Global Software Development: Measuring, Approximating and Reducing the Complexity of Global Software Development Using Extended Axiomatic Design Theory

Hadi Kandjani
Centre for Enterprise Architecture Research and Management (CEARM)
School of ICT, Griffith University, Brisbane, Australia
Email: H.Kandjani@grifith.edu.au

Peter Bernus
Centre for Enterprise Architecture Research and Management (CEARM)
School of ICT, Griffith University, Brisbane, Australia
Email: P.Bernus@griffith.edu.au

Lian Wen
Institute for Integrated and Intelligent Systems (IIIS)
School of ICT, Griffith University, Brisbane, Australia
Email: L.Wen@griffith.edu.au

Abstract

This paper considers GSD projects as designed artefacts, and proposes the application of an Extended Axiomatic Design theory to reduce their complexity in order to increase the probability of project success. Using an upper bound estimation of the Kolmogorov complexity of the so-called 'design matrix' (as a proxy of Information Content as a complexity measure) we demonstrate on two hypothetical examples how good and bad designs of GSD planning compare in terms of complexity. We also demonstrate how to measure and calculate the 'structural' complexity of GSD projects and show that by satisfying all design axioms this 'structural' complexity could be minimised.

Keywords

Global Software Development, Complexity, Extended Axiomatic Design Theory, Kolmogorov Complexity

INTRODUCTION

A Global Software Development (GSD) project has to go through complex processes to finish projects within 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. This structure in itself increases the complexity of software processes (Šmite and Borzovs 2008), where part of this complexity is due to dynamic dependencies among components of the software product (Cataldo et al. 2006) and to dependencies among life cycle activities of project planning and product development. 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, therefore these projects could be looked at as intrinsically complex adaptive systems: they can not 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 are various kinds of engineered systems, including software products, developed using a global engineering effort. Common to all is a highly complex (or complicated) project design, as many of these project have been usually designed "without having a theoretical framework for complexity" (Suh 2005). GSD therefore is becoming more complicated unless fundamental theories and principles for reducing complexity are developed (or adopted from 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 question that may arise is: "What is Complexity?"

> **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_{ej}, and the complexities of the interactions C_{ik}" (Gershenson 2007).

Axiomatic Design (AD) Theory (Suh 2001) defines a 'complex' system as one that can not be predicted to always satisfy its functional requirements. Suh (2001) and other authors, such as (Melvin, 2003), target the concept of 'the probability of satisfying all functional requirements all the time'. Functional requirements are defined in axiomatic design as "a minimum set of independent requirements that completely characterize the functional needs of a product (software, organization, systems, etc.) in the functional domain" (Suh 1990; Suh 2001). 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), thus the need for a design theory that reduces the complexity of a system.

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". Suh believes the idea that 'physical things with various parts are inherently more complex' "may or may not be true from the functional point of view", because "the complexity defined in the functional domain is a measure of uncertainty in achieving a set of tasks defined by FRs in the functional domain". 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). Considering a GSD project as an artefacts 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 performing what it needs to do).

This paper has the following structure: the introduction reviews the problem of complexity in GSD. Subsequently we review a reference model for GSD projects and Extended Axiomatic Design theory, and use this theory to address the complexity of GSD planning and development projects. After these reviews, we use an upper bound estimation of the Kolmogorov complexity of the design matrix (as a proxy measure of Axiomatic Design theory's Information Content metric). Using this proxy it is possible to measure the complexity of design. Finally, in two hypothetical examples, we (a) compare both good and bad designs of GSD planning projects and (b) compare good and bad designs of GSD development projects from the complexity point of view. Using two hypothetical examples, we also demonstrate how to measure and calculate the 'structural' complexity of GSD projects and show that by satisfying all design axioms this 'structural' complexity could be minimised.

A REFERENCE MODEL FOR GLOBAL SOFTWARE DEVELOPMENT

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

A. Organizational dimension (Planning phases) Prikladnicki et al. (2006) state that the planning phase is important to more properly organise and manage the distributed projects. They identified the initial planning as a formal and basic phase 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 different globally developed software products." Based on their case studies, they proposed the GSD planning phase as a former life cycle activity of many project cycles that are in fact derived from the planning process.

B. Project dimension (Development phases) This includes (in Prikladnicki et al.'s sense (2006)) "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, build, integration, test…, and release the end product into operation. We interpret these dimensions as two phases of a) GSD planning life cycle activities and b) GSD development life cycle activities.

COMPLEXITY ADDRESSED BY AXIOMATIC DESIGN THEORY

As we try to solve the difficulty of having to use complex design descriptions in GSD projects, we turn to Axiomatic Design Theory's complexity measures. Axiomatic Design (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 2011). Note that AD proposes techniques for reducing complexity in multiple

engineering domains (incl. software development (Suh and Do 2000). AD is a theory of complex systems (that can not be predicted for sure to always satisfy their functional requirements (Suh 1990)). AD explains reasons of emerging complexity, and offers a formal design theory and two design axioms that system designs must satisfy to minimise complexity (measured by the probability that the structure always performs the function).

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 axiomatic design creates "uncoupled or decoupled interrelationships and arrangements among 'modules', and is easy to change, modify, and extend" (Suh and Do 2000).

In this paper we apply AD to GSD development projects as well as GSD planning processes that design and change the GSD development projects. Arsenyan and Büyüközkan (2009) also presented an AD based collaboration model within the context of software industry. They proposed a model for collaborative software development structure based on AD and defined goals, strategies and methodologies that influence the 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.

Axiom I: Independence Axiom (Suh 1990). *'The independence of Functional Requirements (FRs) must always be maintained.*' (An FRi is independent of others if there exist 'design parameters' [DP] so that if changing one FRi only one DPi must change, whereupon [FR] = [[A]] * [DP]. Here [FR] is the vector of FRs, [DP] is the vector of DPs and [[A]] is the matrix mapping DPs to FRs. If [[A]] is diagonal then the design is uncoupled (full independence is achieved). If [[A]] is triangular then the design is decoupled (the implementation process is 'serializable'). Otherwise the design is coupled (the implementation process of DPs is not 'serializable').

Axiom II: Information Axiom (Suh 1990) *'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', meaning that the system always satisfies its FRs. In this paper we use an upper bound estimation of the Kolmogorov complexity of the design matrix as a proxy of Suh's Information Content.

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, observe that the complexity of GSD planning processes: the processes that create a GSD project) is not automatically addressed by introducing AD. Therefore, Axioms I & 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 & II, but they themselves need to be 'axiomatically designed' (Kandjani and Bernus 2011).

Systems (here GSD development projects) at one stage of life may satisfy Axioms I & II but may lose this design quality as they evolve / change, and through reducing the likelihood of success of the change process this quality may even be lost permanently. To prevent such 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 & II. Pragmatically: a GSD development project as large and complex system is created by GSD planning projects (also as complex systems) to the design of which axiomatic design 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 the axioms I and II.

Axiom III: Recursion Axiom (Kandjani and Bernus 2011): '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 & II to design a system, that process is best which itself satisfies axioms I & II".

If a GSD project wishes to reduce its own complexity as well as to subsequently maintain reduced complexity through 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 and to increase the probability of success.

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 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 Kolmogorov complexity of the design matrix as a proxy of Num Suh's Information Content.

A. Definitions

The concept of Kolmogorov complexity 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. Kolmogorov complexity 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, Kolmogorov complexity has been applied in various disciplines (Li and Vitányi 2008).

The Kolmogorov complexity $K_U(x)$ 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:

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

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 halt. In this case, we define the *conditional Kolmogorov complexity* as:

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

Theorem 1 If *U* is a universal computer, $\forall V$ which is another universal computer, $\exists c$ is a constant, so for $\forall x \in \{0,1\}^*$ (i.e. for each binary number *x*)

$$K_U(x) \le K_V(x) + c$$

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:

$$\log^* n = \log n + \log\log n + \log\log\log n + ...$$

until the last positive term.

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

$$K(n) \le \log^* n + c$$

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 Kolmogorov complexity *K(n|l(n))* satisfies:

$$K(n \mid l(n)) \le \log n + c$$

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 Kolmogorov complexity.

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 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 Kolmogorov complexity of *M* as:

$$K(M) \le K(n) + K(m) + K(\binom{n^2}{m}) \le \log^* n + \log^* m + \log \frac{n^2!}{m!(n^2-m)!} + c \quad (1)$$

If *M* is a diagonal matrix, because all the none 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 Kolmogorov complexity of an identity matrix can be estimated as:

$$K(I_n) = K(n) \le \log^* n + c \quad (2)$$

HYPOTHETICAL EXAMPLES

We introduce two hypothetical examples to demonstrate the application of the three design axioms. The first hypothetical example demonstrates an example of a coupled design (a bad design which is more complex) for a virtual enterprise (GSD Development project *X*) and applies the first two design axioms which results in an uncoupled design. The second hypothetical example also demonstrates an 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 project which designs, creates and implements GSD Development project *X*.

We use an upper bound estimation of the Kolmogorov complexity 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 axiomatic design theory can reduce the complexity of designing a GSD Development project *X* as a system of interest, as well as the complexity of the GSD planning project (Pr_X) as a system which designs the system of interest.

GSD Development projects, at one stage of life, may well satisfy Axioms I & 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. Therefore the second hypothetical example demonstrates the application of the third axiom as a solution to this problem i.e. the problem of complex GSD planning projects. Note that we satisfy the FRs by means of design parameters (DP). 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: Application of the Axiom I & II in Designing a Virtual Enterprise X

'GSD Development project X', which is actually a virtual enterprise, 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 a architeture design.
FR2:Sub1 needs component development and database module.
FR3: Sub2 needs component development and GUI development.
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 design parameters to implement these functions are as follows:

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

Based on the FRs and the DPs above, we can write the FR to DP mapping formula for GSD Development project *X* as:

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

It is clear that the design transition matrix is coupled.

According to the first axiom of design, we must maintain the independence of the functional requirements all the times. Therefore to apply axiomatic design 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 design parameters as follows:

FR_1: Each sub-system needs to have a architecture.
FR_2: Each sub-system needs to be implemented.
FR_3: All sub-systems need to be unit and integrating tested.

DP_1: company A provides service of architecture design.
DP_2:company B provides service of software implementation.
DP_3: company C provides unit and integrating testing.

Then the FR-DP transition matrix for GSD Development project *X*' is shown below is a diagonal matrix:

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

The FRs and DPs for the GSD broker company *B* are:

FR_1: Some sub-systems need compoment 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 is a diagonal 3×3 matrix as well. Let us now calculate the Kolmogorov complexity of each transition matrix of the GSD Development case study. In the original design, the transition matrix *M* is:

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

For this transition matrix, we have *n*=5, *m*=9, so based on inequality (1), we have:

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

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

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Based on inequality (2), we have:

$$2K(I_3) \leq 2(\log^* n + n\log d) \approx 4.5 \ bits$$

It is clear that the design based on Axiomatic Design principles is much simpler.

B. Hypothetical Example Two: Application of Axiom III of Design in Designing the 'GSD planning project' that create GSD Development projects X, Y and Z

Let *N* be the network which is the aggregation of n global software development companies as collaborative distributed partners $P = \{p_1, p_2, ..., p_n\}$. The network *N* is managed by a Network Office *M*. *M* utilizes *N* to form a number of 'GSD planning projects', Pr_X, Pr_Y, Pr_Z… to create 'GSD development projects' (VEs), such as *X*, *Y* and *Z* etc. Each 'GSD development projects' as consists of 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 companies for GSD development projects *X*, *Y* and *Z* respectively. Each GSD company may (or may not) participate in one, two or all (any) GSD development projects. Therefore, we have:

$$P_X \subseteq P, \ P_Y \subseteq P, \ P_Z \subseteq P, \ \phi \subseteq P_X \cap P_Y, \phi \subseteq P_X \cap P_Z, \phi \subseteq P_Y \cap P_Z.$$

We suppose that the network *N* that designs, creates and changes GSD development projects (including *X*) 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 of GSD development project X and specify all its requirements (functional and non-functional),*
>
> *FR2: Provide the Preliminary or Architectural Design of GSD development project X (Estimate cost, resources needed, selected members etc.),*
>
> *FR3: Provide the detailed design descriptions, and all the tasks that must be carried out to build or re-build and release GSD development project X into operation.*

Let the design parameters to implement this project 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);*
>
> *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);*
>
> *DP3: P_{X3} detailed design of the common parts of the GSD development project X with a list of the qualified GSD companies creates and releases the new GSD development project into operation.*

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

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

For the transition matrix

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

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

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

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 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 axiomatic design theory, but it has to also have reduced complexity and be 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 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 acticities is not managed and controlled.

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 to reduce 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 high level we have:

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

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 & 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 design parameters:

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 design parameters can be expressed as:

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

Under the new design approach, we have two transition matrices which are actually two identity matrices I_2 and I_3.

Based on inequality (2), we have:

$$K(I_2) + K(I_3) \leq \log^* 2 + \log^* 3 \approx 3.3 \ \textit{bits}$$

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, after all Pr_{O1}, Pr_{O2}, Pr_{O3} possibly share contributors and teams, but their contribution is in different roles.

Further work will be needed to study the complexity of GSD planning and development project life histories (as opposed to 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.

CONCLUSIONS

In this paper we reviewed how the complexity of GSD projects can be reduced using Extended Axiomatic Design 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 Kolmogorov complexity 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 behaviour. For further research, the authors plan to

take an empirical research strategy to demonstrate the application of the Extended Axiomatic Design theory using data from real GSD case studies, which would validate and verify the outcomes in real practice.

REFERENCES

Arsenyan, J., and Büyüközkan, G. 2009. "Modelling Collaborative Software Development Using Axiomatic Design Principles," *IAENG International Journal of Computer Science*, vol. 36(3).

Cataldo M., Bass M., Herbsleb J. D., and 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 2006*, pp 15-18.

Cataldo M., Bass M., Herbsleb J. D., and Bass L. 2007. "On coordination mechanisms in global software development," in *Proceeding of the Second IEEE Int. Conf. on Global Software Engineering*, pp 71-80.

Cover T. M. and Thomas J. A. 2006. *Elements of Information Theory 2nd Edition*, (Wiley Series in Telecommunications and Signal Processing), Wiley-Interscience, 2 edition.

Do S.H. and Park G.J., 1996, "Application of Design Axioms for Glass-Bulb Design and SoftwareDevelopment for Design Automation," in *Proc 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.

Kandjani, H. and P. Bernus 2011. "Engineering Self-Designing Enterprises as Complex Systems Using Extended Axiomatic Design Theory," *IFAC Papers OnLine V18 (Part1)* Amsterdam *:* Elsevier pp11943-11948.

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

Kolmogorov A. N. 1969. "On the Logical Foundations of Information Theory and Probability Theory," *Problems of Information Transmission*., vol. 1(1), pp. 1-7.

Li M. and Vitányi P. M. B. 2008. *An introduction to Kolmogorov complexity and its applications*. Springer.

Marczak S. and 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, Dept. 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*.

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 Int Conf on Axiomatic Design*, ICAD2006, pp 1-8.

Prikladnicki R., Audy J. L. N., and Evaristo R. 2006. "A reference model for global software development: findings from a case study," in *Proceeding of the Int Conf on Global Software Eng*, ICGSE '06, pp. 18-28.

Sangwan R., Mullick N., and Bass M. 2006. *Global software development handbook*: 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 *Proc 3rd Int Conf on Axiomatic Design*, ICAD2004, pp 1-6.

Šmite D. and Borzovs J. 2008. "Managing Uncertainty in Globally Distributed Software Development Projects," *Computer Science and Information Technologies*, vol. 733, pp. 9-23.

Suh N. and Do S. 2000. "Axiomatic design of software systems," *CIRP Annals-Manuf. Tech*, vol.49, pp. 95-100.

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

Suh N. P. 1999. "A theory of complexity, periodicity and the design axioms," *Research in Engineering Design*, vol.11, pp. 116-132.

Suh N. P. 2001. *Axiomatic design: advances and applications*, ed: Oxford University Press, New York.

Suh, N. P. 2005. "Complexity in engineering," *CIRP Annals-Manufacturing Technology*, 54(2): 46-63.

Suh, N.P., 1997, "Design of Systems," *Annals of CIRP*, Vol. 46, pp 75-80.

COPYRIGHT