Enterprise Architecture Cybernetics for Complex Global Software Development

Reducing the Complexity of Global Software Development Using Extended Axiomatic Design Theory

Hadi Kandjani, Peter Bernus, Lian Wen
Centre for Enterprise Architecture Research & Management,
School of ICT, Griffith University, Brisbane, Australia
{H.Kandjani, P.Bernus, L.Wen}@griffith.edu.au

Abstract— **Global Software Development projects could be best understood as intrinsically complex adaptive living systems: they can not purely be considered as 'designed systems', as deliberate design/ ontrol episodes and processes (using 'software engineering' models) are intermixed with emergent change episodes and processes (that may perhaps be explained by models). Therefore the evolution of GSD projects includes the emergent as well as the deliberate aspects of system change. So to study GSD projects as complex systems we need to focus on both the state of the art of GSD research, as addressed in the software engineering discipline, as well as other disciplines that studied complexity such as Enterprise Architecture, Complexity and Information Theory, Axiomatic Design theory, for example. In this paper we study the complexity of GSD projects and propose the application of Extended Axiomatic Design (EAD) theory to reduce the complexity of GSD projects and to increase their probability of success. We also demonstrate that by satisfying all design axioms this 'structural' complexity could be minimised. By satisfying all three axioms of EAD, GSD management could 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 next relevant states of the life history and avoid a chaotic change in such projects.**

Keywords; Global Software Development; Complexity; Enterprise Architecture; Extended Axiomatic Design Theory; Self-designing

I. INTRODUCTION

Global Software Development (GSD) as a practice has to go through complex processes to complete projects within allocated budget, allotted time schedule, and with all customer attributes and functional requirements satisfied [1]. The concept of GSD assumes distributed teams from different organisations and/or geographical locations collaborating to design, manage and execute life cycle activities of a joint GSD project, mush like a supply chain, increasing the complexity of software processes [1] due to the complexity of project coordination (across temporal and geographical distances), and communication (cultural diversity and lack of proximity) [1].

Part of this complexity is due to dynamic dependencies among components of GSD products [2] as well as dependencies among life cycle activities of GSD planning and development activities. This complexity is an emergent property as planning for and development of GSD projects embraces uncertainty and ambiguity due to the high number of elements as well as the numerous dependencies among GSD products, projects or project activities. E.g. the emergent communication structure among the diverse roles in the project does not always follow a deterministic and planned communication structure prescribed by the organization [3].

The above emergent behaviour is explained if GSD projects are understood as intrinsically complex adaptive living 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). The mix of deliberate and emerging processes can create a situation in which the GSD project as a system is in dynamic equilibrium (for some stretch of time) – a property studied in General Systems Theory [4].

Therefore the evolution of the GSD project includes the emergent as well as the deliberate aspects of the GSD project change; we believe (and apply this approach in this paper) that to study GSD projects as complex systems we need to focus on the state of the art of GSD research addressed in the software engineering discipline as well as disciplines that studied complexity such as Enterprise Architecture, Axiomatic Design Theory and etc.

In section II of this article, we briefly describe Enterprise Architecture Cybernetics as the research methodology applied in this research, then in section III we review a reference model for GSD projects – through which we identify and address different types of complexity in GSD projects in section V.

Section IV briefly elaborates on the complexity of GSD projects as an important problem facing these projects; and section VI briefly reviews Extended Axiomatic Design theory as well as uses this theory to address different types of complexity in GSD projects.

In section VII, we propose the concept of self-designing Global Software Development and demonstrate by logical argument that self-design may increase the probability of success of GSD

projects through reducing apparent complexity of models necessary for GSD planning and development projects

II. Enterprise Architecture Frameworks for Complex Global Software Development

Enterprise architecture (EA) frameworks e.g., GERAM (Generalised Enterprise Reference Architecture and Methodology) [5-7], aim at advising on a complete collection of tools, methods and models to be used by enterprise engineering/change efforts. GERAM is a "toolkit of concepts for designing and maintaining enterprises for their entire life history (Fig.1)" [*ibid*] therefore we expect that it may be used to systematise various contributions of the field that address the creation and sustenance through life of GSD planning as well as development projects as complex systems.

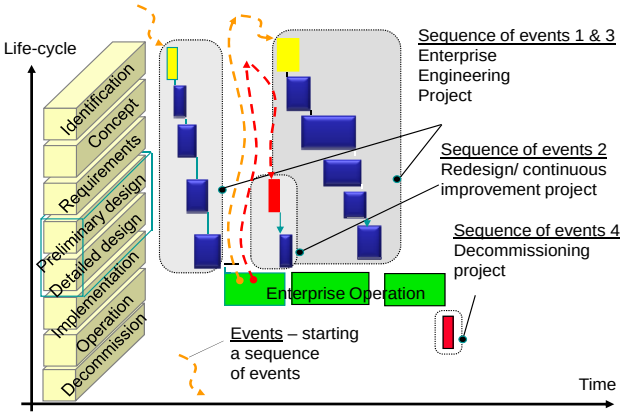

Figure 1. Life History according to GERAM [5-7].

EA as an inter-disciplinary and multidisciplinary discipline, not only applies models, methods and theories of management and control – it also uses the same from engineering, linguistics, cognitive science, environmental science, biology, social science, artificial intelligence, systems thinking and cybernetics. The GERA (Generalised Enterprise Reference Architecture) modeling framework of GERAM reduces the apparent complexity of enterprise models by introducing the view(point) concept as the generalisation of the view(point) concepts of several other architecture frameworks (such as CIMOSA, GRAI and PERA [6]). Different views constructed on the basis of these viewpoints may highlight certain aspects and level of detail of an enterprise entity and hide the rest of aspects (see Fig.2).

It is important to note the completeness of GERA's scope. GERA uses an 'epistemological trick' to achieve completeness; the scope is complete by definition because subdivisions in the GERA meta-model are made in a way that preserves completeness – namely GERA considers:

• everything that is done by humans and everything done by non-humans,

• everything that the system does to satisfy its purpose (manufacture products or provide service for someone else) and everything that the system does for itself (management and control),

• everything done by hard systems ('hardware') and by non-hardware [software] (i.e. 'information that controls the state / configuration of the hard system'),

• everything from the system's functional viewpoint (function and information), as well as the system's structural viewpoint (constituent resources), and the relationship, or mapping, from resources to functions / information (i.e. the system's organisation);

• every process that is done *to* the system (pertaining system change) and *by* the system to create (or change) it, operate it and decommission it (or part of it). These 'life cycle processes' (or 'phases') may be performed by the system, or by the environment (which latter may contain other systems). The processes that create or change the system are further categorised by the same completeness-preserving method: we account for every process that considers the system at a level more abstract than requirements-level (identification and concept), at the requirements-level, and everything that considers the mapping of requirements to structure (architectural design), as well as every process needed to institute the change (detailed design & implementation, building / release to operation)

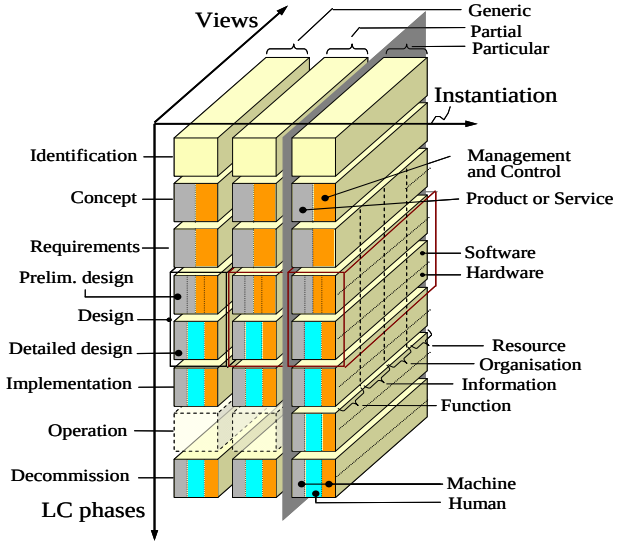

Figure 2. The GERA Modelling Framework of GERAM [5-7].

As a result, to study and deal with the complexity of GSD projects and gain more control over them, it would be helpful to turn to EA frameworks to guarantee a holistic approach as well as the laws and principles of systems thinking and cybernetics which try to tackle the problem of designing and controlling complex systems.

Therefore, to reduce the 'apparent complexity' of the design of GSD projects as complex systems, this paper proposes the application of Enterprise Architecture Cybernetics [8] – a distinct field of the Enterprise Architecture which formalises, synthesises, harmonises and systematises the achievements and results of the systems thinking and cybernetics and demonstrates them in EA practice to design

(and manage the complexity of) enterprises as complex systems (or, in this context of this paper, GSD projects as complex systems).

III. A Reference Model for Global Software Development

Prikladnicki *et al.* [9] proposed a reference model for GSD based on the results of real GSD case studies. Their proposed reference model includes 'organizational' and 'project' dimensions. Although there exist other related reference models developed for GSD, we chose this reference model due to the nature of its classification of dimensions of development and planning areas in GSD.

A. Organizational dimension

Prikladnicki *et al.* [9] argue that the planning phase is important for organising and managing distributed projects. They identified the initial planning as a "formal and basic phase to decide if a project can be distributed and how to plan for its development."

Based on their case studies, they proposed the GSD planning phase as a, initial life cycle activity of many project cycles that are in fact derived from the planning process. They therefore identified two cycles of planning for management of GSD projects. The first one as:

1) The first cycle is 'Strategic planning'

This is conducted by the organisation's headquarters with the purpose of identifying and prioritizing new GSD development projects demanded by external clients or by the network of GSD companies, taking the responsibility of the strategic alignment between goals of each GSD development team, unit or company and their headquarters [9] (what we call the 'network office').

2) The second cycle is: 'tactical-operational planning'

This is conducted in the scope and domain of each GSD development team, unit or companiy from a network of GSD companies. The overlapping of strategic and tactical-operational planning cycles happens when GSD projects are allocated, and involves planning for, resource allocation and choosing projects to be developed in each GSD team, unit or company [9].

B. Project dimension.

This includes (in Prikladnicki *et al.*'s sense [9]) "general coordination of work between collaborators, interfaces among teams, communication, and contacts with clients and conflict solving." However, these functions and activities may also belong to the planning level of GSD projects, whether on the organizational or project level. Therefore, for orthogonality of concepts (proposed in the next sections), we interpret the project dimension as a set of life cycle activities that deal with the requirements analysis, design, build, integration, test…, and release into operation of the product.

IV. Global Software Development as a Complex System

An important problem facing GSD projects is complexity, because uncontrolled complexity can cause undesired design qualities and therefore unsatisfied requirements of GSD projects. First we must adopt a definition of '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}" [10].

Melvin [11] argues that we need large and complex systems to be able to satisfy all functional requirements. On the other hand, Axiomatic Design (AD) Theory [12] defines a 'complex' system as one that can not be predicted to always satisfy its functional requirements. In fact, Melvin and Suh's statements above target the problem 'what is the probability of satisfying all functional requirements all the time'?

V. Measures of Complexity

According to Lloyd [13] there are three questions that are posed when attempting to quantify the complexity of an entity:

a) How hard is it to describe the entity?

b) How hard is it to create the entity?

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

These measures (as interpreted by Kandjani and Bernus [14]) can be classified as those that characterise the difficulty to describe the a) function, behaviour, and states of the GSD development project as a system, and c) architecture (relationship between physical and functional structure of the GSD development project as a system) and b) the GSD planning process/project (which create GSD development projects).

In this case, categories a) and b) measure the complexity of the GSD development projects, and c) measures the complexity of GSD planning projects that designs and create GSD development projects.

Kandjani and Bernus [8] 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 analysing, designing or controlling it).

Groups (a) and (c) of complexity target the complexity of the *project dimension* at the GSD development level as indicated in Prikladnicki *et al.*'s reference model. However, complexity of category (b) targets the complexity of the *organizational dimension* at the planning level for GSD projects. Both of these types of complexity are addressed in Extended Axiomatic Design theory (see Section VI).

VI. Complexity Addressed by Axiomatic Dewsign Theory

Axiomatic Design (AD) [15] claims to codify in a discipline-independent way what a 'best design' is, and in particular aims at avoiding unnecessary complexity of

categories (a) and (c). However, to be able to avoid complexity of category (b) AD had to be extended by introducing the Recursion Axiom, stipulating that the system that designs the system must also obey the axioms of AD [14]. Note that AD proposes techniques for reducing complexity in multiple engineering domains (incl. software development [16]).

The question arises: how to measure the complexity of designing GSD projects and their planning processes, and how to maximise the probability of the success of GSD projects (what design principles to use to this effect)?

A. *A Pragmatic Measure of Complexity*

We earlier proposed [14] the 'number of relevant states of a system's environment' as a proxy for Suh's Information Content described in AD (whereupon two states of the environment are deemed different if the system must respond to them differently). In other words, to function correctly, the system needs to know the relevant states of its surroundings.

By describing (emcoding) these relevant states, the minimum length of the system's description grows with the number of these states, so the number of these relevant states can be used as a complexity measure (and call it the system's 'information content' (IC)). *This* IC as a complexity measure is a proxy of what Suh calls information content (IC_{Suh} = the negative logarithm of the probability that the system always satisfies its functional requirements).

B. *Extended Axiomatic Design Theory for Complex Global Software Development*

Axiomatic Design (AD) is a theory of complex systems; it 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).

Axiom I: Independence Axiom [17]. 'The independence of Functional Requirements (FRs) must always be maintained.' (An FR_i is independent of others if there exist 'design parameters' [DP] so that if changing one FR_i only one DP_i 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 [17] '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'.)

Axioms I and II together intend to minimise the complexity of the system's architecture – complexity type 'c' and can be used to design less complex GSD projects. However, observe that complexity type 'b' (complexity of GSD planning processes: the processes that create, maintain, or change 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' [14].

Axiom III: Recursion Axiom [14]: '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 thereby increase the probability of success.

Based on this Extended Axiomatic Design theory, a Capability Maturity Model was developed for the use of capability assessment and strategy making to reduce the complexity of change systems as well as the complexity of designed systems [8]. This maturity model included both Process- and People maturity levels. These models could also be used as a roadmap for incorporating extended axiomatic design practices and techniques into GSD practice.

VII. Global Software Development as a Self-designing System

Sangwan *et al.* [18] list a number of critical success factors for GSD projects including reducing ambiguity, and facilitating coordination. Marczak and Damian [3] also refer to the importance of communication among diverse roles in projects, and the importance of critical members of the projects "whose absence, whether temporary or permanent, would disrupt the information flow if removed from the project" as well as "new hirees" being isolated from team collaboration.

These critical members of the GSD projects in fact have developed tacit knowledge throughout time being involved in several GSD development as well as planning projects. Therefore models used by these critical members as 'embedded designers' can be simpler because they know the important distinctions between relevant states of the GSD projects and relevant states of the surrounding environment.

Conversely, these critical members of GSD projects as embedded designers can recognise two states of a GSD project as identical from the point of view of relevance to the GSD planning projects (change processes), whereupon the same two states of the GSD change environment would be considered different by new members as 'outsiders' (external observers) who do not have tacit knowledge of this environment.

As a consequence, the requisite variety of the GSD planning project as a change system [19] can be minimised through the GSD planning project being designed by these critical members as embedded designer agents.

Kandjani and Bernus [14] argued that "self-design results in minimised information content (IC) and thus creates a 'best design', from the point of view of the likelihood of success." They derived and explained this as the '*self-evolution conjecture*' explaining that: "Among those change systems which satisfy and apply the first two design axioms, that "change system" is the best which is designed by the stakeholders of the system of interest itself " [14].

In other words that GSD planning project is most likely to succeed whose model used to create and control it is the simplest. The argument behind this is that the GSD management office M (the designer of the GSD network's future) needs to be able to make decisions and plan in light of information about the change environment, and it is the critical members from the GSD collaborating companies with the tacit knowledge developed both at project and planning level who have the least need for explicit information in order to make predictions about and control GSD planning projects.

As a result, to minimise the information content of planning projects and to increase the probability of success of GSD planning projects, we propose that the management office of the collaborative network of GSD companies should include these critical members from GSD companies with the most tacit knowledge of their companies, teams or units in the planning process of GSD projects.

VIII. Conclusions

In this paper we reviewed the complexity of GSD projects and applied Extended Axiomatic Design theory show methods to reduce the complexity of GSD projects and thereby increase their probability of success. By satisfying all three axioms the GSD management 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 next relevant states of the life history and avoid a chaotic change. We also argued that to minimise the information content of planning projects and to increase the probability of success of GSD planning projects, the GSD management office of a collaborative network of GSD companies should include the critical members from GSD companies with the most tacit knowledge of their companies, teams or units in the planning projects for GSD projects, and collaborating GSD companies in GSD development projects should design their change system themselves.

References

[1] D. Šmite and J. Borzovs, "Managing Uncertainty in Globally Distributed Software Development Projects," University of Latvia, Computer Science and Information Technologies, vol. 733, pp. 9-23, 2008.

[2] M. Cataldo, M. Bass, J. D. Herbsleb, and L. Bass, "Managing Complexity in Collaborative Software Development: On the Limits of Modularity," Supporting the Social Side of Large Scale Software Development, p. 15, 2006.

[3] S. Marczak and D. Damian, "How interaction between roles shapes the communication structure in requirements-driven collaboration," in Proc 19th IEEE International Requirements Engineering Conf., IEEE, pp. 47-56, 2011

[4] L. Von Bertalanffy, "General System Theory-Foundations and Developments," New York : George Braziller, Inc, p. l0, 1968.

[5] P. Bernus, L. Nemes and G. Schmidt (Eds) Handbook on Enterprise Architecture. Berlin : Springer Verlag, 2003.

[6] IFIP-IFAC-Taskforce, "GERAM: Generalised Enterprise Reference Architecture and Methodology (1999)," Version 1.6.3, 1999.

[7] ISO15704, "Industrial automation systems - Requirements for enterprise-reference architectures and methodologies.," Geneva : ISO TC184.SC5.WG1, 2000, Amd.2005.

[8] H. Kandjani and P. Bernus, "Capability Maturity Model for Collaborative Networks Based on Extended Axiomatic Design Theory," in L.M. Camarinha-Matos, A. Pereira-Klen and H. Afsarmanesh (eds), Adaptation and Value Creating Collaborative Networks, *IFIP AICT 362*, Berlin : Springer pp. 421-427, 2011.

[9] R. Prikladnicki, J. L. N. Audy, and R. Evaristo, "A reference model for global software development: findings from a case study," in Proc ICGSE'06, IEEE, pp. 18-28, 2006.

[10] C. Gershenson, Design and control of self-organizing systems. Mexico City: CopIt ArXives, 2007

[11] J. W. Melvin, "Axiomatic System Design: Chemical Mechanical Polishing Machine Case Study," Massachusetts Institute of Technology, Dept. of Mechanical Engineering, Cambridge : MIT, 2003.

[12] N. P. Suh, "Axiomatic design: advances and applications, 2001," ed: New York : Oxford University Press, 2001.

[13] S. Lloyd, "Measures of complexity: a nonexhaustive list," *IEEE Control Systems Magazine*, vol. 21, pp. 7-8, 2001.

[14] H. Kandjani and P. Bernus, "Engineering Self-Designing Enterprises as Complex Systems Using Extended Axiomatic Design Theory.," in Bittanti, S., Cenedese, A., Zampieri, S. (Eds) Proc IFAC 2011, *IFAC Papers On Line*, vol 18. Amsterdam : Elsevier. pp.11943-11948, 2011.

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

[16] N. Suh and S. Do, "Axiomatic design of software systems," CIRP Annals-Manufacturing Technology, vol. 49, pp. 95-100, 2000.

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

[18] R. Sangwan, N. Mullick, and M. Bass, Global Software Development Handbook: CRC Press, 2006.

[19] W. R. Ashby, "An introduction to cybernetics," London, Chapman & Hall, 1957.