edu.au

Amir Mohtarami[g]
[g]Department of Information Technology Management,
Faculty of Management and Economics, Tarbiat Modares University, Tehran, Iran
E-mail: A.Mohtarami@modares.ac.ir

Abstract

Global Software Development (GSD) projects could be best understood as intrinsically complex adaptive living systems: they cannot purely be considered as 'designed systems', as deliberate design/control episodes and processes (using 'software engineering' models) are intermixed with emergent change episodes and processes (that may perhaps be explained by models). Therefore to understand GSD projects as complex systems we need to combine the state-of-the-art of GSD research, as addressed in the software engineering discipline, with results of other disciplines that study complexity (e.g., Enterprise Architecture, Complexity and Information Theory, Axiomatic Design theory). In this paper we study the complexity of GSD projects and propose an upper bound estimation of Kolmogorov complexity (KC) to estimate the information content (as a complexity measure) of project plans. We demonstrate using two hypothetical examples how good and bad project plans compare with respect to complexity, and propose the application of extended Axiomatic Design (AD) theory to reduce the complexity of GSD projects in the project planning stage, as well as to keep this complexity as low as possible during the project execution stage.

Keywords: Global software development; Complexity; Extended axiomatic design theory; Kolmogorov complexity.

*Corresponding author at: Business Systems and Analytics Department, Lindback Distinguished Chair of Information Systems and Decision Sciences, La Salle University, Philadelphia, PA 19141, United States. Tel.: +1 215 951 1129; fax: +1 267 295 2854.

1. Introduction

A Global Software Development (GSD) project has to go through complex processes to finish projects within an allocated budget, time schedule, and with customer satisfaction and completely fulfilled functional and non-functional requirements. The concept of GSD implies distributed teams from different organisations and geographical locations who collaborate to design, manage and execute life cycle activities of a joint software development project functioning as a supply chain (Oshri et al., 2007). This structure in itself increases the complexity of distributed GSD projects (Sahay et al., 2003; Šmite and Borzovs, 2008; D'Mello and Eriksen, 2010), where part of this complexity is due to dynamic dependencies among components of the software development products (Cataldo et al., 2006) as well as dependencies among life cycle activities of project planning and software development groups (Abbott et al., 2013). This complexity creates uncertainty and ambiguity due to the high number of elements and also the high amount of dependencies among GSD products, projects or project activities (Marczak and Damian, 2011).

Given the highly distributed nature of GSD projects a completely centralised control is very hard to achieve, and subsequently these projects could be looked at as intrinsically complex adaptive systems: they cannot purely be considered as 'designed systems', as deliberate design/control episodes and processes ('software engineering', using models) are intermixed with emergent change episodes and processes (that may perhaps be explained by models).

There exist various kinds of engineered systems, including software products, which are developed by a global engineering effort. Common to all is a highly complex (or complicated) project design, as many of these projects have been usually designed "without having a theoretical framework for complexity" (Suh, 2005). GSD therefore is becoming more complicated unless fundamental theories and principles, and corresponding methods for reducing complexity are developed (or adopted from the complexity field). An ultimate goal of the complexity field is to replace the "empirical approach" in designing, operating and managing complex systems with a more "scientific approach" (Suh, 2005). Complexity is therefore an important problem facing GSD projects, because uncontrolled complexity can cause undesired design qualities and therefore unsatisfied requirements of GSD projects. The first question that may arise, before going any further, is: "What is Complexity?"

Gershenson (2007) defines the complexity of a system (C_{sys}) as a function of the number of its elements (#*E*), the number of interactions between them (#*I*), the complexities of the elements (C_{ej}), and the complexities of the interactions (C_{ik}) among elements. Axiomatic Design (AD) theory (Suh, 2001) defines a 'complex' system as one that cannot be predicted to always satisfy its functional requirements. Suh (2001) and other authors, such as (Melvin, 2003), define the concept of system complexity through considering 'the probability of satisfying all functional requirements all the time'. Functional requirements are defined in AD as "a minimum set of independent requirements that completely characterise the functional needs of a product (software, organisation, systems, etc.) in the function domain" (Suh, 1990; Suh, 2001).

For software engineers the notion of a software not always satisfying its functional requirements may seem odd, a normal reaction to such state of affairs would be that this is due to the lack of a complete verification. However, in large scale systems verification cannot be complete, especially because one must take into account that the ability to produce the correct output by transforming an input that satisfies the preconditions, depends on other 'assumed inputs', for example that at the time the transformation must take place, the necessary processing power and storage are available. Even if every component of a system was designed to perform perfectly in isolation, they would not necessarily always perform accordingly as part of a system in every possible operational scenario (with a potentially intractable number of possible operational states), implying the need for a design theory that explains, and for methods that can

be used to reduce, the complexity of a system.

Axiomatic Design is a theory that aims to distil into two 'design axioms' the essence of what is a good design, especially from the point of view of eliminating unnecessary complexity. Many readers may already be familiar with AD, but for those who are not, Section 3 gives a brief introduction to the details of the design axioms that are the core of this theory.

Many applications of AD in product design, system design, organisational decision making, and software development have appeared in the literature. AD was first applied in software engineering by Kim et al. (1991) and was first applied in system design concepts by Suh (1997). Do and Park (1996) also introduced new concepts by applying AD specifically to software design. Designing software based on AD creates "uncoupled or decoupled interrelationships and arrangements among 'modules', and is easy to change, modify, and extend" (Suh and Do, 2000).

Harutunian et al. (1996) used the first ('independence') axiom of AD to evaluate design decisions that provide an optimal software development project sequence. Suh and Do (2000) combined the independence axiom of the AD theory and object-oriented programming to design large-scale software development systems. They were able to shorten the lead-time of software, improve reliability, reduce costs, and increase productivity.

Chen et al. (2001) used the independence axiom to build a hierarchical knowledge base system. They constructed a simulation model and combined it with a decision support system to illustrate the effectiveness of the proposed knowledge base system. Huang (2002) extended the AD principles and defined two master domains: design workspace and review workspace. They investigated the relations between the two domains based on the independence axiom. Huang and Jiang (2002) used fuzzy set theory and expressed past experiences and insights as the membership functions of design parameters and evaluation criteria.

Lindkvist and Söderberk (2003) used the independence axiom of AD and robust design to compare and evaluate assembly concept solutions. Chen et al. (2003) used the independence axiom to facilitate both the integration of existing software and the modification of software since changes in one module did not affect other modules. Chen and Feng (2004) used the independence axiom to test a computer-aided design model whether the proposed model satisfied the independence axiom or not. Yi and Park (2005) developed software to analyse and construct the design process according to the independence axiom of the AD theory.

Togay et al. (2008) proposed a component-oriented approach based on the AD theory. In the study, the V-Model proposed by Suh and Do (2000) was extended since the AD process model did not address component-level architecture issues. Kulaka (2010) provides a comprehensive overview of the literature on AD theory and principles.

Suh (2005) divides "the treatment of complexity" into two distinct domains: treating the complexity in the "physical domain" and treating it in the "functional domain." In the first domain most engineers, physicists and mathematicians consider complexity as an "inherent characteristic of physical things, including algorithms, products, processes, and manufacturing systems". The "functional" approach is to treat complexity as a relative concept that evaluates how well we can satisfy "what we want to achieve" with "what is achievable" (Suh, 2005). By considering a GSD project as an artifact it may be possible to apply AD theory to the project, and increase the probability of satisfying all project requirements (i.e. the project always performs what it needs to do).

The remainder of this paper is organised as follows. In Section 2 we introduce a reference model for GSD projects and Extended AD theory and use this theory to address the complexity of GSD planning and development projects in Section 3. After these reviews, in Section 4 we use an upper bound estimation of the complexity of the design matrix (by applying

a complexity measure well known from information theory, Kolmogorov complexity (KC), and use this as a proxy measure of AD theory's Information Content metric). Using this proxy it is possible to measure the complexity of the design of an object, whereupon in this article the objects of interest are the software project planning project and the software product development project itself (i.e. we are *not* talking about the complexity of the software product). In Section 5 we present two hypothetical examples to compare both good and bad GSD planning projects and development projects from the complexity point of view. In Section 6 we discuss the separation of management functions from operations, and in Section 7 we present conclusions and future research directions.

2. A Reference Model for Global Software Development

Prikladnicki et al. (2006) proposed a reference model for GSD based on the results of real GSD case studies. Their proposed reference model includes the organisational and the project dimensions:

Organisational dimension (Planning): Prikladnicki et al. (2006) state that planning is important to properly organise and manage distributed projects. They identified the initial planning as a formal and basic stage to decide if a project can be distributed, how to plan for its development, and how to coordinate and manage different GSD projects that produce globally developed software products. Based on their case studies, they proposed a GSD planning stage as a precursor to the development project's activities that are determined by the planning process. In order to avoid ambiguity we use here the term stage, rather than 'phase': an explanation of the important difference between *life cycle* (phases) and *life history* (stages) is presented in the appendix (based on the Generalised Enterprise Reference Architecture and Methodology (GERAM)).**Project dimension** (Development): This includes Prikladnicki et al.'s (2006) interpretation as, "general coordination of work between collaborators, interfaces among teams, communication, and contacts with clients and conflict solving." This dimension is defined as a set of life cycle activities that deal with the requirements analysis, design, building, integration, testing, and release into operation of the end product.

We interpret these dimensions as two sets of processes in the life history of GSD (a) the set of GSD project planning life cycle activities and (b) the set of GSD product development life cycle activities. In other areas of engineering, it is customary to separate these two sets into two separate projects: a bidding project (for planning) and an EPC (Engineering, Procurement and Construction) project (this latter usually consisting of a set of interrelated sub-projects).

However, there is also an important difference: set '(a)' here has two subsets: the up-front project planning activities, i.e., planning during the bidding stage, and the ongoing project planning activities performed by the project manager of the EPC project during the project execution stage (as part of 'shifting planning', such planning is normally performed due to the need for detail that was not available at the time of initial planning, or due to change of circumstances that require the modification of the original project plan).

3. Complexity addressed by Axiomatic Design (AD) theory

According to Lloyd (2001) there are three questions that are posed when attempting to quantify the complexity of an entity:

(a) How difficult is it to describe the entity?

(b) How difficult is it to create the entity?

(c) What is the degree of organisation of the entity?

Applied to GSD development projects, these measures, as interpreted by Kandjani and Bernus (2011a), can be classified as those that characterise the difficulty of describing (a) the function, behavior, and states of the GSD development project as a system, (c) the architecture

(relationship between physical and functional structure of the GSD development project as a system), and (b) the GSD planning process (which creates or changes GSD development projects). In this case, categories (a) and (c) measure the complexity of the GSD *development* projects. As opposed to this, (b) measures the complexity of a GSD *planning* entity(ies), implemented partly as an initial planning project, and partly as the ongoing project management of GSD development projects – with the view to design, create and maintain GSD development projects.

Kandjani and Bernus (2011b) point out that groups (a) and (c) of complexity measures above have one thing in common: they measure the difficulty that a 'design authority' deals with when describing the GSD development project as a system (for analyzing, designing or controlling it).

Groups (a) and (c) of complexity target the complexity of the project dimension at the GSD development level as defined by Prikladnicki et al.'s (2006) reference model for GSD projects. At the same time, the complexity of category (b) characterises the complexity of the organisational dimension of this reference model, namely the initial planning and ongoing management of GSD projects.

As we try to solve the difficulty of having to use complex design descriptions of GSD projects, we first turn to AD Theory's complexity measures. AD (Suh, 1999) claims to codify in a discipline-independent way what a 'best design' is, and in particular aims at avoiding unnecessary complexity. However, to be able to avoid the complexity of a system that designs another system, AD was extended by introducing the Recursion Axiom stipulating that the system that designs a system must also obey the axioms of AD (Kandjani and Bernus, 2011a).

Note that AD proposes techniques for reducing complexity in any engineering domain, including software development (Suh and Do, 2000). AD is a theory of complex systems, systems that cannot be predicted to consistently satisfy their functional requirements all the time (Suh, 1990)). AD explains the reasons for emerging complexity, and offers a formal design theory as well as two design axioms that system designs must satisfy to minimise complexity. AD measures this complexity by the negative logarithm of the probability that the system always performs its desired function, also called by AD the system's 'information content'. The inverted commas '' are to remind the reader that in this paper we shall use a similar but not identical measure, namely information content as defined and measured by KC of an object (see Section 4).

The idea to use Axiomatic Design in software engineering is not new. Arsenyan and Büyüközkan (2009) presented an AD-based collaboration model in the context of the software industry. They proposed a model structure for collaborative software development, as well as strategies and methodologies that influence the successful execution of collaborative efforts in software development. Their collaborative software development model based on AD, could also be used as a reference model to effectively plan as well as to develop GSD projects.

Carnevalli et al. (2010) proposed the application of AD for minimizing the difficulties of Quality Function Deployment (QFD), which result could also be used in developing software systems.

In this paper we intend to apply Axiomatic Design to minimizing in a measurable way the complexity of GSD development projects as well as the complexity of GSD planning processes.

In order to be able to demonstrate our intended use of Axiomatic Design, we first give a short introduction to AD.

Suh (1990) defined 'design' as a sequence of mapping functions, from user requirements to functional requirements, from functional requirements to design solutions and from design

solution to implementation. He found, after analysing a large number of good and bad designs, that the principles that underly good designs (designs that have a number of desirable characteristics) can be abstracted into two design axioms, i.e. statements that must be true of all good designs. We explain these two axioms below.

> **Axiom I: Independence axiom** (Suh 2007, 1990). '*The independence of Functional Requirements* (FRs) *must always be maintained.*'

A functional requirement FR_i is independent of other functional requirements if there exist 'design parameters' [*DP*] such that when changing one FR_i only one DP_i must change. A design parameter DP_i is a part of the design solution that implements one of more functional requirements, e.g., a subsystem may be a DP. The mapping from FRs to DPs is represented as [FR] = [[A]]*[DP], where [*FR*] is the vector of FRs, [*DP*] is the vector of *DP*s, and [[*A*]] is the matrix mapping *DP*s to FRs, effectively describing which DP is necessary for which FR.

If [[*A*]] is a diagonal matrix then the design is uncoupled (full independence is achieved). If [[*A*]] is triangular then the design is decoupled (the implementation process is 'serialisable'). Otherwise the design is coupled (the implementation process of DPs is not 'serialisable').

> **Axiom II: Information axiom** (Suh, 1990, 2001). '*Out of the designs that satisfy Axiom I that design is best which has the minimal information content.*'

Suh defined information content (IC) as the negative logarithm of the 'probability of success' (success here means that the system always satisfies its FRs).

In this paper we use an upper bound estimation of the KC of the design matrix as a proxy of Suh's Information Content.

Informally, KC is the measure of the amount of information contained in an object. For example if the design description of project P1 is a lot longer than the design description of another project P2, then we would suspect that the design description of P1 has more information in it than the design description of project P2.

One might first think that we could just consider for such comparison a simple measure, such as the number of characters in the file that contains the design description, or some similar measure, but this would be very unsatisfactory, because one description may have been written in a very concise manner, while the other not. Thus what we are really interested in, how long would be the shortest possible descriptions of P1 and P2 respectively, because that would give us an objective measure for comparison.

In Section 4 we shall present a basic mathematical introduction to KC, but we note that in the application context the calculation of this measure may be a built-in function of a project management software, and end users would not need to know the details of how this is calculated.

In order to satisfy Axiom I the designer uses the design matrix and manipulates functional requirements and the structure of the matrix to achieve an uncoupled or decoupled design. When it comes to satisfying Axiom II by minimising the information content of the design, and the proposed information content calculation is also based on this matrix.

Axioms I and II together intend to minimise the complexity of the system's architecture and can be used to design less complex GSD projects. However, consider the complexity of GSD planning processes: the processes that create a GSD project is not automatically addressed by introducing AD. Therefore, Axioms I and II must also be applied to the change system (the processes, programs or projects that create GSD projects). This is called the 'recursion' axiom (below), meaning that change projects (as a system of systems) not only must follow Axioms I and II, but they themselves need to be 'axiomatically designed' (Kandjani and Bernus, 2011a).

Systems (here GSD development projects) at one stage of life may satisfy Axioms I and II but may lose this design quality as they evolve / change, and because of the reduction of the likelihood of success of the change process this quality may even be lost permanently. To prevent such a state of affairs we have to apply Axiom III to the system (GSD planning project) that designs GSD development projects. Accordingly, Axiom III is independent of Axioms I and II. Pragmatically: a GSD development project as a large and complex system is created by GSD planning projects (also as complex systems) for the design of which AD needs to be applied. Consequently, among those design processes (GSD planning projects) that apply the first and second axioms to design a GSD development project, that design process is best which itself satisfies axioms I and II.

Axiom III: Recursion Axiom (Kandjani and Bernus, 2011a). '*The system that designs a system must satisfy the two Axioms of design*.' Note: 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. Consequently, "among those design processes that apply axioms I and II to design a system, that process is best which itself satisfies axioms I and II".

If a GSD project wishes to reduce its own complexity as well as to subsequently maintain reduced complexity through its life, it may wish to adopt AD as a strategy. Therefore it is legitimate to ask whether the GSD project and the GSD companies and collaborators are ready to use such practices to increase the probability of success.

4. Kolmogorov complexity as a proxy for the information content of a design

Generally, the information content is measured by the probability of success. Shin et al. (2004) introduced various methods for the calculation of information content in Mechanical Engineering. Pimentel and Stadzisz (2006) also proposed a method to calculate the information content of software that was designed based on a use case based object-oriented software design approach. These methods of calculation of information content are domain-dependent however what we propose in this paper is a domain-independent method. We use an upper bound estimation of the KC of the design matrix as a proxy of Num Suh's Information Content.

The reason for us to use this particular complexity measure to estimate the information content of a design (in this case the information content of the design description of a software project) is the following. We want to use a complexity measure that is independent of the form, in which the design is expressed, and KC is known to be agnostic of the form of description, i.e., whether the design description is in some kind of textual-, graphical-, or other format. Although there is no algorithm to calculate the *exact* value of KC of a design description, it is very easy to estimate an upper bound of its value.

With KC, we have a complexity measure that can be used in practice: we can take the design description of a software project (the description is some binary string, such as a computer file), and using a simple formula estimate its true information content. For a complete mathematical treatment of the background and arguments regarding why KC is a preferable way of measuring and comparing the information contents of objects we refer the interested reader to (Li and Vitányi, 2008).

Below we present a basic mathematical introduction to KC, but we note that in the application context the calculation of this measure may be a built-in function of a project management software, and end users do not need to know the details of how this is calculated.

A. Definitions: The concept of KC was developed by the Russian mathematician Andrey Kolmogorov (Kolmogorov, 1969). While Kolmogorov is credited with the concept, several other mathematicians appear to have arrived at the same conclusion simultaneously but independently of each other (Nannen, 2010) in the 1960s. KC is one of the key elements in

information theory; it provides a mathematical definition of the information quantity in individual objects, which can be abstracted as binary strings or integers. For about half a century, KC has been applied in various disciplines (Li and Vitányi, 2008).

The KC ($K_U(x)$) of a string x with respect to a universal computer U is defined as the length l of the shortest program p running on U that prints x and halts. It is denoted as:

$$K_U(x) = \min_{p:U(p)=x} l(p) \tag{1}$$

If the computer already has some knowledge about x, for example the length of x as $l(x)$, it may require a shorter program that prints x and halts. In this case, we define the *conditional KC* as:

$$K_U(x \mid l(x)) = \min_{p:U(p,l(x))=x} l(p) \tag{2}$$

Theorem 1. If U is a universal computer, $\forall V$ which are universal computers $\exists c$ is a constant, such that for (i.e. for each binary number x),

$$K_U(x) \le K_V(x) + c \tag{3}$$

The proof can be found in (Cover and Thomas 2006) and will not be repeated here.

Theorem *1* indicates the universality of the Kolmogorov complexity; it shows that the difference of Kolmogorov complexity with respect to different computers is smaller than a constant. If the string x is long, the difference of Kolmogorov complexity caused by different computers becomes trivial. Therefore, we can discuss Kolmogorov complexity $K(x)$ without referring to a particular computer.

We use $\log n$ to mean $\log_2 n$. We also define:

$$\log^* n = \log n + \log\log n + \log\log\log n + ... \tag{4}$$

until the last positive term.

Theorem 2. For an integer n, the Kolmogorov complexity *K(n)* satisfies:

$$K(n) \le \log^* n + c \tag{5}$$

The proof of Theorem 2 can also be found in (Cover and Thomas, 2006). We will explain it in an informal manner here. Generally, we can use a program like "print the integer n" to print n. The program needs the number n, which can be encoded in $\log n$ bits. However, the length of n is unknown, so it requires $\log\log n$ bits to code the length of n and then requires $\log\log\log n$ bits to code the length of the length of n etc.

Theorem 3. For an integer n, if the length of n is known, the conditional KC $K(n|l(n))$ satisfies:

$$K(n \mid l(n)) \le \log n + c \tag{6}$$

The proof of Theorem 3 is similar to the previous theorem.

B. Estimating the Kolmogorov complexity of a transition matrix: For a given transition matrix M, we propose a simple scheme to calculate an upper boundary of its KC.

Let M be a $n \times n$ matrix, where the value of each element in the matrix can only be 1 or 0. The number of ones in M is m. In order to describe (encode) the matrix, we need to record the following information: the number n, the number of ones m, and the position of those ones. Accordingly, we can calculate the KC of M as:

$$K(M) \le K(n) + K(m) + K(\binom{n^2}{m}) \le \log^* n + \log^* m + \log \frac{n^2!}{m!(n^2-m)!} + c \quad (7)$$

If M is a diagonal matrix, because all non-zero elements are ones, it is an identity matrix. It is obvious that in order to record an identity matrix, the only information we require is its size n. Therefore, the KC of an identity matrix can be estimated as:

$$K(I_n) = K(n) \le \log^* n + c \quad (8)$$

In practice the constant c can be ignored because the complexity of two objects would be evaluated on the same universal computer.

5. Hypothetical examples

We introduce two hypothetical examples to demonstrate the application of the three design axioms and how this can be used to reduce the complexity of projects. We shall use KC to compare the complexity of project descriptions. (Note that KC means description complexity, so when we say a project design D is more complex than a project design D', we mean that to fully describe D requires more information than to fully describe D'.)

The first hypothetical example demonstrates an example of a coupled design (a bad design which is more complex than necessary) for a virtual enterprise (GSD Development project *X*). We subsequently apply the first design axiom which results in an uncoupled design.

The second hypothetical example demonstrates the example of a decoupled design for a GSD planning project Pr_X, which creates GSD Development project *X* and the application of the 3rd axiom of design, which is the axiom of recursion, to reduce the complexity of the planning project that designs, creates and implements the GSD Development project *X*.

We use an upper bound estimation of the KC of the design matrix as a proxy of Num Suh's Information Content to demonstrate the difference between the bad and the good designs (by calculating the complexity of the design matrix in both hypothetical examples before and after applying design axioms). We therefore demonstrate in both hypothetical examples how the application of extended AD theory can reduce the complexity of the design description of a GSD Development project *X* as a system of interest, as well as the complexity of the design description of a GSD planning project (Pr_X) as a system, which designs the above system of interest.

Note that in the examples below, we satisfy the FRs by means of Design Parameters (DPs). FR is "what it is we want to achieve" and DP is "how we are going to satisfy the FR". The potential DPs that can satisfy one FR may be many and we have to choose the DP that may be the best.

A. Hypothetical example one (to demonstrate the application of axioms I and II):

'GSD development project X' is a virtual enterprise that produces one software system including three sub-systems: Sub1, Sub2 and Sub3. There are five functional requirements listed below:

- *FR1: Each sub-system needs to have an architectural design.*
- *FR2: Sub1 needs component development and a database module.*
- *FR3: Sub2 needs component development and a GUI module.*
- *FR4: Sub3 needs component development, a database module and a GUI module.*
- *FR5: Each sub-system needs to have a complete unit testing and integration testing.*

Let the original DPs to implement these functions be as follows:

- *DP1: company A provides architectural design services.*
- *DP2: company I provides component development services.*
- *DP3: company J provides database modules.*
- *DP4: company K provides GUI modules.*
- *DP5:company L provides the service of unit- and system integration testing.*

As one can see, the FRs constitute the *tasks* of the project, whereupon in the example, the 'design parameters' (DPs) are the *companies* that can be allocated to these tasks. In this example we need only one company (A) to satisfy FR1, but we need three companies (*I*, *J* and *K*) to satisfy FR4. (In general, when designing a project, we determine who will perform which project task.)

Based on the above, the FR to DP mapping matrix for the GSD Development project *X* is:

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \\ FR_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \\ DP_5 \end{bmatrix} \tag{9}$$

It is clear that this design transition matrix is coupled.

According to the first axiom of design, we must try to maintain the independence of the functional requirements all the time. Therefore to apply AD principles, we introduce a GSD broker company *B* which provides the generic service of 'software implementation'. Then we refine the structure of GSD Development project *X* to GSD Development project X with the functional requirements and DPs as follows:

- *FR_1: Each sub-system needs to have a architectural design.*
- *FR_2: Each sub-system needs to be implemented ('implementation' as a function stands for the generalisation of the component development, database development and GUI development functions).*
- *FR_3: All sub-systems need to be unit- and integration tested.*
- *DP_1: company A provides service of architecture design.*
- *DP_2:company B provides service of software implementation.*
- *DP_3: company C provides component- and integration testing.*

The FR-DP transition matrix for the GSD Development project *X* is now a diagonal matrix:

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{bmatrix} \tag{10}$$

At the same time, the FRs and DPs for the GSD broker company *B* are:

- *FR_1: Some sub-systems need component development.*
- *FR_2: Some sub-systems need a database module.*
- *FR_3: Some sub-systems need a GUI module.*

- *DP_1: company I provides component development.*
- *DP_2: company J provides database modules.*
- *DP_3: company K provides GUI modules.*

The design transition matrix for broker *B* is a diagonal 3×3 matrix as well.

The technique we used here to modify the coupled design to an uncoupled design, while preserving the functional requirements, was to decompose the original functional requirements into independent functional requirements and to create an intermediary DP (broker *B*).

It is noteworthy that normally a function can be decomposed in many different ways: essentially to plan an implementation of a function, we design a process that consists of a series of coordinated invocations of some more elementary functions (if the process is procedural then this process is called an algorithm). Clearly, there are many alternative processes that can implement the same function, which gives the project manager the ability to consider which alternative is best from the point of view of complexity reduction.

Let us now calculate the KC of each transition matrix of the GSD Development case study. In the original design, the transition matrix *M* is:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

For this transition matrix, we have n=5, m=9, so based on inequality (7), we have:

$$K(M) \le \log^* n + \log^* m + \log^* d + \log \frac{n^2!}{m!(n^2 - m)!} + m \times \log d \approx 29.9 bits \tag{12}$$

In the interest of generality, we slightly extended inequality (7), introducing *d*, the number of bits needed to encode an element of the matrix. In this way the formula is also valid for arbitrary transition matrices as found in mechanical engineering, for example. However, since in our cases d=1 bit (a project participant either does [1] or does not [0] contribute to performing a required function) these additional two terms evaluate to zero.

For the new design, based on AD principles, we have two diagonal transition matrices and both matrices happen to be 3×3 identity matrices:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

Based on inequality (8), we have:

$$2K(I_3) \le 2(\log^* n + n \log d) \approx 4.5 \text{ bits} \tag{14}$$

It is clear that the design produced using AD principles is much simpler. GSD Development projects, at one stage of life, may well satisfy Axioms I and II but may lose this design quality (through uncontrolled change), because uncontrolled change reduces the likelihood of success of the change process and the above quality may even be lost permanently.

The second hypothetical example (below) demonstrates the application of the third axiom as a solution to this problem i.e. to axiomatically design the GSD *planning* project to reduce the danger of causing uncontrolled change to the GSD *development* project(s).

B. Hypothetical example two (application of axiom III in designing the 'GSD planning project' that creates GSD development projects *X*, *Y* and *Z*):

Let *N* be the network which is the aggregation of n globally distributed software development companies as collaborative partners $P = \{p_1, p_2, ..., p_n\}$. The network *N* is managed by a Network Office *M*. *M* utilises *N* to form a number of 'GSD planning projects', Pr_X, Pr_Y, Pr_Z… to create 'GSD development projects' such as *X*, *Y* and *Z* etc. Each of these GSD development projects is expected to operate as a well managed (virtual) enterprise (VE), complete with its own software development and management processes. The 'GSD development projects' activities are performed by a set of GSD companies collaborating to create the value chain of the respective GSD development projects. We use P_X, P_Y and P_Z to denote the *sets* of associated GSD collaborating partner companies for GSD development projects *X*, *Y* and *Z* respectively. Given a GSD company in the network *N,* at any one time it may (or may not) participate in one or more GSD development projects. Therefore, we have:

$$P_X \subseteq P, P_Y \subseteq P, P_Z \subseteq P \tag{15}$$

We assume that the network *N* that designs, creates and changes GSD development projects already exists (e.g. may have been created by the network office *M*).

Now consider the GSD development project's planning/creation project Pr_X. Pr_X has the functional requirements listed below:

- *FR1*: Provide the Identification and Concept development of GSD development project *X* and specify all of its requirements (functional and non-functional),
- *FR2*: Provide the Preliminary or Architectural Design of GSD development project *X* (Estimate cost, resources needed, select project members P_X, etc.),
- *FR3*: Provide the Detailed Design descriptions of *X*, including all the tasks that must be carried out, personnel role and skill descriptions and the technology to be used by project personnel, all that is necessary to build or re-build and release GSD development project *X* into operation.

Let the DPs to implement this planning project Pr_X be the following:

- *DP1*: P_{X1} is the set of participants who together identify and develop the concept (such as principles, business model, etc) of GSD development project *X*, (this would typically require the knowledge of at least some feasible architectural solutions, and knowledge of design and build effort needed). Normally, these participants would include the high level stakeholders of project *X*, such as lead companies on the network, and the customer / acquirer of the system to be developed by project *X*;
- *DP2*: P_{X2} is the set of participants who together develop the Architectural Design ('master plan') of GSD development project *X* identifying the list of the selected members, cost and time necessary to build GSD development project *X*, etc. (This would typically be done by reusing existing designs ['reference models' or 'partial models'] where the feasibility of design and building under the constraints of the non-functional requirements is known). Typically this would include some lead engineering and consulting companies of the network, as well as an already appointed project manager (i.e. project *X* effectively participates in its own design). These participants together define how project *X* will be structured, both from the point of view of the

project's operations and the project's management;

- *DP3*: P_{X3} is the set of participants who together develop the detailed design of the common parts of the GSD development project *X* with a list of the qualified GSD companies which creates and releases the new GSD development project into operation. Typically this would be performed by project X's management as a lead participant, plus such network participants who are nedeed to set up the operations of project X (such as contractors who deploy tools for project X participants, train project personnel, etc).

Based on the FRs and the DPs above and the typical feedback loops in the life cycle dependencies between project tasks of Requirements Analysis, Architectural Design, detailed Design and Build, the transition matrix between DPs to FRs is as below:

$$\begin{bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{bmatrix} \tag{16}$$

For the transition matrix

$$M_X = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

We have n=3, d=1, m=6. Based on inequality (7), we estimate the information content:

$$K(M_X) \le \log^* n + \log^* m + \log^* d + \log \frac{n^2!}{m!(n^2-m)!} + m \times \log d \approx 18.6 \text{ bits} \tag{18}$$

According to Axiom III of design: "The system that designs another system not only must apply but also must satisfy the axioms of design".

The GSD planning project (Pr_X) that creates GSD development project *X* could be a system that designs/changes another system (GSD development project *X*). Thus the GSD planning project Pr_X is itself (based on its life cycle dependencies shown in the triangular matrix above) a complex system that not only should design another system that has reduced complexity (namely GSD development project *X*) by applying AD theory, but should also reduce the complexity of itself by being designed to satisfy axioms I and II.

To achieve the above, we shall reduce the direct communication among life cycle activities of GSD planning project Pr_X. Neglecting this communication creates additional complexity in the execution of life cycle activities (FR1, FR2 and FR3) of Pr_X. Notice, that practically, the problem is caused by mixing the information dependencies among the life cycle activities with the control of their (repeated, iterative) invocation. These dependencies may result in unpredictable chaotic states of the GSD planning project Pr_X, and decrease the probability of success of the resulting design (the GSD development project *X*). This effect is well-known in managing complex projects and arises if the information flow among life cycle activities is not managed and controlled.

6. Separation of management functions from operations

Some researchers showed that a considerable amount of the complex communication in GSD is due to the design and architecture life cycle activities of GSD projects (Cataldo et al., 2007). This is in fact the communication that needs to be encapsulated at the management level of GSD planning projects. Sangwan et al. (2006) list a number of critical success factors for GSD projects including reducing ambiguity, facilitating coordination.

What is required to solve the problem of complex communications in the execution of the life cycle activities of the GSD planning project Pr_X, is the reduction of the complexity of the design of the GSD planning project Pr_X itself to guarantee the achievement (or preservation) of the design qualities of GSD development project X. A solution is to allocate a sub-project manager to each life cycle activity (FR1, FR2 and FR3) and to have them take part in the project management board meetings and to communicate 'just' at the management level.

Using this method the project manager of the GSD planning project Pr_X should make the project's life cycle activities as independent as possible by delegating each life cycle activity to independent sub-projects that communicate just through management of each project and hide the unnecessary operational details of each life cycle activity of creating the GSD planning project Pr_X from the rest of the project's operations.

We therefore decompose GSD planning project Pr_X into two parts: Pr_M is the management of the GSD planning project and Pr_O is the operation of the GSD planning subproject. Let FR_M be the functional requirement (to 'Manage' Pr), and FR_O the functional requirement(s) describing what Pr has to actually achieve (i.e., the function of the planning project's 'Operations'). In this case Pr_M (the GSD planning project's management) takes care of the control of the communication among operational boundaries. Thus on the upper level we have:

$$\begin{bmatrix} FR_M \\ FR_O \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} DP_M \\ DP_O \end{bmatrix} \tag{19}$$

The operational function of the GSD planning project can be further decomposed into three functions (i.e., life cycle activities, or 'phases'):

(1) the identification phase,

(2) the architectural design phase and

(3) the detailed design and building phase of the GSD development project X.

During the three phases, there are three corresponding functional requirements:

- FR_{O1}: Provide the Identification and Concept of the GSD development project X and specify all its requirements – based on input/control (received from the GSD planning project's management Pr_M);
- FR_{O2}: Provide the Preliminary or Architectural Design of GSD development project X (Estimates of cost, resources needed, selected GSD companies of the GSD development project X etc.) – based on input/control (received from the GSD planning project's management Pr_M);
- FR_{O3}: Provide the detailed design descriptions, and all the tasks that must be carried out to build or re-build and implement the GSD development project X – based on input/control (received from the GSD planning project's management Pr_M).

Based on the three functional requirements, we construct three DPs:

- DP_{O1}: Pr_{O1} identifies different GSD development project (VE) types, develops their master plan based on existing preliminary design of partial models of the new GSD development project X, and provides a detailed design of common parts of project X with a list of the qualified GSD companies.
- DP_{O2}: Pr_{O2} provides the Architectural Design of the GSD development project X with a list of the selected GSD companies for Architectural Design of the GSD development project;

- DP_{O3}: Pr_{O3} creates and operates the new GSD development project, and monitors the results of GSD development project X.

The relationship between the functional requirements and the DPs can be expressed as:

$$\begin{bmatrix} FR_{O1} \\ FR_{O2} \\ FR_{O3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} DP_{O1} \\ DP_{O2} \\ DP_{O3} \end{bmatrix} \quad (20)$$

Under the new design approach, we have two transition matrices which are actually two identity matrices I_2 and I_3, and based on inequality (8), we have:

$$K(I_2) + K(I_3) \leq \log^* 2 + \log^* 3 \approx 3.3 \text{ Bits} \quad (21)$$

Compared with the original design, which has the complexity of the design matrix of about 18.6 bits, the design based on the AD principles is significantly simpler. Note that the reader may suspect a 'trick' in this design, because the internal management process of the GSD planning project's management Pr_M needs to channel the communication among invocations of life cycle activities. This is true of course, however, the separation of *'content'* from *'control'* has a significant effect: the GSD planning project's management Pr_M only needs to know about the state of the information maintained by the subprojects, not the content. For example, managers of large projects normally use controlled information/version release processes so as to avoid project instability and ensure convergence. Note also that the method is not to be taken as a counter-argument against collaborative design, where frequent communication across contributing teams is advantageous – after all Pr_{O1}, Pr_{O2}, and Pr_{O3} possibly share contributors and teams, but their contribution plays different roles.

The point is that instead inter phase document releases being uncontrolled, these releases are managed (each release flowing from a life cycle activity instance to another life cycle activity instance), while intra-phase communication (for cooperation and collaboration) is not management regulated.

Further work will be needed to study the complexity of GSD planning and development project life histories (as opposed to the structure that was studied here), i.e., how to apply the above design axioms (and associated design methods) to reduce the complexity of dependencies among life cycle activity *instances* of GSD planning and development projects. This is an interesting new problem, because due to iterations and feedback most life cycle activities will be performed several times during the project, thus there is scope for the development of a new type of complexity reduction method.

7. Conclusions and future research directions

As discussed in the Introduction, literature agrees that global software development is particularly sensitive to the complexity arising from the linkages between participants that perform the processes of project planning as well as software development. In case of co-located developers, projects of the same complexity may experience fewer problems due to ease and speed of communication. Therefore the contributions of this article (discussing complexity measurement and complexity reduction measures) are of particular relevance to *global* software development.

In this paper we reviewed how the complexity of GSD projects can be reduced using Extended AD theory in order to increase their probability of success. In the first hypothetical example we demonstrated a coupled design for a GSD development project X (a bad design which is more complex than needed) and how we can apply the first two design axioms to arrive at an uncoupled, less complex design.

The second hypothetical example shows a decoupled design for a GSD planning project Pr_X (a project which designs and creates a GSD development project X) and shows the

application of Axiom III to reduce the complexity of the project (which designs, creates, implements, or changes, *X*).

We applied a known approximation of the upper bound of KC to calculate a proxy of Num Suh's 'Information Content' measure and compared the bad and the good designs by calculating the (approximate) complexity/information content of the design matrix. We therefore demonstrated in two hypothetical examples how one can reduce the complexity of designing GSD planning and development projects as 'designing' and 'designed' systems respectively. By satisfying all three axioms the GSD management office *M* should attempt to make the life cycle activities of GSD planning and development projects as independent, controlled and uncoupled as possible so that the designer can predict the future states of these projects and avoid a potentially chaotic behavior.

For further research, we plan to take an empirical research strategy to demonstrate the application of the Extended AD theory using data from real GSD case studies, which would validate and verify the outcomes in practical situations.

To our knowledge, supporting the axiomatic design process by estimating the information content of a design as measured by KC, is an original contribution of this article. Note that the algorithm we used to estimate KC takes only m and n as inputs. A more sophisticated algorithm could consider the distribution of 1s in the design matrix, and therefore reach a lower upper bound, which would therefore be a more accurate estimation of the KC. In reality, we need to balance the difficulty of the algorithms and the accuracy of the results. This means that what the paper proposes is an *approach*, and further research could be done to refine the estimation of the KC upper bound of various systems of interest, such as software development projects.

8. References

Abbott, P., Zheng, Y., Du, R., Willcocks, L. (2013). From boundary spanning to creolization: A study of Chinese software and services outsourcing vendors. *Journal of Strategic Information Systems*, 22(2), 121-136.

Arsenyan, J., Büyüközkan, G. (2009). Modelling collaborative software development using axiomatic design principles. *IAENG International Journal of Computer Science*, 36(3).

Bernus, P., Nemes, L., Schmidt, G. (2003). Handbook on Enterprise Architecture. Berlin: Springer Verlag.

Bernus, P., Nemes, L., Williams, T.J. (1996), *Architectures for Enterprise Integration*. London: Chapman and Hall, p. 368.

Bernus, P., Nemes, L. (1996). A Framework to Define a Generic Enterprise Reference Architecture and Methodology. *Computer Integrated Manufacturing Systems*, 9(3), 179-191.

Carnevalli, J.A. , Miguel, P.A.C., Calarge, F.A. (2010). Axiomatic design application for minimising the difficulties of QFD usage, *International Journal of Production Economics*, 125(1), 1–12.

Cataldo, M., Bass, M., Herbsleb, J.D., Bass, L. (2006). Managing complexity in collaborative software development: on the limits of modularity. In *Proceedings of Supporting the SocialSide of Large ScaleSoftware Development, the CSCW Workshop*, pp. 15-18.

Cataldo, M., Bass, M., Herbsleb J.D., Bass, L. (2007). On coordination mechanisms in global software development. In *Proceeding of the Second IEEE International Conference on Global Software Engineering*, pp. 71-80.

Chen, K.Z., Feng, X.A. (2004). CAD modeling for the components made of multi heterogeneous materials and smart materials. *Computer-Aided Design*, 36(1), 51–63.

Chen, K.Z., Feng, X.-A., Zhang, B-B. (2003). Development of computer-aided quotation system for manufacturing enterprises using axiomatic design. *International Journal of Production Research*, 41(1), 171–191.

Chen, S.J., Chen, L.C., Lin, L. (2001). Knowledge-based support for simulation analysis of manufacturing cells. *Computers in Industry*, 44(1), 33–49.

Cover, T.M., Thomas, J.A. (2006). *Elements of Information Theory* (*2nd Edition*), Wiley Series in Telecommunications and Signal Processing, Hoboken, NJ: John Wiley and Sons, Inc.

D'Mello, M., Eriksen, T.H. (2010). Software, sports day and sheera: culture and identity processes within a global software organization in India. *Information and Organization*, 20(2), 81-110.

Do, S.H., Park, G.J. (1996). Application of design axioms for glass-bulb design and software development for design automation. In *Proceedings of the 3rd CIRP Workshop on Design and Implementation of Intelligent Manufacturing*, pp. 119-126.

Gershenson, C. (2007). *Design and control of self-organizing systems*. Mexico City: CopIt ArXives.

Harutunian, V., Nordlund, M., Tate, D., Suh, N.P. (1996). Decision making and software tools for product development based on axiomatic design. *Annals of the CIRP*, 45(1), 135–139.

Huang, G.Q. (2002). Web-based support for collaborative product design review. *Computers in Industry*, 48(1), 71–88.

Huang, G.Q., Jiang, Z. (2002). Web-based design review of fuel pumps using fuzzy set theory. *Engineering Applications of Artificial Intelligence*, 15(6), 529–539.

IITF(1999). GERAM: Generalised enterprise reference architecture and methodology, v1.6.3. IFIP-IFAC Task-Force 1999, (also available as Chapter 2, Bernus, P., Nemes, L. and

Schmidt,G. (eds.), *Handbook on Enterprise Architecture*, Berlin: Springer Verlag), 2003.

ISO15704 (2000, Amd. 2005). Industrial automation systems–requirements for enterprise-reference architectures and methodologies, Geneva : ISO TC184.SC5.WG1. Geneva, ISO TC184.SC5.WG1, 2000, Amd. 2005.

Kandjani, H., Bernus, P. (2011a). Engineering self-designing enterprises as complex systems using extended axiomatic design theory. *IFAC Papers OnLine V18 (Part1)* Amsterdam: Elsevier, pp. 11943-11948.

Kandjani, H., Bernus, P. (2011b). Capability maturity model for collaborative networks based on extended axiomatic design theory. In L.M. Camarinha-Matos, A. Pereira-Klen and H. Afsarmanesh (editors), Adaptation and Value Creating Collaborative Networks, *IFIP AICT 362*, pp. 421-427, Berlin: Springer verlag.

Kim, S.J., Suh N.P., Kim S.-K. (1991). Design of software systems based on axiomatic design. *Annals of the CIRP*, 40, 165-170.

Kolmogorov, A.N. (1969). On the logical foundations of information theory and probability theory. *Problems of Information Transmission*, 1(1), 1-7.

Kulaka, O. (2010). Applications of axiomatic design principles: A literature review. *Expert Systems with Applications*, 37(9), 6705–6717.

Li, M., Vitányi, P.M.B. (2008). *An introduction to Kolmogorov complexity and its applications*. Berlin: Springer verlag.

Lindkvist, L., Söderberk, R. (2003). Computer-aided tolerance chain and stability analysis. *Journal of Engineering Design*, 14(1), 17–39.

Lloyd S. (2001). Measures of complexity: a nonexhaustive list. *IEEE Control Systems Magazine*, 21, 7-8.

Marczak, S., Damian, D. (2011). How interaction between roles shapes the communication structure in requirements-driven collaboration. In *Proceeding of the 19th IEEE International Conference on Requirements Engineering* (*RE*), pp. 47-56.

Melvin, J.W. (2003). Axiomatic system design: chemical mechanical polishing machine case study. Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge : MIT.

Nannen. V. (2010). A short introduction to model selection, Kolmogorov complexity and Minimum Description Length (MDL). *Arxiv preprint arXiv:1005.2364*.

Oshri, I., Kotlarsky, J., Willcocks, L.P. (2007). Global software development: Exploring socialization and face-to-face meetings in distributed strategic projects. *Journal of Strategic Information Systems*, 16(1), 25–49.

Pimentel, A.R., Stadzisz, P.C. (2006). A use case based object-oriented software design approach using the axiomatic design theory. In *Proceedings of the 4th International Conference on Axiomatic Design*, ICAD2006, pp. 1-8.

Prikladnicki, R., Audy, J.L.N., Evaristo, R. (2006). A reference model for global software development: findings from a case study. In *Proceeding of the International Conference on Global Software Engineeribng*, ICGSE '06, pp. 18-28.

Sahay, S., Nicholson, B., Krishna, S. (2003). *Global IT Outsourcing: Software Development Across Borders*. Cambridge: Cambridge University Press.

Sangwan R., Mullick N., Bass M. (2006). *Global Software Development Handbook*. Boca Raton, FL: CRC Press.

Shin, G.S., Yi, J.W., Yi, S.I., Kwon, Y.D., Park, G.J. (2004). Calculation of information content in axiomatic design. In *Proceedings of the 3rd International Conference on Axiomatic Design*, ICAD2004, pp. 1-6.

Šmite, D., Borzovs, J. (2008). Managing uncertainty in globally distributed software development projects. *Computer Science and Information Technologies*, 733, 9-23.

Suh, N.P. (1990). *The principles of design*, 226, New York: Oxford University Press.

Suh, N.P. (1997). Design of Systems. *Annals of CIRP*, 46(1), 75-80.
Suh, N.P. (1999). A theory of complexity, periodicity and the design axioms. *Research in Engineering Design*, 11, 116-132.
Suh, N.P. (2001). *Axiomatic Design: Advances and Applications*, New York: Oxford University Press.
Suh, N.P. (2005). Complexity in engineering. *CIRP Annals-Manufacturing Technology*, 54(2), 46-63.
Suh, N.P. (2007). Ergonomics, axiomatic design and complexity theory. *Theoretical Issues in Ergonomics Science*, 8(2), 101–121.
Suh, N.P., Do, S. (2000). Axiomatic design of software systems. *CIRP Annals-Manufacturing Technology*, 49(1), 95-100.
Togay, C., Dogru, A.H., Tanik, C.U. (2008). Systematic component-oriented development with axiomatic design. *The Journal of Systems and Software*. 81(11), 1803–1815.
Yi, J.W., Park, G.J. (2005). Development of a design system for EPS cushioning package of a monitor using axiomatic design. *Advances in Engineering Software*, 36(4), 273–284.

Appendix

Generalised Enterprise Reference Architecture and Methodology (GERAM)

The GERAM framework (Bernus and Nemes, 1996; IITF, 1999; ISO15704, 2000, Amd. 2005) defines a comprehensive set of concepts to represent and explore enterprise systems. GERAM is a "toolkit of concepts for designing and maintaining enterprises for their entire life history" (ibid) and the objective of this framework is "to systematise various contributions of the field that address the creation and sustenance through life of the enterprise, here GSD project, as a complex system".

An explanation of life cycle and life history concepts used in this article

A. Life cycle (LC): Life cycle 'phases' are types of activities, and associated abstraction levels, pertinent to change in the entity's life, and encompass all activities from identification to decommissioning (or end of life). Figure 1 shows the GERA (Generalised Enterprise Reference Architecture) life cycle for any enterprise or its entities.

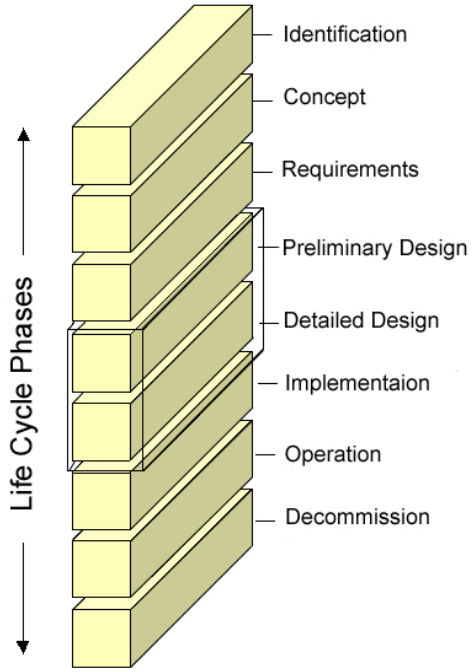

Figure 1. GERA life cycle phases for any enterprise or enterprise entity

The seven life cycle activity types may be further subdivided, e.g. design can be subdivided into preliminary- and detailed design. (Note: LC activities have no temporal connotation, or assumed sequence.)

B. Life history: The life history of a business entity is the representation in time of tasks carried out on the particular entity during its entire life span. Relating to the life cycle concept described above, the concept of life history allows us to identify the tasks pertaining to these different phases as activity types. This demonstrates the iterative nature of the life cycle concept compared with the time sequence of life history. These iterations identify different change processes.

Typically, multiple change processes are in effect at any one time, and all of these may run parallel with the operation of the entity. Moreover, change processes may interact with one another. Within one process, such as a continuous improvement project, multiple life cycle activities would be active at any one time. For example, concurrent engineering design and

implementation processes may be executed within one enterprise engineering process with considerable time overlap, and typically in parallel with the enterprise operation. Life histories of entities are all unique, but all histories are made up of processes that in turn rely on the same type of life cycle activities as defined in the GERA life cycles. For this reason life cycle activities are a useful abstraction in understanding the life history of any entity.

Figure 2 illustrates the relations between life cycle and life history representing a simple case with a total of seven processes: three engineering processes, three operational processes, and one decommissioning process.

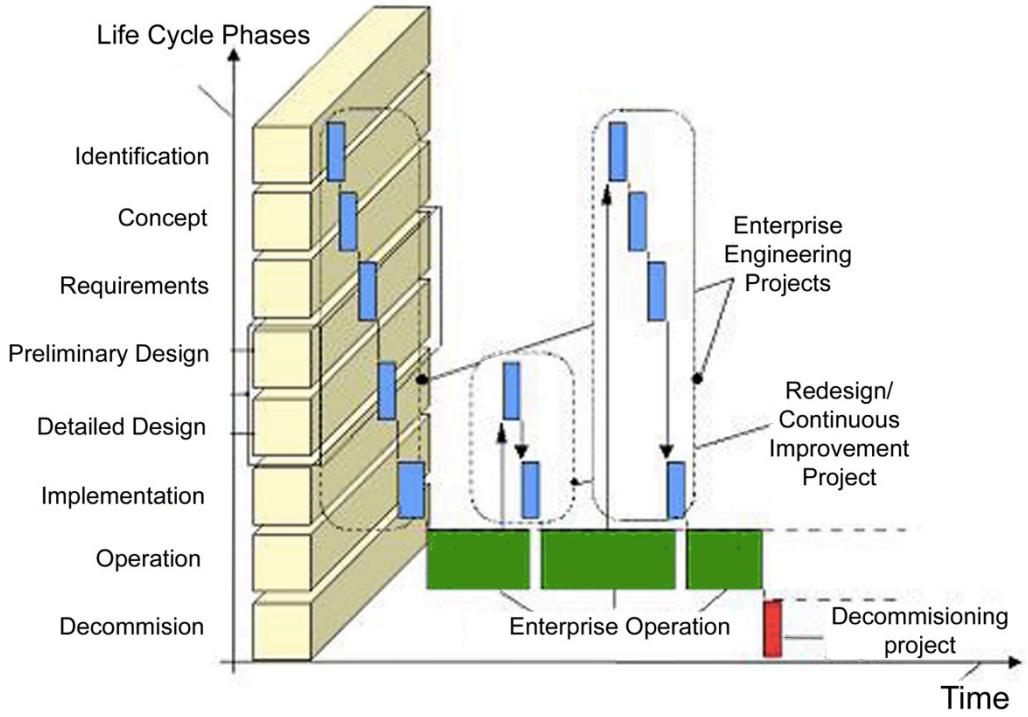

Figure 2. Parallel Processes in the Entity's Life-history