

1

The Design Question

[New ideas would come about] by a connexion and transferring of the observations of one Arte, to the uses of another, when the experience of several misteries shall fall under consideration of one mans minde.

SIR FRANCIS BACON [1605], *THE TWO BOOKS OF THE PROFICIENCE AND ADVANCEMENT OF LEARNING*, BOOK 2, 10

Few engineers and composers ... can carry on a mutually rewarding conversation about the content of the other's professional work. What I am suggesting is that they can carry on such a conversation about design, . . . [and then] begin to share their experiences of the creative professional design process.

HERBERT SIMON [1969], *THE SCIENCES OF THE ARTIFICIAL*, 82

Spiral staircase

Corbis

Is Bacon Right?

Sir Francis Bacon's hypothesis is our challenge. Are there invariant properties of the design process itself, properties that hold across a wide range of media of design? If so, it seems likely that designers in one medium would collectively grasp some of these principles more clearly than other designers, through struggles that are peculiarly difficult for that medium. Moreover, some media, such as buildings, have longer histories of both design and meta-design—"the design of design." If all this is true—and if Bacon's conclusion is true—designers in different media can expect to learn new things about their own several crafts by comparing their experiences and insights.

What Is Design?

The *Oxford English Dictionary* defines the verb *design* as

To form a plan or scheme of, to arrange or conceive in the mind . . . for later execution.

The essentials of this definition are *plan, in the mind, and later execution*. Thus, a design (noun) is a created object, preliminary to and related to the thing being designed, but distinct from it. Dorothy Sayers, the English writer and dramatist, in her magnificent and thought-provoking book *The Mind of the Maker*, breaks the creative process out further into three distinct aspects. She calls them the Idea, the Energy (or Implementation), and the Interaction,¹ that is,

1. The formulation of the conceptual constructs
2. Implementation in real media
3. Interactivity with users in real uses

A book, in this conception, or a computer, or a program, comes into existence first as an ideal construct, built outside time

and space, but complete in essence in the mind of the author. It is implemented in time and space, by pen, ink, and paper; or by silicon and metal. The creation is complete when someone reads the book, uses the computer, or runs the program, thereby interacting with the mind of the maker.

In an earlier paper, I divided the tasks in building software into *essence* and *accident*.² (This Aristotelian language is not to denigrate the accidental parts of software construction. In modern language the terms would more understandably be *essential* and *incidental*.) The part of software building I called *essence* is the mental crafting of the conceptual construct; the part I called *accident* is its implementation process. *Interaction*, Sayers's third step, occurs when the software is used.

The design is thus the mental formulation, which Sayers calls "the Idea," and it can be complete before any realization is begun. Mozart's response to his father's inquiry about an opera due to the duke in three weeks both stuns us and clarifies the concept:

Everything has been composed, just not yet written down.

LETTER TO LEOPOLD MOZART [1780]

For most human makers of things, the incompletenesses and inconsistencies of our ideas become clear only during implementation. Thus it is that writing, experimentation, "working out," are essential disciplines for the theoretician.

The phases of Idea, Implementation, and Interaction operate recursively. Implementation creates a space in which another cycle of design must be done. Thus Mozart Implemented his opera Idea with pen on paper. The conductor, Interacting with Mozart's creation, conceived an Idea of an interpretation, Implemented it with orchestra and singers, and the Interaction with the audience completed the process.

A *design* is a created object; associated with it is a *design process*, which I shall call *design*, without any article. Then there is the verb *to design*. The three senses are intimately related; I believe context will resolve ambiguity.

What's Real? The Design Concept

If a number of individuals have a common name, we assume them to have also a corresponding idea or form:—do you understand me?

I do.

Let us take any common instance; there are beds and tables in the world—plenty of them, are there not?

Yes.

But there are only two ideas or forms of them—one the idea of a bed, the other of a table.

True.

And the maker of either of them makes a bed or he makes a table for our use, in accordance with the idea.

PLATO, *THE REPUBLIC* [360 BC], BOOK X

At the 2008 Design Thinking Research Symposium 7, each of the speakers presented analyses of the same four design team meetings.³ Videos and transcripts had been distributed well in advance.

Rachael Luck of the University of Reading identified in the architectural conversations an entity that none of us had remarked but all then recognized: *the Design Concept*.⁴

Sure enough, both architect and client referred from time to time to this shared invisible entity. Speakers usually gestured vaguely toward the drawings when they spoke thus, but it was clear they were not referring to the drawings or any particular thing therein. Always, the concern was for the conceptual integrity of the developing design.

Luck's insight made the *Design Concept* a thing in its own right. This resonated strongly with my experience. When the IBM System/360 “mainframe” computer family's single architecture

was being developed (1961–1963), such an entity was always present in the architecture group, although never named. Exploiting Gerry Blaauw's brilliant insight, we had explicitly separated the System/360 design activities into *architecture*, *implementation*, and *realization*.⁵ The basic concept was a computer family with one face to the programmer—the architecture—and multiple concurrent implementations at various positions on the performance and price curves (Chapter 24).

The very simultaneity of multiple implementations, with their several engineering-manager champions, drove the common architecture toward generality and cleanliness and insulated it from small cost-saving compromises. These forces, however, were merely shields for the instincts and desires of the architects, who each wanted to make a clean machine.⁶

As the architecture design progressed, I observed what at first seemed quite strange. For the architecture team, the *real* System/360 was the Design Concept itself, a Platonic ideal computer. Those physical and electrical Model 50, Model 60, Model 70, and Model 90 things under construction out on the engineering floors were but Plato's shadows of the real System/360. The real System/360's most complete and faithful embodiment was not in silicon, copper, and steel, but in the prose and diagrams of *IBM System/360 Principles of Operation*, the programmer's machine-language manual.⁷

I had a similar experience with the View/360 beach house (Chapter 21). Its Design Concept came to be real long before any construction began. It persisted through many versions of drawings and cardboard models.

Interestingly enough, I never felt such a Design Concept entity of the Operating System/360 software family. Perhaps its architects did; perhaps I did not have an intimate enough acquaintance with its conceptual bones. Perhaps the Design Concept didn't emerge for me because OS/360 was in fact a fusion of four somewhat separate parts: a supervisor, a scheduler, an I/O control system, and a large package of compilers and utilities (Chapter 25).

What's the Value?

Is there positive value to recognizing an invisible Design Concept as a real entity in design conversations? I think so.

First, great designs have conceptual integrity—unity, economy, clarity. They not only work, they *delight*, as Vitruvius first articulated.⁸ We use terms such as *elegant, clean, beautiful* to talk about bridges, sonatas, circuits, bicycles, computers, and iPhones. Recognizing the Design Concept as an entity helps us to seek its integrity in our own solo designs, to work together for it in team designs, and to teach it to our youth.

Second, talking frequently about the Design Concept as such vastly aids communication within a design team. Unity of concept is the goal; it is achieved only by much conversation.

The conversation is much more direct if the Design Concept per se, rather than derivative representations or partial details, is the focus.

Thus, moviemakers use storyboards to keep their design conversations focused on the Design Concept, rather than on implementation details.

Detailing will of course surface conflicting versions of the Concept and force resolution. For instance, System/360 architecture needed a decimal datatype, as a bridging aid for thousands of existing users of IBM's decimal machines. Our developing architecture already had several datatypes, including a 32-bit fixed-point twos-complement integer and a variable-length character string.

The decimal datatype could be made similar to either one. Which choice better fit the Design Concept of System/360? Strong arguments were made each way; the strength of each depends on one's version of the Design Concept. Some of the architects had implicit Design Concepts reflecting earlier scientific computers; others' implicit concepts reflected earlier business computers. System/360 was explicitly intended to serve both kinds of applications well.

We chose to model the decimal datatype after the character-string one, the one more familiar to the largest particular user

community of the decimal datatype, IBM 1401 users. I would decide that way again.

Thinking about the Design Process

Thinking about designs has a long history, going back at least to Vitruvius (died ca. 15 BC). His *De Architectura* is the important book about design from the Classical period. Major milestones are the *Notebooks* of Leonardo da Vinci (1452–1529) and the *Four Books of Architecture* by Andrea Palladio (1508–1580).

Thinking about the design process itself is much more recent. Pahl and Beitz trace German thought from Redtenbacher in 1852, stimulated by the rise of mechanization.⁹ For me, major milestones have been Christopher Alexander's *Notes on the Synthesis of Form* (1962), Herbert Simon's *The Sciences of the Artificial* (1969), Pahl and Beitz's *Konstruktionslehre* (1977), and the founding of the Design Research Society and the starting of the journal *Design Studies* (1979).

Margolin and Buchanan [1995] is an edited collection of some 23 essays from the journal *Design Issues*, primarily design criticism and theory, with "occasional ventures into philosophical issues that bear on the understanding of design" (p. xi).

My *The Mythical Man-Month* [1975, 1995] reflects on the design process for IBM's Operating System/360, later evolved to MVS and beyond. It emphasizes the human, the team, the management aspects of that design and development project. Of particular relevance to the present work are Chapters 4–6 of those essays, which address how to achieve conceptual integrity in a team design.

Blaauw and Brooks [1997], *Computer Architecture: Concepts and Evolution*, includes extensive discussion of the design of the IBM System/360 (and System/370–390–z) architecture and the relationships of and rationales for dozens of design decisions. It doesn't treat the design process or human aspects of designing at all. But Section 1.4, which discusses criteria for goodness in computer architectural design, is indeed of particular relevance for this work.

Kinds of Design

System Design versus Artistic Design

This book is about the design of complex systems, and the viewpoint is that of the engineer, an engineer focused on utility and effectiveness but also on efficiency and elegance.

This contrasts with much of the design done by artists and writers, whose emphasis is on delight and the conveying of meaning. Architects and industrial designers, of course, fall into both camps.

Routine, Adaptive, Original Design

We often think of bridge design as one of the high arts of engineering, one where breakthroughs in concept or of technology have dramatic and highly visible cost, function, and esthetic consequences.

Well, a high fraction of all highway bridges are short, so cranking out a design for a 50-foot concrete bridge is a routine and automatable process. For short bridges, civil engineers know, and long ago codified into handbooks, the design decision tree, the constraints, and the desiderata. The same situation prevails for the design of compilers for established languages on new platforms. There are many areas of routine, automatable design.

The emphasis in this book is on original design, as opposed to the routine redesign of object after object with changed parameters, or even adaptive design, which is essentially the modification of a preceding design or object to serve new purposes.

Notes and References

1. Sayers [1941], *The Mind of the Maker*.
2. Brooks [1986], "No silver bullet."
3. McDonnell [2008], *About Designing*. This book is the edited Papers from the Design Thinking Research Symposium (DTRS7).
4. Luck [2009], "Does this compromise your design?" reprinted in McDonnell [2008], *About Designing*.

5. Blaauw and Brooks [1964], "Outline of the logical structure of System/360." Blaauw further divides Sayers's "Energy" into Implementation and Realization, a distinction I find immensely useful.
6. Janlert [1997], "The character of things," argues that designed things have character and discusses how one designs that character.
7. IBM Corp. [1964], *IBM System/360 Principles of Operation*.
8. Vitruvius [22 BC], *De Architectura*.
9. Pahl and Beitz [1984], *Engineering Design*.

- Goal
- Desiderata
- Utility function
- Constraints, especially budget (perhaps not \$ cost)
- Design tree of decisions

UNTIL ("good enough") or (time runs out)

DO another design (to improve utility function)

UNTIL design is complete

WHILE design remains feasible,
make another design decision

END WHILE

Backtrack up design tree

Explore a path not searched before

END UNTIL

END DO

Take best design

END UNTIL

2

How Engineers Think of Design—The Rational Model

... [F]or the theory of design is that general theory of search ... through large combinatorial spaces.

HERBERT SIMON [1969], *THE SCIENCES OF THE ARTIFICIAL*, 54

A Rational Model of the design process

The Model

Engineers seem to have a clear, if usually implicit, model of the process of design. It is an orderly model of an orderly process as the engineer conceives it. I shall illustrate with an example of a beach house design (sketched in Chapter 21).

Goal. First one starts with a primary goal, or objective: “One wants to build a beach house to take advantage of wind and wave at an oceanfront lot.”

Desiderata. Associated with the primary goal are a host of desiderata, or secondary objectives: “The beach house should be reinforced to withstand hurricane-force winds; it should sleep and seat at table at least 14 people; it should exploit the stunning views;” and so on.

Utility Function. One wants to optimize the design according to some utility or goodness function that weights the several desiderata as to their importance. So far as I can tell, most designers imagine the terms themselves to be linearly summed, but conceive of each goodness variable individually as not linear, but rather as curved asymptotically to saturation. For example, more window area is a desideratum, something desired in house design. But the utility added by each extra square foot of window diminishes. The same is true of electrical outlets. The utility of the windows and that of the outlets, however, seem simply to sum.

Constraints. Every design, and every optimization, is subject to constraints. Some of these are binary, either satisfied or not—“The house must be set back at least 10 feet from the lot’s side lines.” Others are more elastic, with steeply rising penalties as one approaches a limit, such as schedule constraints—one fiercely wants to have the beach house ready when warm weather comes.

Some constraints are simple, such as setback limits. Others blithely conceal terrifying complexity—“The house must satisfy all the building codes.”

Resource Allocations, Budgets, and Crucial Budgets. Many constraints take the form of a fixed resource to be allocated

among design elements. The most common is a total cost budget. But this is by no means the only such constraint, nor is it necessarily the one that most controls the designer's attention in a particular project. In the beach house floor plan, for example, the controlling commodity to be rationed was the feet (even inches) of ocean frontage. In the design of a computer architecture, the critical budget may be the bits in a control register or an instruction format, or the uses of the total memory bandwidth. When people were solving Year 2000 problems in software, working days on the schedule were the crucial allocable resource.

Design Trees. Now, so the Rational Model goes, the designer makes a design decision. Then, within the design space narrowed by that decision, he makes another.¹ At each node he could have taken one or more other paths, so one can think of the process of design as the systematic exploration of a tree-structured design space.

In this model, design is conceptually (at least) very simple. One searches the tree-structured design space, testing each option against the constraints for feasibility and choosing so as to optimize the utility function. The search algorithms are well known and can be cleanly described.

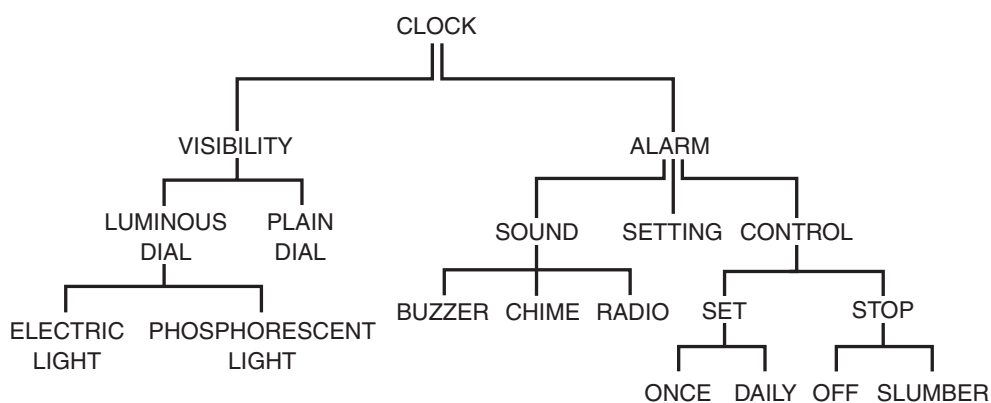


Figure 2-1 Portion of a design tree for an alarm clock
 From Blaauw and Brooks [1997], *Computer Architecture*,
 Figures 1-12, 1-14.

That cleanliness holds only for an exhaustive search of all paths, seeking a truly optimal solution. Designers commonly satisfice by searching only until a “good enough” solution is found.² Many engineers seem to approximate some sort of depth-first search strategy, choosing at each node the most promising or attractive option and exploring it to the end. At dead ends, one backtracks and takes another path. Hunches, experience, consistency, and esthetic taste guide each option selection.³

Whence Formulations of This Model?

The notion that the design process should be modeled as a systematic step-by-step process seems to have first developed in the German mechanical engineering community. Pahl and Beitz present the most widely used exposition in seven successive editions of their great work.⁴ They observe the practice, but not the explicit statement, of systematic search of design alternatives in the *Notebooks* of Leonardo da Vinci (1452–1519).

Herbert Simon independently argues for design as a search process in *The Sciences of the Artificial* [1969, 1981, 1996]. His model and his discussion of it are much more sophisticated than those here. Simon, optimistic that the design process was a fit target for artificial intelligence (once adequate processing power became available), was motivated to lay out a strictly rational model of design precisely *because* such a model was a necessary precursor to automating design. His model remains influential even if today we recognize the “wicked problem”⁵ of original design as one of the least promising candidates for AI.

In software engineering, Winston Royce, appalled at the failures of the “just write it” approach for large software systems, independently introduced a seven-step Waterfall Model to bring order to the process, as shown in the next chapter’s frontispiece. In fact, Royce introduced his waterfall as a straw man that he then argued against, but many people have cited and followed the straw man rather than his more sophisticated models. I made that mistake myself in my younger days, and publicly repented of it later.⁶ Even if ironically, Royce’s seven-step model must be

considered one of the foundational statements of the Rational Model of design.

As Royce emphasizes, his seven steps are distinctly different from one another and must be planned and staffed differently. Iteration is provided for but carefully limited in scope:

The ordering of steps is based on the following concept: that as each step progresses and the design is further detailed, there is an iteration with the [immediately] preceding and succeeding steps but rarely with the more remote steps in the sequence. ... What we have is an effective fallback position that tends to maximize the extent of early work that is salvageable and preserved.⁷

The notion that a design space can be formulated as a tree is implied by Simon. It is described and illustrated by Gerry Blaauw and me in our *Computer Architecture*.⁸ There we arrange the design choices for processor architecture strictly hierarchically in a giant tree, represented by 83 linked subtrees. A simple example of the design tree for the alarm of an alarm clock is shown in Figure 2-1. In it one observes two types of branches indicated by open and closed roots. The first, as shown for “Alarm,” shows a subdivision; each branch is a different design attribute that must be specified. This is called an *attribute branch*. The *alternative branch*, shown for “Sound,” enumerates alternatives of which one must be chosen.

What's Right with This Model?

Any systematization of the design process is a great step forward compared to “Let’s just start coding, or building.” It provides clear steps for planning a design project. It furnishes clearly definable milestones for planning a schedule and for judging progress. It suggests project organization and staffing. It helps communication within the design team, giving everyone a single vocabulary for the activities. It wonderfully helps communication between the team and its manager, and between the manager and other stakeholders. It is readily teachable to novices. It tells the novice facing his first design assignment where to begin.

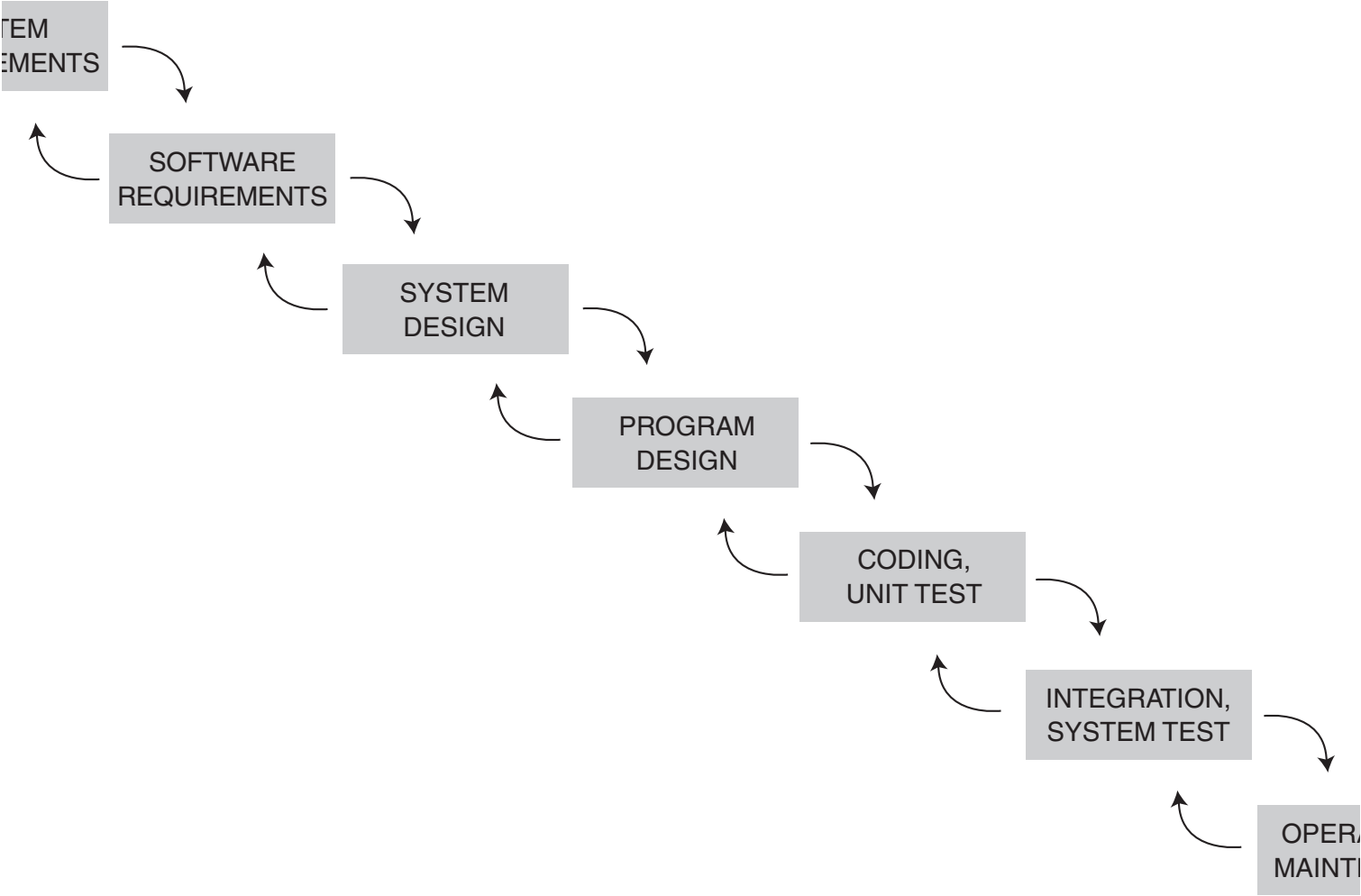
The Rational Model in particular brings yet more advantages. The early explicit statement of goals, secondary desiderata, and constraints helps a team avoid wandering, and it breeds team unification on purposes. Planning the whole design process before starting coding or formal drawings avoids many troubles and much wasted effort. Casting the process as a systematic search of a design space broadens the horizon of the individual designers and lifts their eyes far beyond their previous personal experiences.

But the Rational Model is much too simplistic, even in Simon's richly developed version. Hence we must examine its faults.

Notes and References

1. Following Simon [1981], *The Sciences of the Artificial*, throughout this book I use *man* as a general noun, encompassing both genders, and *he*, *him*, and *his* as androgynous pronouns. I find it more gracious to continue the long tradition of including women and men equally in these general pronouns than to adopt more awkward, hence distracting, constructions.
2. To *satisfice* is to make good enough without necessarily optimizing (Simon [1969], *The Sciences of the Artificial*, 64).
3. But see Akin [2008], "Variants and invariants of design cognition," who finds evidence from the DTRS7 protocols that building architects tend to search laterally among several alternatives at every level, whereas engineering designers emphasize depth-first search based on an initial solution proposal.
4. Pahl and Beitz [1984ff.], *Engineering Design*.
5. Rittel and Webber [1973], "Dilemmas in a general theory of planning," define this term formally. It is well discussed in http://en.wikipedia.org/wiki/Wicked_problem.
6. Brooks [1995], *The Mythical Man-Month*, 265.
7. Royce [1970], "Managing the development of large software systems," 329.
8. Blaauw and Brooks [1997], *Computer Architecture*.

This page intentionally left blank



3

What's Wrong with This Model?

Sometimes the problem is to discover what the problem is.

GORDON GLEGG [1969], *THE DESIGN OF DESIGN*

A designer makes things. ... Typically his making process is complex. There are more variables—kinds of possible moves, norms, and interrelationships of these—than can be represented in a finite model.

DONALD A. SCHÖN [1984],
THE REFLECTIVE PRACTITIONER

Software's Waterfall Model

After Royce [1970], "Managing the development of large software systems," and Boehm [1988], "A spiral model of software development and enhancement." Royce © 1970 IEEE. Boehm © 1988 IEEE.

In fact, every designer will recognize the Rational Model as only an ideal. It somehow describes how we think the design process ought to work, but not how it works in real life.

Indeed, not every engineer will even admit to harboring so naive and idealistic a model in his heart. But I think most of us really do; I did for quite a long time. Therefore, let us take a hard critical look at the Rational Model of design, to identify precisely where it most departs from reality.

We Don't Really Know the Goal When We Start

The most serious model shortcoming is that the designer often has a vague, incompletely specified goal, or primary objective. In such cases:

The hardest part of design is deciding what to design.

As a student I spent one summer working at a large missile company, where I was once set to work designing and building a little database system for keeping track of the 10,000 drawings for a radar subsystem, and the updating status of each.

After a couple of weeks, I had a working version. I proudly presented a sample output report to my client.

"That's fine—it is what I asked for—but could you change it so that . . . ?"

Each morning for the next few weeks, I presented my client with the output report, revised yet again to accommodate the previous day's request. Each morning he studied the product report and asked for yet another system revision, using the same polite mantra.

It was a simple system (implemented on punched-card machines), and the revisions were conceptually simple. The most comprehensive change was to list the drawings sorted by, and indented to show, goes-into level, where level was represented by a single 0–9 digit in the card. Other refinements included multilevel subtotals, with exceptions of course, and the automatic marking of various noteworthy values with asterisks.

For a while, this frustrated me sorely: "Why can't he make up his mind as to what he wants? Why can't he tell me all at once, instead of one bit a day?"

Then, slowly, I came to realize that the most useful service I was performing for my client was helping him *decide* what he really wanted.

Well, today the software engineering discipline is much more sophisticated. We recognize that rapid prototyping is an essential tool for formulating precise requirements. Not only is the design process iterative; the design-goal-setting process is itself iterative.

This sophistication in software engineering does not forestall, or noticeably reduce, the numerous references in the literature to the “product requirements” as a normal given for a design process. But I will argue that knowing complete product requirements up front is a quite rare exception, not the norm:

A chief service of a designer is helping clients discover what they want designed.

In software engineering, at least, the concept of rapid prototyping has a name and a recognized value, whereas it does not always have the same status in computer design and in building architecture. Nevertheless, I see the same goal iteration happening in these design fields. Increasingly, designers build simulators for computers and virtual-environment walk-throughs for buildings as rapid prototypes to drive goal convergence. Goal iteration must be considered an inherent part of the design process.

We Usually Don't Know the Design Tree—We Discover It as We Go

For the original design of complex structures, such as computers, operating systems, spacecraft, and buildings, each major design effort has enough novelty in the

- Goal
- Desiderata, and the utility function
- Constraints
- Available fabrication technologies

that the designer can rarely sit down and a priori map out the design tree.

Moreover, in high-technology design, few designers can know enough to draw the basic decision tree for their domains. Design projects often last two years or more. And designers get promoted out of design. Consequently, few designers will work in any depth on as many as 100 projects over a working life. This means the individual designer has not begun to explore all the branches of the basic design tree for his discipline. For it is characteristic of engineering designers, as opposed to scientists, that they rarely explore alternatives that are not clearly on the way to a solution.¹

Instead, designers discover the design tree as they work—making a decision, and then seeing the alternatives it opens and closes for the next consequent design decision.

The Nodes Are Really Not Design Decisions, but Tentative Complete Designs

In fact, the very decision tree is itself only a simplistic model of the tree-search process. As Figure 2-1 illustrates, there are parallel attribute branches, as well as alternative branches. The choices in one branch are linked to those in others—by exclusion, affinity, or trade-off. Our massive design tree in *Computer Architecture* is much too simple; the entire “Computer Zoo” in that work is necessary to elucidate the decision linkings.²

This means that at each node of a design tree, one faces not a simple alternative choice among options for one design *decision*, but an alternative choice among multiple tentative complete *designs*.

Moreover, the ordering of decisions laying out a design tree matters greatly, as Parnas expounds in his classic paper “Designing software for ease of extension and contraction.”³

The explosive combinatorics of these complications to the tree model boggle the mind. (This situation is like that of move trees in chess.) This difficulty is explored further in Chapter 16.

The Goodness Function Cannot Be Evaluated Incrementally

The Rational Model assumes that design involves a search of the design tree, and that at every node, one can evaluate the goodness function of the several downward branches.

In fact, one cannot in general do this without exploring all the downward branches to all their leaves, for many goodness measures (for example, performance, cost) will depend heavily on the subsequent design detailing. So although the goodness evaluation is possible in principle, one encounters here again the combinatorial explosion of alternatives in practice.

So what is a designer to do? Estimate, of course, either formally or informally. One *must* trim the design tree as one goes down.

Experience. Many aids help intuition in this process. One is experience, both direct and surrogate: “The designers of OS/360 exposed detailed formats of system-wide-shared Control Blocks in Operating System/360, and it proved a maintenance nightmare. We will encapsulate them as objects.” “The Burroughs B5000 family long ago explored the descriptor-based computer architecture. The performance hit was inherently too great, so we won’t explore that subtree.” Of course, the technological trade-offs are no longer the same, but the experience lesson is illuminating anyway. The most potent reason to study design history is to learn what *doesn’t work*, and *why*.

Simple Estimators. Designers routinely use simple estimators early in the design tree exploration. A building architect, given a budget goal, applies a rough square-foot cost estimate, derives a square-foot goal, and uses that for subsequent pruning of the design tree. Computer architects use instruction mixes to do rough-cut early estimates of computer performance.

A danger, of course, is that the rough estimator may lop off design branches that are in fact feasible but appear infeasible because of the very approximation involved in the estimator. I have watched an architect quote high costs for pushing out a wall under an already specified roof structure, based purely on a routine square-foot estimator. In fact, most of the cost of the added space was in the roof, and that was already committed, so the marginal cost was very low.

One can often get something for nothing, if one has previously bought nothing for something.

The Desiderata and Their Weightings Keep Changing

Donald Schön, the late MIT professor of urban studies and education, and a design theorist, said:

[As the designer] shapes the situation in accordance with his initial presentation of it, the situation "talks back" and he responds to the situation's back-talk.

In a good process of design, this conversation with the situation is reflexive. In answer to the situation's back-talk, the designer reflects-in-action on the construction of the problem, the strategies of action, or the model of the phenomena, which have been implicit in his moves.⁴

In short, as one ponders the trade-offs, there comes a new understanding of the whole design problem as an intricately interlocked interplay of factors. With it comes a change in the weightings of the desiderata. The same thing happens as the client, if there is one, grows his understanding of what he will get and develops his detailed vision of how he will use it.

In our house-remodeling design, for example (Chapter 22), a simple question, overlooked in the original program, arose well along in design, as my wife and I applied use scenarios to preliminary designs: "Where will guests at meetings put their coats?" This seemingly low-weight desideratum in fact tipped the big scales, and occasioned moving the Master Bedroom from one end of the house to the other.

Moreover, for designs that must be separately fabricated, such as buildings and computers, the designer learns from the builders a growing understanding of the interactions between design and fabrication. So many desiderata and constraints shift and refine. The fabrication technology may evolve as well, an especially common occurrence during computer design.

Since many desiderata (such as speed) are weighted on a value/cost ratio, yet another phenomenon occurs. As design proceeds, one finds opportunities to add some particular goodness at a very low marginal cost. So something that had not entered the original desiderata list at all comes in, and it often takes on a value that may demand preserving in later design changes.

Only after UNC's Sitterson Hall was designed, built, and in use, for example, did the Computer Science Department, as user, learn that the suite of spaces consisting of the Lower Lobby, Upper Lobby, Faculty Conference Room, Lecture Halls, and Vestibules combined beautifully into a facility well suited for hosting conferences of up to 125 people, with minimal impact on the work in the rest of the building. This was serendipitous—no such function was contemplated in the original architectural program. Yet it is a high-value feature: any future revision of Sitterson would surely aim to preserve this capability.

The Constraints Keep Changing

Even if the goal were fixed and known, all the desiderata enumerated, the design tree known precisely, and the goodness function precisely defined, design would still be iterative, because the constraints keep changing.

Often the environment changes—the city council passes new shadow-casting setback requirements; the electrical code has an annual updating; a microchip one planned to use is withdrawn by the vendor. The world keeps changing around us, even while we design.

The constraints also change due to discovery during the design process, or during the fabrication—the builders hit solid rock; analysis shows that chip cooling has newly become a constraint.

Not all constraint changes are increases. Often constraints go away. When this is fortuitous instead of intentional, the skillful designer recognizes the new opportunity and, with his flexible design, leaps to exploit it.

Alas! Not all designs are flexible. More commonly, when we are deep into a design process, we do not recognize that a constraint has disappeared, nor do we remember which design alternatives it formerly foreclosed.

It is important to list the known constraints explicitly at the start of the design process, as part of what architects call the *design program*. The design program is a document, prepared with the client, that sets forth the goal, the desiderata, the constraints. An example is given in this book's Web site. The design program is *not* the same thing as a formal requirements statement, which usually has contractual force in defining acceptability of a design.

The explicit listing of constraints smokes them out early, avoiding unpleasant surprises. It also impresses them on the designer's mind, radically improving the chances that he will recognize when one goes away.

All of us have designed around constraints, a process that calls forth much invention and exploration of unconventional corners of the design space. This is part of the fun of design, and a big part of the challenge.

Changing Constraints Outside the Design Space. Sometimes, however, a design breakthrough is achieved by stepping completely outside the design space, and working there to remove the design constraint. In designing the house wing (Chapter 22), I wrestled a long time, unsuccessfully, with a shadow-casting setback requirement constraint and the Music Room's desiderata (hold two grand pianos, an organ, and a square space for a string octet plus a 1-foot teaching margin). Figure 3-1 shows one iteration of the design, and the constraints.

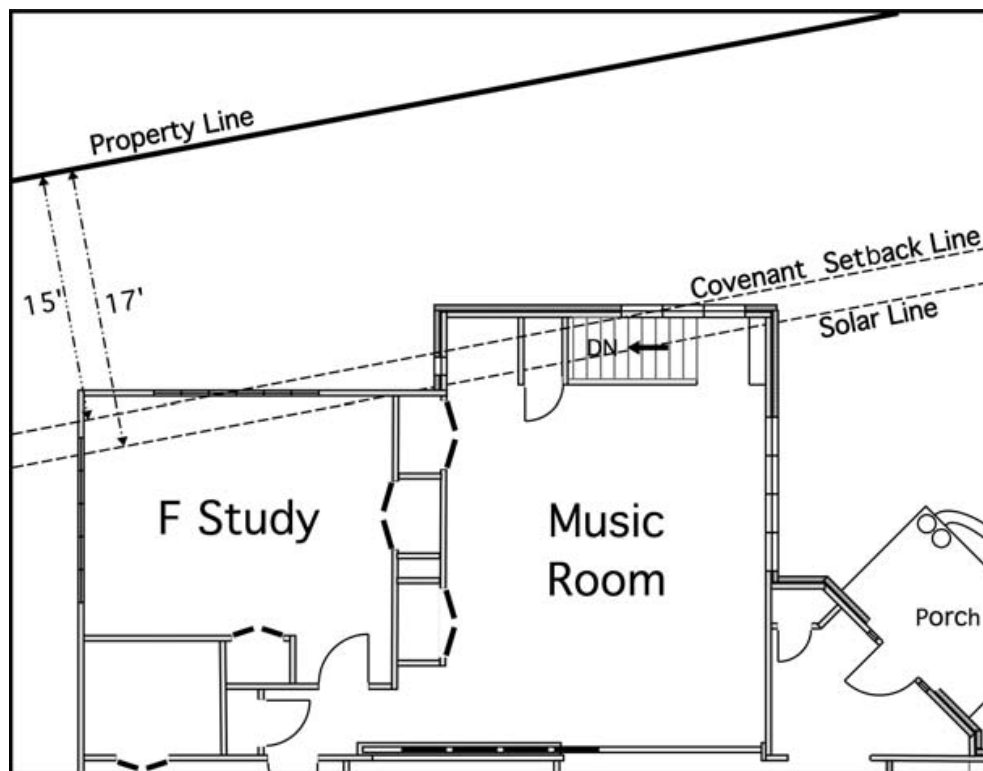


Figure 3-1 Design up against constraints

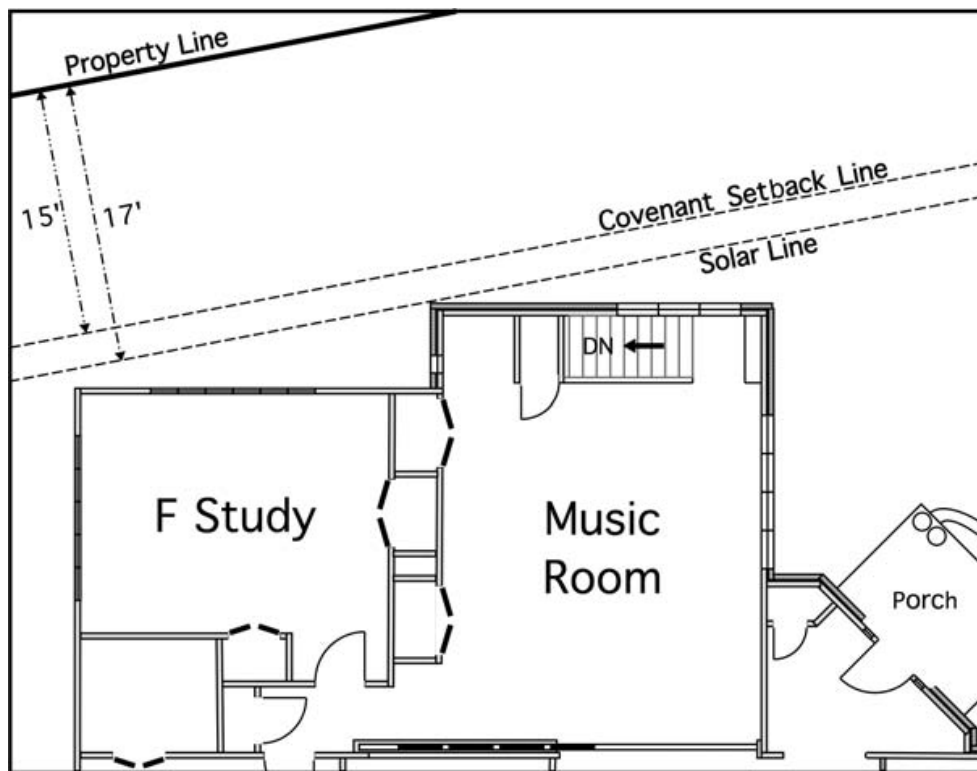


Figure 3-2 Constraint eased

The intractable design problem was finally solved completely outside the design space—I bought a 5-foot strip of land from my neighbor. This was probably cheaper and surely faster than attempting to get a setback variance from the city council, another outside-the-design-space approach. It also liberated other parts of the design, notably the placement of the northwest corner of F Study (Figure 3-2).

The explicit listing of known constraints in the design program helps here, too. The designer can periodically scan the list, asking, “Can this constraint now be removed because the world has changed? Can it be entirely circumvented by working outside the design space?”

Others' Critiques of the Rational Model

A Natural Model. The Rational Model as presented and criticized above may seem naive. But it is a very natural model for

people to conceive. This naturalness is strongly corroborated by the independent creation of the Simon version, the Waterfall Model version, and the Pahl and Beitz version. Yet, from early on, there have been cogent critiques of the Rational Model from the design community.^{5,6,7}

Designers Just Don't Work That Way. Perhaps the most devastating critique of the Rational Model, although perhaps the hardest to prove, is that most experienced designers just don't work that way. While the published critiques have only rarely made the "emperor has no clothes" statement that the model simply does not reflect professional practice, one senses that overriding conviction behind all the detailed analyses.⁸

Nigel Cross, in his gentlemanly way, is perhaps the most articulate exception. Citing many studies, he says bluntly:

Conventional wisdom about problem-solving seems often to be contradicted by the behavior of expert designers. But designing has many differences from conventional problem-solving. ... we must be very wary about importing models of design behavior from other fields. Empirical studies of design activity have frequently found "intuitive" features of design ability to be the most effective and relevant to the intrinsic nature of design. Some aspects of design theory, however, have tried to develop counter-intuitive models and prescriptions for design behavior [emphasis added].⁹

And,

The appositional nature of design reasoning has been neglected in most models of the design process. Consensus models of the design process, such as that promulgated by the Verein Deutscher Ingenieure [VDI, 1987] ... propose that designing should proceed in a sequence of stages. ... In practice, designing seems to proceed by oscillating between sub-solution and sub-problem areas, as well as by decomposing the problem and combining sub-solutions.¹⁰

I find both the argument and the empirical evidence quite convincing. This oscillation has indeed characterized all my design experiences. The "where to put the coats?" requirement discovered deep into our house design process is typical.

Royce's Critique of the Waterfall Model. Royce in his original paper describes the Waterfall Model so that he can point out its deficiencies.¹¹ Basically he argues that even with back-arrows describing counterflow between adjacent boxes in the waterfall, the model doesn't work. His prescription is, however, simply to augment the model with counterflow arrows that go back two boxes. A Band-Aid, not a cure.

Schön's Summary of the Critiques.

[Simon] has identified a gap between professional knowledge and the demands of real-world practice. ... Simon proposes to fill the gap ... with a science of design, his science can be applied only to well-formed problems already extracted from situations of practice.

If the model of Technical Rationality ... fails to account for practical competence in "divergent" situations, so much the worse for the model. Let us search, instead, for an epistemology of practice implicit in the artistic, intuitive processes which some practitioners do bring to situations of uncertainty, instability, uniqueness, and value conflict.¹²

But Despite All These Flaws and Critiques, the Rational Model Persists!

Often the original proponent of a theory or technique understands its promise, its liabilities, and its proper domain more clearly than his later disciples. Less gifted, more fervent, their very fervor leads to rigidity, misapplication, oversimplification.

So, unfortunately, are many applications today of the Rational Model. Writing as recently as 2006, design researcher Kees Dorst has to admit,

Although there have been many developments since then, the original work on problem solving and the nature of ill-structured problems, written by Herbert Simon, still looms large in the field of design methodology. The rational problem-solving paradigm, based on the conceptual framework that Simon introduced, is still a dominant paradigm in the field.¹³

Indeed so! In the field of software engineering, we all too often still slavishly follow the Waterfall Model, our own embodiment of the Rational Model.

Verein Deutscher Ingenieure Standard VDI-2221. The Verein Deutscher Ingenieure in 1986 adopted the Rational Model, essentially as set forth by Pahl and Beitz, as an official standard for German mechanical engineering.¹⁴ I have seen many rigidities in thinking engendered by this move. But Pahl himself has been at some pains to clarify that

*Procedures given in VDI-Richtlinie 2221-2223 and Pahl & Beitz (2004) are not of the “straight sequence” type, but should be utilized only as guides for basic purposeful action. A useful approach in actual situations might be to choose either an iterative approach (i.e. with “forward and back” steps) or by repetition using the next higher information level.*¹⁵

DoD Standard 2167A. Similarly, the U.S. Department of Defense in 1985 enshrined the Waterfall Model in DoD Standard 2167A.¹⁶ Only in 1994 did they, under the leadership of Barry Boehm, open up their acquisition by admitting other models.

So What? Does Our Design Process Model Matter?

Why all this fuss about the process model? Does the model we and others use to think about our design process really affect our designing itself? I believe it does.

Not Every Design Thinker Agrees with Me. Professor Ken Wallace of Cambridge, who translated three editions of Pahl and Beitz's work into English, believes the major step forward is to have some model that is readily understood and communicated. He points out how useful it is for beginning designers. The Pahl and Beitz model gives the novice a place to start work on a design, so he doesn't just wander. “I put up the Pahl and Beitz diagram [their Figure 1.6] and explain it. And then my very next slide says, ‘But this is not the way real designers work.’”¹⁷

Hooray! But I am concerned whether younger teachers with less personal design experience always say that.

Suzanne and James Robertson, consultants who practice internationally and authors of excellent major works on requirements formulation, also feel that the deficiencies in the Rational Model don't really matter. "People who understand what design is, know better."¹⁸

Nevertheless, I believe our inadequate model and following it slavishly lead to fat, cumbersome, over-featured products and to schedule, budget, and performance disasters.

Right-Brained Designers. Designers are mostly right-brained people, visually and spatially oriented. Indeed, one of my curb-stone tests for potential design talent is to ask, "Where is next November?" When my listener is puzzled, I elaborate, "Do you have a spatial mental model of the calendar? Many folks do. If you do, would you describe it for me?" The strong candidates almost always have one; the models themselves vary wildly.

Similarly, software design groups invariably scrawl diagrams, not words or code, on their shared whiteboards. Architects consider the broad-pen sketch on tracing paper an indispensable tool for communication, but even more for solo thinking.

Since we designers are spatial people, our process models live deep in our minds as diagrams, whether Pahl and Beitz's vertical rectangle, Simon's tree, or even the waterfall Royce draws and condemns. The diagrams subconsciously influence much of our thinking. Hence I believe a deficient process model hinders us in ways we cannot fully know and can barely suspect.

One obvious injury done by accepting the Rational Model is that we mis-educate our successors. We teach them modes of working that we ourselves do not follow. Hence we leave them unaided in arriving at their own real-world working modes.

I doubt if this is the case with more senior teachers, particularly those with industrial designing experience. We are keenly aware that models are intentional oversimplifications to help us with real-life problems that are frighteningly complicated. So we warn our students that "the map is *not* the terrain," the model is not a complete picture; it may even be inaccurate in what it does incorporate.

In software engineering practice, another kind of harm can readily be spotted—the Rational Model, in any of its forms, leads us to demand up-front statements of design requirements. It leads

us to believe that such can be formulated. It leads us to make contracts with one another on the basis of this enshrined ignorance. A more realistic process model would make design work more efficient, obviating many arguments with clients and much rework. Chapters 4 and 5 elaborate on the requirements problem.

The Waterfall Model is wrong and harmful; we must outgrow it.

Notes and References

1. The engineer needs a satisficing solution; the scientist needs a discovery, and wider exploration often yields one.
2. Blaauw and Brooks [1997], *Computer Architecture*, 26–27, 79–80.
3. Parnas [1979], “Designing software for ease of extension and contraction,” explicitly treats the design process as tree traversal. He argues strongly for making a design as flexible as possible. He urges that one do that by putting the decisions least apt to change nearest the tree root. Flexibility of a design is an important goal. In software engineering, both object-oriented design and agile development methodology have this as a fundamental aim.
4. Schön [1983], *The Reflective Practitioner*, 79.
5. Surprisingly, I found few critiques of the Pahl and Beitz formulation of the Rational Model and many of Simon’s formulation. Pahl and Beitz themselves recognized the inadequacy of the model: in successive editions of their work, their model (Figures 3.3, 4.3 in the second and third English editions) includes more and more explicit iteration steps (Pahl and Beitz [1984, 1996, 2007], *Engineering Design*). Simon’s three editions of *The Sciences of the Artificial* do not reflect any change in the model as proposed, although in personal conversation with me in November 2000 he said that his own understanding of the model had evolved, but that he had had no opportunity to rethink and rewrite accordingly.

Visser [2006], *The Cognitive Artifacts of Designing*, has an excellent Section 9.2, “Simon’s more nuanced positions in later work,”

which examines Simon's evolution as embodied in later papers. Visser shares my surprise that this evolution didn't get reflected in the later editions of *The Sciences of the Artificial*.

6. Holt [1985], "Design or problem solving":

There are two distinct interpretations of engineering design. The problem-solving approach, popular in many tertiary institutions and with an emphasis on solving structured, well defined problems using standardized techniques, may be traced to "hard" systems thinking. The creative design approach, on the other hand, combines analytical and systems thinking with human factors in engineering design to create and take advantage of opportunities to serve society. This paper discusses the limitations of the problem-solving approach in dealing with many real world tasks.

7. Whereas Cross's critique is empirical, Schön criticizes the philosophy underlying the Rational Model. He says that the Rational Model, as enunciated by Simon, is a natural outgrowth of a much more pervasive philosophical mind-set, which he calls Technical Rationality and identifies as a heritage of now-discredited positivism. He finds the underlying philosophy itself totally inadequate for understanding design, even though it has been institutionalized into most professional design curricula:

From the perspective of Technical Rationality, professional practice is a process of problem solving. Problems ... are solved through the selection, from available means, of the one best suited to established ends. But with this emphasis on problem solving, we ignore problem setting, the process by which we define the decision to be made, the ends to be achieved, the means which may be chosen. In real-world practice, problems do not present themselves to the practitioner as givens. They must be constructed from the materials of problematic situations which are puzzling, troubling, and uncertain. ... a practitioner must do a certain kind of work. He must make sense of an uncertain situation that initially makes no sense. ... It is this sort of situation that professionals are coming increasingly to see as central to their practice. ... Technical Rationality depends on agreement about ends.

8. A vivid example is Seymour Cray's 1995 quote: "I'm supposed to be a scientific person, but I use intuition more than logic in making basic decisions." <http://www.cwhonors.org/archives/histories/Cray.pdf>, accessed September 14, 2009.
9. Cross [2006], *Designerly Ways of Knowing*, 27.
10. Cross [2006], *Designerly Ways of Knowing*, 57. Dorst [1995], "Comparing paradigms for describing design activity," has an especially good discussion of Simon versus Schön. Their journal article is reprinted in Cross [1996a], *Analysing Design Activity*. Dorst also shows that for the Delft II protocols, Schön's model fits the observed designer behavior much more accurately.
11. Royce [1970], "Managing the development of large software systems."
12. Schön [1983], *The Reflective Practitioner*, 45–49.
13. Dorst [2006], "Design problems and design paradoxes."
14. VDI [1986], *VDI-2221: Systematic Approach to the Design of Technical Systems and Products*.
15. Pahl [2005], "VADEMECUM—recommendations for developing and applying design methodologies."
16. DoD-STD-2167A tried to fix this but unfortunately put a waterfall diagram in a prominent place and left things pretty much as they were. MIL-STD-498 superseded 2167A and addressed the model problem. DoD has since superseded 498 by adopting industry standards IEEE/EIA 12207.0, IEEE/EIA 12207.1, and IEEE/EIA 12207.2.
17. Personal communication [2008].
18. Personal communication [2008].

This page intentionally left blank