Revisiting Naur's Programming as Theory Building for Enterprise Architecture Modelling

Balbir S. Barn and Tony Clark

Middlesex University, Hendon, London, UK, NW4 4BT

Abstract. The recent burgeoning interest in Enterprise Architecture and its focus on artifact driven methods is taken as a motivation for the re-appraisal of Peter Naur's notion of "programming as theory building". Naur strongly disputes the value of the role and orientation of the IS discipline around artifacts and argues that algorithmic methods do not lead to a theory of a domain. Such a viewpoint provides an alternative lens with which to view current developments and may lead to additional insights by providing the reader with a source for questioning and reflecting critically on the re-focusing of method design on conversation rather than artifact production . It is suggested that such a conversational framework based on Toulmin and Pask may provide a means to establish and test theory building views of enterprise architecture.

1 Introduction

This account sets out to re-appraise Peter Naur's influential paper on Programming as Theory Building [11] in the context of model building and the recent focus on Enterprise Architecture. It is the intention of this paper to evaluate how theory building can play an important role in helping organizations make more use of their enterprise architecture activity and in particular how theory building may influence methods, techniques and tools to support enterprise architecture by focusing on conceptual modelling as a conversation process.

The starting point for this work has been triggered by the extent of activity that is currently surrounding Enterprise Architecture. As systems supporting business become increasingly more significant and complex an important approach to management and planning of systems that has gained prominence is model-based Enterprise Architecture (EA). EA has its origins in Zachman's original EA framework [21] while other leading examples include the Open Group Architecture Framework (TOGAF) [17] and the framework promulgated by the Department of Defense (DoDAF) [19]. In addition to frameworks that describe the nature of models required for EA, modeling languages specifically designed for EA have also emerged. One leading architecture modelling language is Archimate [7].

Central to enterprise architecture is the notion of development and presentation of models. Given the plethora of models available two concerns of note arise: Firstly, given the range of models available, it is difficult to ascertain why

a particular model is relevant and preferable over others. This arises from a lack of clarity of the link between the contents and structure of a model on one hand and its purpose on the other [4]. Secondly, evaluation of quality of models in general, and therefore EA models, is relatively under researched. While there are international standards for software systems there is "little agreement among experts as to what makes a "good" model" [8]. Empirical measurements of the goodness of an EA model are generally lacking in the literature. Is that because we want to evaluate the final outputs of the modelling process rather than the success of the modelling process? Would a re-appraisal of Naur's ideas will provide a new insight and approach to questioning the "why" of a model? Similarly, would a re-appraisal provide insight to the "goodness" of a model for assessing the efficacy of a model in representing knowledge of a domain? These questions are the subject of this paper.

The remainder of the paper is structured as follows: Section 2 outlines the main hypotheses posed by Naur. Section 3 and 4 presents a more detailed analysis of aspects of the hypotheses (programs as models and methods). Section 5 provides an alternative view of how methods for EA should structured and presents two underlying philosophical and psychological theories and their conceptual integration as the basis for a conversation framework approach to EA method design rather than the algorithmic artifact oriented views that are currently prevalent. Section 6 concludes with an overview of the implications arising from this re-appraisal of Naur's seminal paper in the context of conversational processes for EA modelling.

2 Programming as Theory Building

Peter Naur wrote "Programming as Theory Building" in 1985, it was reprinted later in his collection of works, Computing: A Human Activity in 1992 [10]. The paper presents a discussion that contributes to what programming is. While it is tempting to assume from the title of the paper that Naur is focused on the minutiae of programming, he is specific in that programming denotes the "whole activity of design and implementation" and thus his theory applies to the field of software engineering. The fundamental premise asserts that programming should be regarded as an activity by which programmers achieve a certain insight or theory of some aspect of the domain that they are addressing. The knowledge, insight or theory that the programmer has come into possession of is a theory in the sense of Ryle [15]. That is, a person who has a theory knows how to do certain things and can support the actual doing with explanations, justifications and responses to queries. That insight or theory is primarily one of building up a certain kind of knowledge that is intrinsic to the programmer whilst any auxillary documentation remains a secondary product. Of particular interest, is how Naur explains the life-cycle of a program. Programs are created by the establishment of a theory, the maintenance of a program is dependent on the theory being transferred between programmers; and the program dies when the theory has decayed. Program revival is described as re-establishing the

theory behind the program which cannot be done merely from documentation and should only be considered in exceptional circumstances as the cost of theory revival is prohibitive and the resulting theory may be different from that originally conceived.

In addition to the theory view of a program life cycle, he also directed criticism at the then significant emphasis of methods for program development. He claimed that the methods based on sequences of actions of certain kinds cannot lead to the development of a theory of the program because the intrinsic knowledge held by a human has no inherent division into parts nor an inherent ordering. Instead the person possessing the knowledge is able to present multiple viewpoints as responses to requests. Where methods were supplemented with notations or formalizations then these were treated as secondary items as the theory of a program is intrinsic and cannot be expressed. Thus: "...there can be no right method".

Having outlined the basic hypotheses of Naur's paper, the remainder of this account continues the critique of Naur's ideas and applies them to modeling and Enterprise Architecture.

3 Programs as models

Naur was concerned with programs, but Enterprise Architecture is concerned with the production of models of interconnected systems or components. Thus we need to explore the relationship between programs and models and use that as a basis for analysing the applicability of Naur's hypothesis in the context of Enterprise Architecture.

A major activity in software engineering and computer science in general is modelling and as Fetzer [1] has noted "the role of models in computer science appears to be even more pervasive than has been generally acknowledged..". A key feature of modelling is the existence of an isomorphic relationship between the parts of the model and the parts of the thing modelled at some level of abstraction. Smith [16] whilst noting these different types of models emphasizes the nature and importance of "representation":

> "To build a model is to conceive of the world in a certain delimited way... Computers have a special dependence on these models: you write an explicit description of the model down inside the computer...".

Smith suggests this feature distinguishes computers from other machines because they run by manipulating representations. "Thus there is no computation without representation" [16, p, 360] If we pursue this analysis further: From Naur we can state that the program is a theory; from general computation principles, we can state: the program is a model. This leads to the notion that there is an equivalence between program = theory = model. We might moderate this further by noting that a program is a representation of a slice of "the" theory. In general though, this blurring between programs, theories and models is confusing and inaccurate. While models may exhibit an isomorphic relationship with

their subject matter, this relationship may not reveal the theoretical connections that allow the theory to be defended in the form of Ryle's definition of a theory. Ideally, then, the theory must be statable independent of the computer model. In an essay that predates Naur's paper but still based on a prevailing view of the time that programs are theories, James Moor notes:

> "My claim is that this is rarely, if ever, the correct response. Even if there is some theory behind a model, it cannot be obtained by simply examining the computer program. The program will be a collection of instructions which are not true or false, but the theory will be a collection of statements which are true or false. Thus, the program must be interpreted in order to generate a theory. Abstracting a theory from the program is not a simple matter for different groupings of the program can generate different theories. Therefore, to the extent that a program, understood as a model, embodies one theory it may well embody many theories."[9, p.221]

From this analysis arises two key concerns. Firstly, programs and models may have multiple theories and a program or model may not refer to the same theory. Secondly these theories must be state-able independent of the program or the model. Then, there is an additional dichotomy: Is a program a representation of one view of an aggregate theory or is the program a representation of a component theory of the aggregate theory? These complexities, in the case of Naur's Programming as theory perspective have implications, because if the program is the only vehicle through which a theory can demonstrate that requirements of the intended system have been met, then, that theory testing process comes too late in the system life cycle.

4 On methods and theory building

Earlier we noted that Naur had reserved considerable criticism for methods. We develop this discussion further in this section. The tendency of methods research in the IS discipline is to propose algorithmic steps to analysing and designing solutions to problems. As Naur notes: "A method implies a claim that program development can and should proceed as a sequence of actions leading to a particular kind of documented result". In contrast, a theory building view holds that a theory "held by a person has no inherent division into parts and no inherent ordering". At large, IS/SE research is embarked on a journey based on epistemological foundations and as a consequence has mostly neglected *techne* (the technical know how of getting things done) and *phronosis* (wisdom derived from socialised practices) [20]. In a more generalised form, this has correspondence to the distinctions between explicit and tacit knowledge [12] and Naur would seem to be arguing the case for methods research that suggests more attunement with the effects that methods may have in the education of programmers. That is, the creation and embedding of tacit knowledge rather than the production of artifacts representing explicit knowledge through an algorithmic process.

Naur cites a study of five different methods by Floyd et al (cited here for completeness [2])where the key result that a system of rules will lead to good solutions is an illusion, what remains is *the effect of methods on the education of programmers*. Thus the use of methods may themselves not lead to a good design but the practice of the method may lead to a better innate ability for theory building.

5 Theory building and testing as a conversation process

The act of constructing a conceptual model that describes an enterprise architecture is essentially all about communication. For example, when we engage in a discussion of budgetary requirements, we are requiring the architecture description to provide us with a theory of budgetary models. A description of the communication of how that theory is explicated is at the heart of that architecture description. In a modelling process, participants such as the domain expert and the systems analyst (who may have no knowledge of the domain) engage in a conversational process through which concepts understood by the domain expert are formalised by the systems analyst through some dialogue document in a controlled language[3]. The goal of the modelling process is to reach a state where all participants agree that they have some degree of common understanding[14].

When Naur describes theory building amongst teams of programmers who share the same theory he would appear to be alluding to a similar socialisation process. More recently and in line with what we propose in this section, Kruchten [5] provides a critique of software architecture from a knowledge management perspective where architectural knowledge is a composite of the architecture (design) and a rationale for design decisions. The support for the rationale comes through a socialisation process framed by the SECI (socialisation, externalisation, combination, internalisation), model [12]. Significantly, though, the artifact remains central albeit augmented by more human centred activity.

In the development of a theory, Naur also suggests three tests to check if the programmers knowledge transcends the written documentation consistent with Ryle's notions that a theory should be defensible and justifiable by the presentation of evidence. These are: the programmer can explain how the solution relates to the affairs of the worlds that it helps to handle; the programmer can explain why each part of the program is what it is, in other words, is able to support the actual program text with a justification of some sort; and the programmer is able to respond constructively to any demand for a modification.

Here we propose that the first test can be addressed by consideration of an integration of two other philosophical theories in this field: Toulmin's informal argumentation model [18] and Pask's conversation theory [13] and to suggest that theory testing can be achieved by constrained conversations using models as the subject. More pertinently, it may help us to address the "why" of an enterprise architecture.

5.1 Conversation Theory

A conceptual model represents the arrival of a shared understanding of a subject area between two different actors – the domain expert and the systems designer. One way of viewing the process of understanding is through the lens of Conversation Theory (CT) [13]. As theory of exposition and defence, CT can be summarised as follows: one participant (say, the domain expert) describes a body of knowledge to a second participant (the Systems Analyst). Both these participants are a type of organization – the psychological (p-) individual. A p-individual is a stable closed system comprising memory (facts), rules for interpreting the memory (concepts), rules for structuring the derivation of concepts – "how to" understand concepts and rules for understanding how topics in the memory relate to each other. In a basic conversation ("skeleton of a conversation"), there are two levels – the "how" and "why". The "how" level describes how to do a topic for example, recognizing, constructing and maintaining a topic, while the "why" level is focused on explaining or justifying the topic perhaps in terms of other topics. The basic conversation is provocative, that is participants are provoked into constructing understandings of each others' beliefs. A "modeling facility" provides the medium in which concepts are understood between individuals.

A key aspect of CT is the embodiment of knowledg (e.g. the workings of the combustion engine, finite state machines or any other coherent whole) which is viewed as a set of topics or facts that are related to each other. Relations between topics are either decompositional (hierarchical) or analogous (heterarchical), when such relationships and topics are static then that static representation is called an entailment structure. When a topic is understood by a learner (via a reproducible procedure) then the topic also exists as a concept for potential sharing with another p-individual.

5.2 Argumentation Theory

A person who has or possesses a theory knows how to do certain things and can support those actions with explanations, justifications and answers to queries. This is similar to Toulmin's argumentation model [18] - a logical structure for reasoning about the validity of arguments, the structure of which are described below:

Claim A proposition representing a claim being made in an argument;
Grounds One or more propositions acting as evidence justifying the Claim;
Warrant One or more rules of inference describing how the Grounds contribute to the Claim;
Backing The knowledge establishing the Grounds for believing the Warrant;
Qualifier A phrase qualifying the degree of certainty in the argument for the Claim;
Rebuttal One or more propositions challenging the validity of the Claim.

An example of a Toulmin argumentation model might be as follows: Object oriented modelling is a more natural way for most business analysts to capture requirements. Such a statement is a claim that includes a qualifier - most. The grounds for this statement might refer to hard facts or evidence that supports this claim. The warrant might indicate how object concepts provide a closer correspondence to objects in the real world. The backing for the claim might be: because object modeling is derived from entity modeling and entity relationship modeling has considerable history of efficacy in requirements capture. A rebuttal is a counter claim and has its own argumentation model.

Taken together, the two theories present a potential opportunity to review how we design methods and their supporting tools. The argumentation model presents a conversational framework which allows the theory builder to create an orderly and intelligible conversation - a discussion of the theory. But because such discourse analysis has the potential to generate large amounts of data by utilizing a limited set of concepts derived from the domain (the topics in the entailment mesh from conversation theory) it is possible to make the resulting analysis more amenable.

6 Implications for Enterprise Architecture

Enterprise Architecture (unlike programming) has no target theory. The execution of a program can be used to validate the quality of the theory that a programmer constructed but mechanisms for executing enterprise architectures are still largely an area of research focus. Prevailing methods and languages for EA (and using ArchiMate as a canonical example) have focused on developing artifacts and models for explicit knowledge [6, p.75] and so are subject to Naur's criticisms. EA frameworks such as TOGAF provides an exhaustive set of activities, phases of activities, ordering of activities and artifacts to be produced by activities with the intent of capturing in its entirety a theory of the EA. Accepting the theory building view forces us to reject firstly that such an exhaustive methodological approach can lead us to a universal theory of Enterprise Architecture for a domain. Secondly the focus on explicit knowledge does not allow us to extract from the plethora of method the essence of "why". Instead, a model of incremental, modular theory building which involves the real world thorugh a conversational process as a source of knowledge and validation may unlock the real value of an enterprise architecture.

This takes us then to a more fundamental re-thinking of method development. A method for EA should not (algorithmically) take us to a model of EA (because no one model exists), instead a method should instill in the practitioner, the cognitive processes for constructing theories about the enterprise architecture. The conversational approach outlined earlier is one such candidate basis for such cognitive processes as it enables both the testing of a theory and the collaborative development of a theory. Indeed it might allow us to measure the efficacy of a method not by how a solution is designed or quality of solution but by how the

engineer has modified their psychological processes for theory building and so the corresponding implications for software engineering education.

References

1. J.H. Fetzer. The role of models in computer science. *The Monist*, 82(1):20–36, 1999.
2. Christiane Floyd. Eine untersuchung von software-entwicklings-methoden. In H. Morgenbrod, W. Sammer, and I. Tagung, editors, *Programmierumgebugnen und Compiler*. Tuebner Verlag, 1984.
3. S. Hoppenbrouwers, HA Proper, and T.P. der Weide. A fundamental view on the process of conceptual modeling. *Conceptual Modeling–ER 2005*, pages 128–143, 2005.
4. P. Johnson, M. Ekstedt, E. Silva, and L. Plazaola. Using enterprise architecture for cio decision-making: On the importance of theory. In *the Proceedings of the 2nd Annual Conference on Systems Engineering Research (CSER)*, 2004.
5. P. Kruchten. Documentation of Software Architecture from a Knowledge Management Perspective–Design Representation. *Software Architecture Knowledge Management*, pages 39–57.
6. M. Lankhorst. *Enterprise architecture at work: Modelling, communication and analysis*. Springer-Verlag New York Inc, 2009.
7. M. M Lankhorst, H.A H.A. Proper, and J. Jonkers. The Anatomy of the ArchiMate Language. *International Journal of Information System Modeling and Design*, 1(1).
8. D.L. Moody. Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, 55(3):243–276, 2005.
9. J.H. Moor. Three myths of computer science. *British Journal for the Philosophy of Science*, 29(3):213–222, 1978.
10. P. Naur. *Computing: a human activity*. ACM New York, NY, USA, 1992.
11. Peter Naur. Programming as theory building. *Microprocessing and Microprogramming*, 15(5):253 – 261, 1985.
12. I. Nonaka and H. Takeuchi. The knowledge-creating company. *New York*, 1, 1995.
13. G. Pask. *Conversation, cognition and learning*. Amsterdam and New York: Elsevier, 1975.
14. K. Pohl. The three dimensions of requirements engineering: A framework and its applications* 1. *Information systems*, 19(3):243–258, 1994.
15. G. Ryle. The concept ofmind. *London, Hutchinson*, 1949.
16. B.C. Smith. *Limits of correctness in computers*. Academic Press Professional, Inc., 1991.
17. J. Spencer et al. *TOGAF Enterprise Edition Version 8.1*. 2004.
18. S.E. Toulmin. *The uses of argument*. Cambridge Univ Pr, 2003.
19. DE Wisnosky and J. Vogel. DoDAF Wizdom: A Practical Guide to Planning, Managing and Executing Projects to Build Enterprise Architectures Using the Department of Defense Architecture Framework (DoDAF), 2004.
20. Boris Wyssusek. A philosophical re-appraisal of peter naur's notion of "programming as theory building". In *European Conference on Information Systems (ECIS)*, 2007.
21. J.A. Zachman. A framework for information systems architecture. *IBM systems journal*, 38(2/3):454–470, 1999.