December 2001

A Naive Philosophical Journal, or an Essay on at Least One Limit of Abstraction

Lisa Murphy
Louisiana State University

A NAIVE PHILOSOPHICAL JOURNEY, OR AN ESSAY ON AT LEAST ONE LIMIT OF ABSTRACTION

Lisa D. Murphy
Louisiana State University
ldmurph@lsu.edu

Abstract

In which a recent encounter with an aircraft engineer results in recognition of the power of naïve philosophy to bound conceptions of systems problems and the nature of their solutions, thereby revealing the limits of a familiar and well-respected tool.

Introduction

This essay considers the philosophical origins of a design problem I recently encountered in the field. It is a naïve philosophical journey in the sense of returning to origins, to the unconscious assumptions of those not schooled in philosophical foundations for their world view. Naïve philosophies guide the perceptions of people – users, customers, managers – as they work and live, and thus as they engage with information systems (IS) and information technology (IT). These working philosophies enable, constrain, or otherwise influence the task of systems development including the conceptions of suitable problems and satisfactory solutions (Checkland, 1993; Wand, Monarchi, Parsons, and Woo, 1995), and as such, are not just worthy of our attention, but should demand it.

My motivations here are to clarify my own thinking, to share an interesting problem with colleagues, to illustrate the power of users' naive philosophies to effect systems development, and with luck, to generate productive guidance for my informant. To this end, I use a reflective essay presentation; one way to think of this essay is as an action-research in progress in which the researcher reflects upon his failure. Below, I briefly argue that aircraft engineering is characterized by a positivist ontology. I then present a simplified version of a current, complex case from this field. In closing, I reflect on the implications of users' philosophies-in-practice for their conceptions of problems, and therefore systems developers' conceptions of both problems and solutions.

Engineering Design Abstracts Reality

The domain of this systems design problem is aircraft engineering. Vincenti (1990) describes an airplane as one of a class of "devices that constitute complex systems" (p. 9) and argues that the multilevel and hierarchical nature of aircraft design has consequences for the nature of the engineering effort and knowledge. His presentation and analysis of five case studies from aeronautical engineering (e.g., propeller design, flush riveting) is consistent with Checkland's (1993) description of traditional engineering approaches to systems as having "a built-in positivist ontology" (p. 277). Indeed, at one point Vincenti argues that the insights of socio-technical systems researchers such as Bijker, Hughes and Pinch are interesting but "the social process has little effect on the problem or the solution" (p. 312 n. 9). Thus we expect engineering-related information systems and technologies to reflect a positivist orientation: Successful systems will abstract a physical reality (as seen in the assumptions and advice of Marks and Riley, 1995, in their guidebook for choosing and justifying engineering and manufacturing technologies). Many readers will recognize that this basic philosophy underlies many aspects of IS and IT, not just engineering systems (e.g., Wand et al., 1995). Indeed, therein lies the power of abstraction: manipulating electrons or light pulses is less expensive, time-consuming, and risky than manipulating aluminum, carbon fiber, or human flesh.

Matthew Gets Stuck

My informant engineers aircraft. (Details are disguised to protect the identities of Matthew and his firm.) Historically, aerospace firms developed a single new aircraft design at one time; the process is long, risky, and expensive, typically taking five years and a billion dollars. Beginning with early NASA-sponsored efforts, aerospace engineering has aggressively exploited IT, particularly for mathematical applications (e.g., Vincenti, 1990). To many, engineering technology is nearly synonymous with Computer-Aided Design (CAD) (Henderson, 1998). Perhaps even more so in aerospace due to stories such as the 1996 book and film *Twenty-First Century Jet* (Sabbagh) about Boeing's use of advanced IT for engineering the 777. Undoubtedly, CAD is important for design, but less significant for many aircraft engineering specialties than finite element modeling, computational fluid dynamic models, or other mathematically-intensive tools (e.g., Vincenti,1990; Marks and Riley, 1995). Those new to aerospace may not realize that CAD electronic drawings receive more attention from functions outside engineering than the outputs of these specialized analytical tools. Indeed, the "downstream" dependence on engineering drawings by Tooling, Materiel, Planning, Manufacturing, Quality Assurance, and so forth, has led the development and implementation of a class of IS called Product Data Management systems (PDM). PDM systems take the aircraft as its object in the world to model (Wand et al., 1995) by controlling the creation of part numbers and the authorization to use part definition data by business rules (e.g., a part is not "released" until a predetermined set of engineering drawings and documents exist and are approved). From an IT perspective, PDM systems are large databases with heterogeneous content (e.g., CAD files, word documents, machining data sets) and some workflow attributes; they may interface with manufacturing systems such as MRP and provide data for numerical control programming and other activities that constitute that model into a physical thing (Marks and Riley, 1995).

The presented problem: Matthew's company would like to reduce time-to-market for new aircraft development. In the past, process improvements were difficult to identify for these infrequent, large, multi-year efforts involving hundreds of engineers from dozens of specialties. Anticipating overlapping development projects for the next several years, management has tasked Matthew to apply process improvement and advanced IT concepts to new aircraft engineering. Enter the researcher, who while studying engineering documents ends up at Matthew's door; a 60-minute interview lasts for several hours, and in the end, it isn't at all clear who is being interviewed. Matthew says he is "stuck"; after collating an impressive set of data on the timing and nature of engineering effort on the last three projects, he is hoping *I* have some answers for him.

Where he got stuck: Matthew has taken a typical engineering (read: rational) approach and found a potential duplication of effort not due to rework and apparently not amenable to collaborative technologies. Individual engineering specialties (e.g., aerodynamics, flutter analysis, structural integrity) create definitions for use in their own analytical applications; sharing work-in-progress data for major aircraft components such as the tail across these specialties early in the design process could reduce the engineering equivalent to manufacturing set up costs for the definition of the analytical models. Aye, but here's the rub: Not only do the specializations use different types of data (e.g., structures use matrices of stress and materials properties, aerodynamics and flutter use vector geometry) in different formats and with different tools, they use different scope and scale in their definition (e.g., flutter needs much less detail in the surface geometry but greater coverage than aerodynamics does; structure needs interior definition that aero and flutter do not). Ultimately these differences are resolved in favor of the detail necessary for manufacturing; can or should we presume the a *posteriori* level of abstraction *a priori*?

Aircraft engineering projects, even when they represent what Vincenti (1990) calls "normal design" – that is, the application of familiar engineering knowledge to incrementally varying cases (cf. "radical design"), are characterized by complexity. A brilliant design concept is not enough to ensure no-revision engineering, and a purely linear model of aircraft development is not possible or practical as decisions make in one specialty on one part of the aircraft affect other decisions and their justification (Vincenti).

An oft promoted approach, concurrent engineering (e.g. Loch and Terwiesch, 1998), relies primarily on communication and information exchange mechanisms to facilitate interaction between project sub-teams to alert one group when assumptions shift in another area. Tools for collaborative engineering include paper and pencil, white boards, and interactive CAD displays (Henderson, 1998; Suchman, 1987). However, a new aircraft project is well beyond the twenty-person-or-less core design team recommended by Marks and Riley (1995) for collaborative design with computer-aided engineering systems. In my experience these utopian models of concurrent and collaborative engineering tend to resolve in practice into a lunging-and-lurching parallelism in which subgroups feel continuous pressure to avoid holding other subgroups back from deadlines while the different specialties sub-optimize for their own area as long as possible.

Philosophy as Driver

Systems developers bring a set of experiences and methods to a project (a problem), guided by principles reflecting a philosophical approach (Orlikowski & Baroudi, 1991). Systems methodologies abound, but there is an orthodoxy which represents a generally positivist approach to "modeling" the world of the problem and its domain similar to the philosophy of aircraft engineering proposed by Vincenti (1990). For example, Jackson (1995) argues that conventional systems development approaches start at

the end by asking: What is the desired result? The system is then created by working backward from the end result to build a system which maps to a process or product in the world.

Commonly conceived the desired result of a new aircraft engineering effort is a product definition (now three dimensional and electronic rather than two-dimensional on paper) which controls the (re)creation of that definition in physical form, as a specific aircraft unit. Indeed, the engineering drawings are so significant when looked at for the business as a whole that is easy to forget that the "real" engineering outcome is an FAA "Type Certificate", i.e., authorization from the FAA to sell the aircraft to customers who want to fly it. Drawings, material specifications, QA plans, etc., are what you need to manufacture an aircraft …once you certify it. The visible evidence – the tip of an informational iceberg -- upon which FAA certification is granted is a relatively small number of technical reports (typically a few hundred compared to tens of thousands of drawing documents). However, neither technical reports nor engineering drawings alone are sufficient to describe the information needed to certify; yet, there is little direct link between the two forms of information in the systems that propose to contain the engineering definition of the aircraft. Specifically, PDM systems contain no reference to the certification reports which authorize the use of the design information they contain. Thus, the visible "end" of aircraft certification is the content in the PDM system; the less visible end is the content of the certification reports, which have little role in downstream business operations.

As we discuss ideas for solving his "avoid the set-up" problem, it proves difficult for both Matthew and I to resist trying to extend the concept of product definition (embodied as a solution in PDM systems) backward into the new aircraft development project. This is occurring even as I struggle (and succeed) to hold my tongue about the major frustrations and failures I had been hearing from a contact at one of Matthew's competitors who has been trying to do just that. Somehow, reifying the design *before it is designed* is exceedingly tempting. At what point in the engineering process do we know what is the object to model in the PDM? Grizzled veterans of manufacturing planning might say "never"; today, revisions due to attempted concurrent engineering and the exigencies of customer demand result in nearly continuous design changes. In their simulation analysis of concurrent engineering, Loch and Terwiesch (1998) treat design knowledge as uncertain, i.e., as the average rate of engineering changes, which itself reduces over time (lowering the uncertainty). But, don't you have to have a design before you can have a design change? Is a general pattern of increasing definition (certainty) an appropriate model for complex products in which ambiguity or equivocality may be a bigger source of design risk and where engineering drawing changes only indirectly relate to certification (which must at least be modeled as a sub-goal to product definition)?

The next ring Matthew and I grabbed for was to model the process. Vincenti's five levels of design from project definition down to highly specific design problems makes this sound practical: "Such successive division resolves the airplane problem into smaller manageable sub-problems" (1990, p. 9). Existing engineering project planning reflects the assumptions of this hierarchy; it is unclear that a new aircraft with a unique combination of normal design specifications (if the combination isn't unique, why do you need a new aircraft?) would benefit from "paving the cowpaths" by standardizing on processes based on a very small *n* (three, in Matthew's case) (leaving aside radical or revolutionary designs as not central to his problem). As we talk, our conversation reminds me of a story I had heard about a chief engineer, who, anxious to align with commercial TQM-like initiatives after decades on military projects, is prone to ask his managers "why can't we all wash our socks the same way?" An answer comes to mind: if you use the same processes, you must be solving the same problem; but this is a new aircraft, so something must be different – but how much, where, and when? Process replication under these circumstances has a high chance of institutionalizing mediocrity which inhibits achieving the goal -- a *new* product. Vincenti argues that even "normal technology" designs can have a radical engineering issue lurking behind an innocuous-looking task; inappropriate process emphasis may impede recognition that there is, indeed, new knowledge required.

Abstraction Fails

As Matthew and I talked, I became aware of the limits imposed on our thinking by the naïve (that is, unconscious) philosophy which values positivist-like abstraction of reality into IS/IT. We cannot deny the power of abstraction: CAD abstracts part geometry, finite element models abstract stress, computational fluid dynamic models abstract flows, and so on. Engineering a new aircraft can be characterized as a process of increasing the "concreteness" of an abstraction: payload, speed, price, and mission profiles become cruise speeds, seating options, wing geometry, and material specifications. Similarly, Vincenti (1990) describes a hierarchy of systems: devices become an aircraft system which itself becomes a device in a system of moving passengers; the process of making an aircraft is an abstraction of a particular aircraft unit with certain passengers flying to a particular destination in stormy weather at a 32,000 ft. Each abstraction privileges some point of view (e.g., types of information) more than another, yet without challenging the underlying positivist philosophy. Does aircraft engineering rely on a special positivism in which the object-in-the-world doesn't have to be a real object in the world?

Yet, PDM systems show a high degree of positivism. While they contain some process information (e.g., numerical control data sets, methods for non-destructive testing), they are characterized by abstract representations of the parts of the aircraft (part geometry, material specifications, dimensions) and their relationships to each other (e.g., assembly drawings). These relationships

are not symbolic ones (such as currency-to-money), but nearly direct mappings (e.g., fully defined three-dimensional drawings that map directly to physical attributes of parts-as-made) with legal and practical requirements that treat the drawing as equivalent to the part.

Also, the specialized tools of the flutter, loads, and structural integrity engineers are also positivist – each asserts a rendering of some aspect of aircraft behavior as an adequate model of their particular part of reality. (Indeed, the "validation" of the tools must be addressed in the FAA certification reports by citing evidence that they capture the variation in empirical test data.) Not surprisingly, Matthew tried to apply a positivist model to the integration of these tools; did he mis-apply the philosophy? A reviewer argues that because an abstraction may abstract another abstraction, this "removes all the mystery" of this issue: "an object-in-the-world may be a non-object (or as the author terms it, not a real object) – no problem!" By this argument, Matthew's solution would lie in correctly defining the data required for the different specialties as a hierarchy of abstractions, built upon some common denominator, e.g., if flutter requires less detail than loads, then a flutter dataset for the tail is treated as n+1 level of abstraction over a load dataset for the tail. This implies that the lower you go in the hierarchy, the more closely it maps to the object-in-the-world – that is, the less abstract it is. However, this assumption/solution inverts the process of aircraft engineering which produces less and less abstract definitions beginning with a very abstract one – the bottom of the hierarchy is more abstract, not less. The question, to a great extent is: "what is the tail", not "how much detail do we know about the tail?" Thus, the issue remains in that systems development approaches which rely on abstracting reality may offer little help when the goal is reduced abstraction – removing equivocality, not just uncertainty.

A Journey to Abstraction and Beyond

Although I have a dozen years of systems development experience before moving to academia, this was the first time I encountered a problem so resistant to my most trusted systems analysis tool, abstraction (Wand et al., 1995). Yet, I don't think Matthew and I were falling into a trap of abstracting the problem from its domain (Jackson, 1995, p. 139); the domain of engineering includes the widest realm of specialized software I have encountered, and Matthew's conception includes them in the problem frame. However, this problem derives from the absence of an object to abstract per se; how can the emerging definition of the object *be* the object that itself is being modeled? As a systems developer, I cannot insist my users change or abandon their own philosophy-in-practice because I find it inconvenient; indeed, the need to accommodate the many voices of users has informed many discussions such as those found in this forum. My journey takes me to this place: as the implicit and explicit philosophies of systems development representations encounter the lived philosophy of our users, aren't we privileging our world view over theirs if we don't recognize their philosophy-in-practice, even as that naïve philosophy bounds their choices?

N.B.: While writing this paper, I called Matthew and asked if he had any training in philosophy. "No," he said, he finds his understanding of the world in his work and his religion.

References

Checkland, Peter. *Systems Thinking, Systems Practice.* Chichester, UK: John Wiley & Sons Ltd, 1993.

Henderson, Kathryn. *On Line and On Paper: Visual Representations, Visual Culture, and Computer Graphics in Design Engineering.* Cambridge, MA: MIT Press, 1999.

Jackson, Michael. *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices.* ACM Press, 1995.

Loch, Christoph and Christian Terwiesch. " Communication and uncertainty in concurrent engineering," *Management Science*, 44 (8), 1998, pp. 1032-1048.

Marks, Peter and Kathleen Riley. *Aligning Technology for Best Business Results*: *A Guide for Selecting and Implementing Computer-aided Design and Manufacturing Tools.* Santa Cruz, CA: Design Insight, 1995.

Orlikowski, Wanda and Jack Baroudi. "Studying information technology in organizations: Research approaches and assumptions," *Information Systems Research,* 2, 1991, 28.

Sabbagh, Karl. *Twenty-First-Century Jet, The Making and Marketing of the Boeing 777*. New York, NY: Simon and Schuster, 1996.

Suchman, Lucy. *Plans and Situated Actions: The Problem of Human-Machine Communication.* Cambridge University Press, 1997.

Vincenti, Walter G. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History.* Baltimore, MD: Johns Hopkins University Press, 1990.

Wand, Yair, Monarchi, David, Parsons, Jeffrey, and Woo, Carson. "Theoretical foundations for conceptual modeling in information systems development," *Decision Support Systems,* 15, 1995, pp. 285-304.