Part 4
Architecture as Decisions

Parts 2 and 3 presented principles and methods for analyzing and synthesizing architecture. Because of the complexity inherent in architecture, many of the steps involved in analysis and synthesis are difficult to conduct exhaustively. For example, we considered only seven concepts for the Hybrid Car in Chapter 12, and we did not exhaustively search the space of concept fragment combinations. Although some aspects of these tasks can be reduced to computation, great care must be taken when distilling the outputs of computational models to yield recommendations. With these thoughts in mind, in Part 4 we embark on architectural decision support, where we will investigate how methods and tools can support, but not replace, the role of the architect.

Part 4 will introduce computational methods and tools that can be useful to the system architect. These are taken from the fields of decision analysis, global optimization, and data mining.

Our intentions are twofold. On the one hand, methods and tools can augment the architect's analysis, helping to reduce ambiguity, employ structured creativity, and manage complexity. However, the second purpose of Part 4 is to create mental models for the reader (such as the tradespace mental model). It has been our experience that some of the constructs and modes of reasoning presented in this part are useful for supporting architectural decisions, even without actually building analytic models.

Part 4 is organized as follows. In Chapter 14, we illustrate the system-architecting process as a decision-making process. We provide an overview of decision support, with an eye to restricting the field of view to the correct subset of decisions. In theory, every stroke of the painter's brush is a decision, but not all decisions are *architectural decisions*. We illustrate these ideas with a simple case study of the Apollo mission-mode decision, to demonstrate the power of these methods and tools.

In Chapter 15 we examine how to synthesize information from an architecture tradespace—that is, the result of evaluating a set of architectures. We discuss how to combine tools such as Pareto analysis, design of experiments, and sensitivity analysis to gain a better understanding of the main architectural tradeoffs, sensitivities, and couplings between decisions. In particular, we explore how to structure a complex architectural tradespace around a hierarchy of decisions. We will argue that developing the hierarchy of decisions is a key lever on managing complexity, and that knowing which decisions have the greatest impact is at least as important as finding the model's prediction of the best architecture.

In Chapter 16, we show how to encode architectural decisions in a model that can be used by a computer to automatically generate and explore a tradespace. We start by introducing the idea of system architecting problems and the canonical types of decisions that the architect makes. Based on this study, we propose six patterns of architectural decisions, and we introduce their corresponding formulation as optimization problems. We look at the mathematical structure of these problems and discuss similarities to, and differences from, classical optimization problems (knapsack problem, traveling salesman problem). A single class of tools—heuristic optimization algorithms—is introduced as an approach to solving architectural problems computationally.

The appendices describe additional tools that can be used to support the main functions of decision support (representing, simulating, structuring, and viewing). In particular, we discuss in the appendices how to automatically enumerate and evaluate architectures using rule-based systems, and how to structure the tradespace using clustering algorithms.

Chapter 14

System Architecture as a Decision-Making Process

The authors would like to acknowledge the substantial contributions of Dr. Willard Simmons to this chapter.

14.1 Introduction

The job of a system architect is to transform a set of needs and goals into a system architecture. For complex systems, the task of architecting is challenging, because the complex relationships between design parameters and their alternatives introduce a massive search space that challenges both humans' and computers' abilities to exhaustively process the space. This chapter argues that a system architecture can be effectively represented as a set of *interconnected decisions*. It should be no surprise that we consider these decisions as a system—they have entities (the decisions) and relationships (the connections between the decisions). These decisions are an intermediate system, between the system of needs and the final architecture. Most important, these decisions can be used both cognitively and computationally to reduce the perceived complexity of the architecting task.

Let's start with an example. Consider the process of architecting the Apollo project of the 1960s. In his famous May 1961 speech, President Kennedy stated that the United States would send a man to the Moon and return him safely to Earth by the end of the decade. [1] This and other needs and goals were then transformed into an architecture for the system. System architects achieved this transformation by identifying, and making decisions to reduce, the candidate space of architectures.

NASA reduced the perceived complexity of the tradespace by identifying key decisions that would define the mission-mode, which described how and where the various elements would meet in space and how the crew would shift between elements. Historical evidence [2] shows that progress in the Apollo program was limited until June 1962, when the decision was made to choose Lunar Orbit Rendezvous as the Apollo mission-mode, setting the program on a path to the successful Moon landing in 1969. [3]

There are two major decisions in this architecture: Will there be rendezvous and docking operations in Earth orbit (called Earth orbit rendezvous, or EOR)? And will there be rendezvous and docking operations in lunar orbit (called lunar orbit rendezvous, or LOR)? If there is neither EOR nor LOR, it is called the direct mission-mode, and a single huge spacecraft is launched into Earth orbit, travels to Moon orbit, descends on the Moon's surface, ascends into Moon orbit, and

returns to Earth. This option minimizes the number of vehicles developed. In the LOR mode, two spacecraft are injected into lunar orbit, and only one descends to the Moon. After completing the mission on the Moon's surface, it ascends and is assembled with the second vehicle that carries the propellant required to return to Earth. This option is the most fuel efficient. Finally, in the EOR mission-mode, two spacecraft are assembled in Earth orbit and then travel to the surface of the Moon. This option is preferable from a risk standpoint, because the riskiest operation (the assembly of spacecraft) occurs near Earth, which greatly facilitates the return of the astronauts in case of failure. A fourth mission-mode includes both EOR and LOR.

The mission-mode involving lunar orbit rendezvous (LOR) was chosen for the Apollo mission. By making this decision, the system architects provided design engineers with a stable category of acceptable designs that were eventually refined into a single detailed design.

In this chapter, we present the notion of *system architecting as a decision-making process*—a perspective that can dramatically change how early design activities are conducted. In the introduction to this text, we highlighted the idea that the first ten decisions taken in an architecture determine a majority of the performance and cost. Even with only two choices per decision, there are $2^{10} = 1024$ possible architectures in this design space. Much of what we discussed in Parts 1 through 3 of this text helps system architects apply experience and heuristics to winnow this space. In this chapter we will show how typical decision support tools, such as decision trees, can make it easier to comb through large combinatorial spaces and help the architect to make informed choices. It goes without saying that this is not intended to replace the judgment of the architect, but to augment it. We seek to develop a set of methods and tools that are primarily aimed at reducing complexity but can also help in resolving ambiguity and thinking creatively. In other words, we are trying to optimize the functional allocation between humans and computers in the system architecting task.

14.2 Formulating the Apollo Architecture Decision Problem

Heuristics for Decisions

Formulating the Apollo project as a decision-making problem requires selecting a set of decisions (with their corresponding alternatives) and a set of metrics. This section describes three heuristics for formulating decisions and shows how they are applied to the Apollo architecture problem.

The first heuristic for formulating the decision problem is to *carefully set the boundaries of the architectural space under consideration* (see Principle of the System Problem Statement in Chapter 11). What range of architectures should be included in the Apollo analysis? The highest-level specification of the Apollo system can be derived from President Kennedy's 1961 speech: "landing a man on the Moon and returning him safely to the Earth." This statement sets as the minimum scope that Apollo should land at least one man on the Moon. But architectures that extend to a much larger scope of issues (Mars exploration, space stations, and so on) may not have been appropriate, considering the limited schedule NASA was given.

One decision that NASA faced was the size of the crew. In the early 1960s, landing even one man on the Moon was an extremely ambitious goal. [4] However, because a lunar mission is a relatively long and complex space mission, it may have been too risky to send a lone astronaut on the voyage. Again considering schedule constraints, it was probably inappropriate to consider

missions with large teams of astronauts, such as von Braun's proposed Conquest of the Moon. [5] In this retrospective analysis, we will bound the number of crew members to at least one and no more than three.

Another decision was that of the mission-mode. The feasibility and reliability of in-space rendezvous and docking was a heavily debated topic at the beginning of the Apollo project. Earth orbit rendezvous was considered of lower technical risk but also of lower benefit. John C. Houbolt, a NASA engineer, showed that lunar orbit rendezvous was challenging but technically feasible. He argued that missions including rendezvous and docking in lunar orbit should be considered, because they provided opportunities for saving mass and launch cost. [6] Therefore, in this retrospective analysis we will consider both EOR and LOR.

The second heuristic states that *the decisions should significantly influence the metrics by which the architecture is evaluated*. This seems obvious, but when architecture decision models are created, some of the decisions are often found to have low impact on the metrics (that is, the metrics are relatively insensitive to these decisions), implying that these decisions could be dropped. Two metrics that are considered important in space missions are the total mass of the mission elements and the probability of mission success. Both strongly depend on the mission-mode and the crew size, as well as on the fuel types to be used for spacecraft maneuvers. [7]

The third heuristic states that *the decision model should include only architectural decisions*. For Apollo, decisions related to the mission-mode directly drive the function-to-form mapping. For example, if the mission-mode includes lunar orbit rendezvous, the concept for the mission includes two vehicles: one crew vehicle that has a heat shield so that it can re-enter Earth's atmosphere, and a lunar lander vehicle that is specialized for descent to the surface of the Moon. Sometimes decisions indirectly influence architecture. For example, the fuel types influence the kind of engine used and the requirements for propellant tanks. The application of this third heuristic led to the removal of some decisions from our model; these include the decision on the location of the launch site, which does not substantially change the architecture.

Apollo Decisions

After considering the three heuristics, we selected a set of nine decisions for the Apollo study (see Figure 14.1). The process of creating a decision model is iterative in nature. For example, if a model shows that architectures that differ only in Decision X produce the same metric scores, then either Decision X can be eliminated from the model, or a new metric should be included that reflects the effect of this decision on metrics.

The set of nine decisions shown in Figure 14.1 includes decisions related to the mission-mode, the crew size, and the rocket propellant type used for Apollo. The figure also shows the range of allowed alternatives for each decision. Such tables are called morphological matrices, and in Section 14.5 we will discuss them as a tool to organize architectural decisions.

The first five decisions listed and illustrated in Figure 14.1 are related to the mission-mode: *EOR* (Will there be rendezvous and docking in Earth orbit?); *earthLaunch* (Will the vehicles go into orbit around the Earth or launch directly to the Moon?); *LOR* (Will there be rendezvous and docking in lunar orbit?); *moonArrival* (Upon arrival, will the vehicles go into orbit around the Moon or descend directly to the Moon's surface?); and *moonDeparture*

shortID	Decision	units	alt A	alt B	alt C	alt D
EOR	Earth Orbit Rendezvous	none	no	yes		
earthLaunch	Earth Launch Type	none	orbit	direct		
LOR	Lunar Orbit Rendezvous	none	no	yes		
moonArrival	Arrival At Moon	none	orbit	direct		
moonDeparture	Departure From Moon	none	orbit	direct		
cmCrew	Command Module Crew	people	2	3		
lmCrew	Lunar Module Crew	people	0	1	2	3
smFuel	Service Module Fuel	none	cryogenic	storable		
lmFuel	Lunar Module Fuel	none	NA	cryogenic	storable	

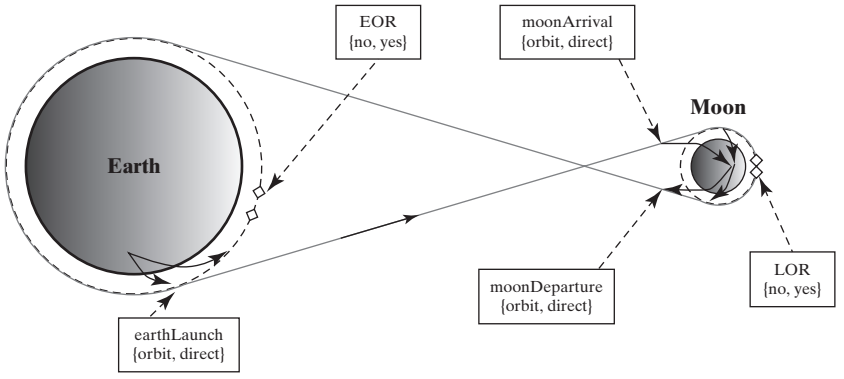

FIGURE 14.1 Mapping of historical Apollo mission-modes to the nine decisions. Note that the combinations of decision assignments listed must also satisfy the logical constraints shown in Figure 14.2.

(Upon ascent, will the vehicles go into lunar orbit or proceed directly toward the Earth?). All five of these decisions indicate alternative maneuvers at different points of the mission. By combining one alternative from each of the five decisions, a mission-mode can be defined.

The four remaining decisions are related to crew size and fuel type. The choice from *cmCrew* sets the size of the Command Module crew, and *lmCrew* includes the choices for the size of the Lunar Module crew (which of course is zero if there is no lunar module). The choices for *smFuel* and *lmFuel* set the fuel of the Service Module and the Lunar Module (NA if it does not exist). "Cryogenic" indicates a higher-energy LOX/LH2 propellant, and "storable" represents a lower-energy but higher-reliability hypergolic propellant.

Constraints and Metrics

In addition to defining the decisions, a complete description of the architecture model contains constraints and metrics. The constraints capture available knowledge about the system and the relationships between the decisions. *Logical constraints* are those that identify combinations of decisions that are not possible. Table 14.1 shows the logical constraints in the Apollo example. For instance, constraint d says that the Lunar Module crew must be smaller than or equal to the Command Module crew—you cannot create astronauts in the vicinity of the Moon!

TABLE 14.1 | Constraints in the Apollo example

Id	Name	Scope	Equation
a	EORconstraint	EOR, earthLaunch	(EOR == yes && earthLaunch == orbit) || (EOR == no)
b	LORconstraint	LOR< moonArrival	(LOR == yes && moonArrival == orbit) || (LOR == no)
c	moonLeaving	LOR, moonDeparture	(LOR == yes && moonDeparture == orbit) || (LOR == no)
d	lmcmcrew	cmCrew, lmCrew	(cmCrew ≥ lmCrew)
e	lmexists	LOR, lmCrew	(LOR == no && lmCrew == 0) || (LOR == yes && lmCrew > 0)
f	lmFuelConstraint	LOR, lmFuel	(LOR == no && lmFuel == NA) || (LOR == yes && lmFuel ! = NA)

Table 14.1 shows that the key decision is LOR. If it is yes, there cannot be direct descent at lunar arrival or direct lunar departure (constraints b and c), and there must be a Lunar Module with a crew of at least 1 and a propellant (constraints e and f).

There are also weaker forms of "reasonableness constraints" that encode things you would *probably* not do together. For example, if you were to construct an international effort to go to the Moon, you would probably not have the United States build a lander, and a second nation also build one. That would be wasteful of resources and hence unreasonable. But there is nothing *logically* incorrect about it.

In assessing the effectiveness of architectures, we usually find that there are metrics that quantify some measures of performance, some measures of cost, and some measures of developmental and operational risk. Our Apollo case is iso-performance, which means that all architectures provide for landing at least one crew on the lunar surface, satisfying Kennedy's goal. Therefore, the two metrics that we use in assessing the potential success of the Apollo project are (1) operational risk and (2) initial mass to low Earth orbit (IMLEO), a proxy for cost. The IMLEO is calculated for any architecture using the rocket equation [8] and the parameters taken from Houbolt's original documents. [9]

The risk metric was based on Table 14.2, which directly links decisions with the probability of successful operation. The overall probability of success is obtained by multiplying the individual probabilities together for the operations represented in any one architecture. The risks metric contains four categories of risk: high (0.9 probability of success), medium (0.95), low (0.98), and very low (0.99). The risk factor for each operation is assessed on the basis of documents written in the early 1960s and interviews with key decision makers.

Metrics are another way in which decisions can be linked to each other. For example, the probability of mission success is computed by multiplying all individual probabilities. Thus each decision is linked through this metric to all other decisions.

Computed Apollo Architecture

Figure 14.2 shows the results of possible architectures for the Apollo program, which were obtained by exhaustively calculating the outcomes (IMLEO and probability of mission success) for all combinations of decisions that are not logically constrained. Among the best solutions closest

TABLE 14.2 | Table used to compute the risk metric in the Apollo example with the probability shown in brackets below each alternative.

shortID	Decision	alt A	alt B	alt C	alt D
EOR	Earth Orbit Rendezvous	no	yes		
risk		(0.98)	(0.95)		
earthLaunch	Earth Launch Type	orbit	direct		
risk		(0.99)	(0.9)		
LOR	Lunar Orbit Rendezvous	no	yes		
risk		(1)	(0.95)		
moonArrival	Arrival at Moon	orbit	direct		
risk		(0.99)	(0.95)		
moon Departure	Departure from Moon	orbit	direct		
risk		(0.9)	(0.9)		
cmCrew	Command Module Crew	2	3		
risk		(1)	(1)		
lmCrew	Lunar Module Crew	0	1	2	3
risk		(1)	(0.9)	1	1
smFuel	Service Module Fuel	cryogenic	storable		
risk		(0.95)^(burns)	(1)		
lmFuel	Lunar Module Fuel	NA	cryogenic	storable	
risk		(1)	(0.9025)	1	

to the Utopia point (low mass and high probability of success, identified with the "U" symbol in Figure 14.2), we've highlighted eight architectures. What is remarkable is that these eight mirror the three main proposals considered at the time: von Braun's Direct mission, Houbolt's LOR concept, and the Soviet mission design.

Point designs 1 and 2 are "direct" missions with three and two crew members, respectively. A direct mission-mode implies that the mission has neither lunar orbit rendezvous nor Earth orbit rendezvous, and no lunar module. These types of missions were among the ones initially proposed by von Braun. [10] They have high mission reliability, at the cost of very high IMLEO.

Point designs 3 to 8 are architectures that include lunar orbit rendezvous maneuvers. Point design 3 matches the actual configuration of Apollo: It has three crew members in the

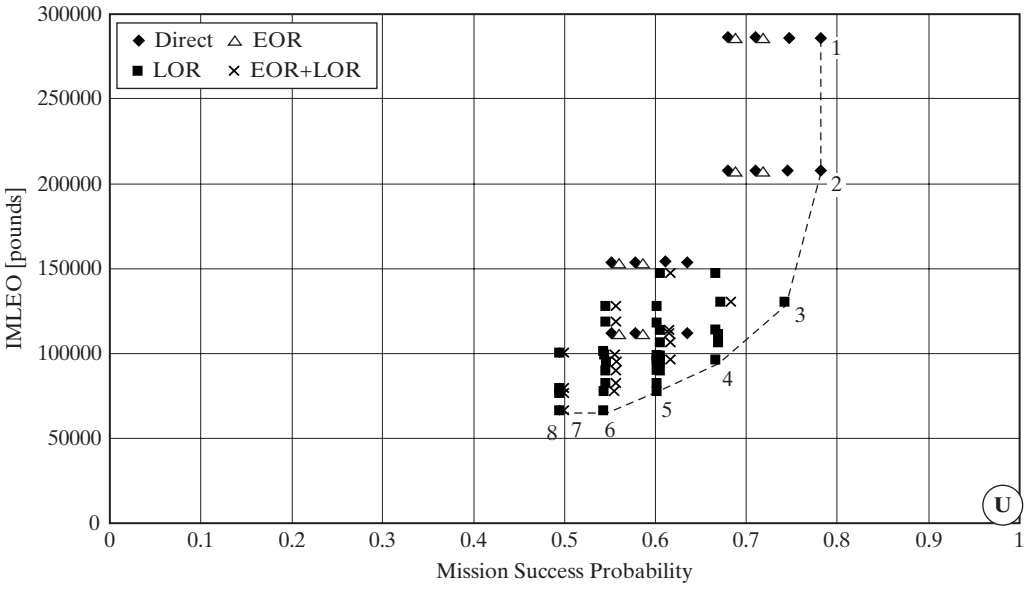

FIGURE 14.2 Apollo tradespace plot comparing IMLEO to probability of mission success. Each point in the plot indicates a logically feasible combination of decision assignments. The dashed line indicates the best architectures. The "U" symbol indicates the Utopia point, an imaginary point with perfect scores for all metrics.

command module, has two crew members in the lunar module, and uses storable propellants for both the service module and the lunar module. [11] It represents a reasonable compromise between mass and risk. Point 8, which is the minimum-mass configuration, uses two crew members in a command module and one crew member in a lander with cryogenic propellants. Point design 8 is the point that most closely models the proposed Soviet lunar mission's architecture. [12] Searching computationally through the possible architectures, we surface the three primary choices considered during the mission-mode decision process in the 1960s, and we come to understand the essential tradeoff between mass and risk. We will see in the next chapter how to mine this type of chart for useful information that can help us structure the system architecture process.

14.3 Decisions and Decision Support

According to R. Hoffman, the word "decide" comes from the Sanskrit word *khid 'ati,* meaning "to tear," the Latin word *cædare,* meaning "to kill" or "cut down," and also the Latin word *decædare,* which means "to cut through thoroughly." [13] In contemporary English, a *decision* is "the passing of judgment on an issue under consideration" [14] or a purposeful selection from mutually exclusive alternatives. *Decision making* is "goal-directed behavior in the presence of options" [15] that culminates in one or more decisions. The key ideas in decision making are that there is a situation with multiple alternatives; a selection is made that separates the solution space; and there is some expected benefit that will be achieved by making this decision.

Decision support is about assisting decision makers in making a decision. Many decision support processes can be described by Herbert Simon's four-phase process, [16] which includes:

1. **Intelligence Activity:** "Searching the environment for conditions calling for a decision."
2. **Design Activity:** "Inventing, developing, and analyzing possible courses of action."
3. **Choice Activity:** "Selecting a particular course of action from those available."
4. **Review Activity:** "Assessing past choices."

According to Simon, decision makers tend to spend a large fraction of their resources in the Intelligence Activity phase, an even greater fraction of their resources in the Design Activity phase, and small fractions of their resources in the Choice Activity phase and the Review Activity phase.

Simon's research makes the complementary observation that there are two "polar" types of decisions: programmed decisions and non-programmed decisions (these have sometimes been called "structured" and "unstructured" decision problems by subsequent authors [17]). Examples and characteristics of these two types of decisions are shown in Figure 14.3.

Programmed decisions are "repetitive and routine decisions" where a procedure for making decisions for this type of problem has been worked out *a priori*. Examples of programmed decisions range from simple to very complex. For instance, deciding how much to tip a waiter, deciding the optimal gains of a control system, and deciding the routing of all aircraft flying over the United States can all be considered programmed decisions. In each case there is a known

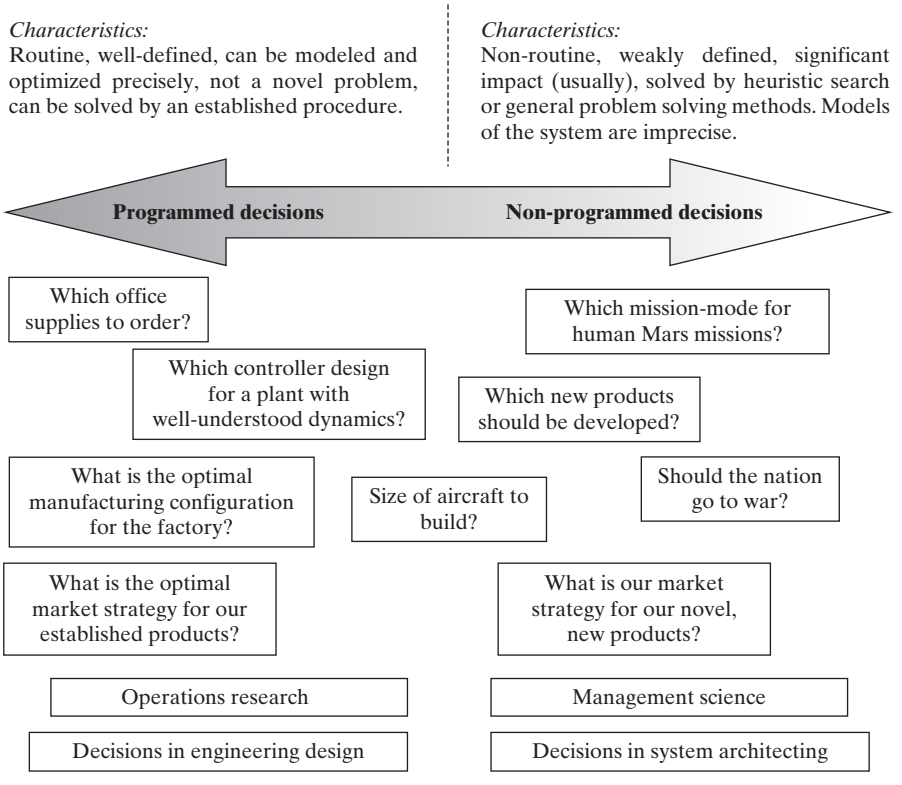

FIGURE 14.3 The spectrum of decisions: Programmed vs. non-programmed decisions.

and defined approach that a decision maker can follow to arrive at a satisfactory choice. Models exist for the behavior of the problem, and the objectives are clearly definable. Note that classifying a decision as programmed does not imply that it is an "easy" decision, but only means that a method to solve it is known and available. Furthermore, the classification "programmed" does not quantify the amount of resources necessary to use the prepared methodology. For many engineering problems, this predetermined routine will be difficult to implement or expensive to compute. [18] Simon considers programmed decisions to lie in the domain of Operations Research.

On the other end of the spectrum are non-programmed decisions. These decisions are novel, ill-structured, and often consequential. An example is the decision of the mission-mode for Apollo. Simon maintains that non-programmed decisions are generally solved by creativity, judgment, rules of thumb, and general problem-solving methods such as heuristics. The examples given in Figure 14.3 include deciding whether a nation should go to war, deciding what market strategy to adopt for new, unproven products, and deciding on the mission-mode for human Mars missions. Simon considers non-programmed decisions to lie in the domain of management science.

In many cases, decisions that have been thought to be non-programmed become programmed once someone is clever enough to invent a programmed method to solve that problem. Perhaps a better name for non-programmed decisions is "not-yet-programmed decisions." An example of people becoming able to systematically program a previously non-programmed problem occurred when Christopher Alexander introduced the concept of "pattern language" as a systematic way to develop civil architectures. The pattern language catalogs the elements of an architecture as reusable triples made up of the context in which they are relevant, the problem they are intended to solve, and the solution they provide. [19] We will discuss the notion of patterns further in Chapter 16.

It is evident that there is sometimes an opportunity to "program" decisions that were previously thought to be "non-programmed." The goal of the remainder of this chapter and the next is to develop a decision support system to comprehensively and efficiently examine a solution space using rigorous analysis as suggested by Simon, rather than using heuristics.

14.4 Four Main Tasks of Decision Support Systems

Decision support consists of assisting decision makers in making a decision. Today, many commercial software programs (such as Decision Lens®, TreePlan®, and Logical Decisions®) provide decision support systems by implementing some form of automation to support the decision-making process.* The goal of these systems is to enhance the efficiency of decision makers by providing tools to quantitatively and qualitatively explore a space of alternatives for single or multiple decisions.

In constructing a decision support system (DSS) for architecture, it is useful to characterize the tasks. We assert that the task of Simon's Design Activity applied to architecture can be described by four "layers": representing, structuring, simulating, and viewing [20]:

- The **representing** layer includes methods and tools for representing the problem for the human decision maker and encoding the problem for computation. The matrix

*The Institute for Operations Research and the Management Sciences (INFORMS) maintains a list of decision analysis software at its website. http://www.orms-today.org/surveys/das/das.html

in Figure 14.1, showing the choices for each Apollo decision, is an example of representing. This morphological matrix is useful because it provides a simple representation of the decisions and choices to be considered. However, it does not provide any information related to constraints (in the structuring layer) or preferences (in the simulating layer). Other ways of representing architectural spaces include trees, graphs, OPM, and SysML.

- The **structuring** layer involves reasoning about the structure of the decision problem itself. This includes determining the order of decisions and the degree of connectivity between decisions. For example, in the Apollo example, one cannot choose to have an Earth Orbit Rendezvous and simultaneously choose to launch directly to the Moon (see Table 14.1). This type of logical constraint, and other types of couplings between decisions, can be represented using Design Structure Matrices (DSMs) containing bilateral interactions between decisions. DSMs are described in more detail in the next section.
- The **simulating** layer is used to determine which combinations of decisions will satisfy logical constraints and calculate the metrics. The simulating layer is thus about evaluating the ability of a system architecture to satisfy the needs of the stakeholders. For example, in the Apollo example, the simulating layer computes the two metrics shown in Figure 14.2. A variety of tools are used for system simulation, ranging in complexity from simple equations to discrete-event simulation. [21]
- The **viewing** layer presents, in a human-understandable format, decision support information derived from the structuring and simulating layers. For example, the chart shown in Figure 14.2 provides some viewing support by graphically representing the evaluation results from all the architectures in the tradespace. We will discuss tradespaces and how to mine information from them in detail in Chapter 15.

Note that we include in Simon's Design Activity the methods and tools that involve representing, structuring, simulating, and viewing. The actual architecture selection process is the goal of Simon's Choice Activity, and the steps leading to this Choice Activity are discussed in Chapter 16.

14.5 Basic Decision Support Tools

We have already seen, in Parts 2 and 3, examples of tools that are useful in some of the four layers of decision support. For example, we used Object Process Methodology (OPM) to represent not only system architectures but also specific decision-making problems, such as function or form specialization (see, for example, Figure 7.2). We introduced morphological matrices as a means to represent simple decision-making problems, such as concept selection (see, for example, Table 7.10). We also used Design Structure Matrices in Part 2, mostly to represent the interfaces between entities of a system. In Part 4, we focus on the other aspects of decision support: structuring, simulating, and viewing.

We start this section by revisiting morphological matrices and DSMs, showing that these tools provide limited support to structuring, and no support to simulation and viewing. Then we introduce decision trees as a widely used decision support tool for decision making under

uncertainty, providing some support to the structuring and simulating layers. (Other widely used decision support tools, such as Markov Decision Processes, are not described in this section because they are seldom used for system architecture purposes.)

Morphological Matrix

The morphological matrix was introduced in Chapter 7 as a way to represent and organize decisions in a tabular format. The morphological matrix was first defined by Zwicky as a part of a method for studying the "total space of configurations" (morphologies) of a system. [22] Since then, the use of morphological matrices as a decision support tool has grown. [23] Figure 14.1 includes the morphological table for the Apollo example.

A morphological matrix lists the decisions and associated alternatives, as shown in Figure 14.1. An architecture of the system is chosen by selecting one alternative (labeled "alt") from the row of alternatives listed to the right of each decision. Note that alternatives for different decisions do not have to come from the same column. For example, a configuration of the Apollo system could be: number of EOR = no (alt A), LOR = yes (alt B), command module crew = 2 (alt A), lunar module crew = 2 (alt C). An example of an expanded or more explicit form of the morphological matrix was given in Table 7.10.

In terms of decision support, the morphological matrix is a useful, straightforward method for representing decisions and alternatives. It is easy to construct and simple to understand. However, a morphological matrix does not represent metrics or constraints between decisions. Thus it does not provide tools for structuring a decision problem, simulating the outcome of decisions, or viewing the results.

Design Structure Matrix

As introduced in Chapter 4, a Design Structure Matrix is actually a form of decision support. The term Design Structure Matrix (DSM) was introduced in 1981 by Steward. [24] DSMs are now widely used in system architecture, product design, organizational design, and project management.

A DSM is a square matrix that represents the entities in a set and their bilateral relationships (see Table 4.4). These entities can be the parts of a product, the main functions of a system, or the people in a team, as demonstrated in the different examples throughout Part 2.

When a DSM is used to study the interconnections between decisions, each row and column corresponds to one of the N decisions, and an entry in the matrix indicates the connections, if any, that exist between the two decisions. The connections could be logical constraints or "reasonableness" constraints, or they could be connections through metrics.

For example, in Table 14.3, the letter a at the intersection between "EOR" and "earthLaunch" indicates that there is a connection between these two decisions as the result of the constraint labeled "a" in Table 14.1. Likewise, the letters b through f indicate the other five constraints. A blank entry in the intersection indicates that there is no direct connection between these decisions imposed by the constraints.

A connection could mean that a metric depends on those two variables. For example, Table 14.4 shows the connection between the decisions given by the IMLEO metric. The entries in the matrix may have different letters, numbers, symbols, or colors to indicate different types of connections.

TABLE 14.3 | DSM representing the interconnection of decisions by logical constraints for the Apollo case

		1	2	3	4	5	6	7	8	9
EOR	1		a							
earthLaunch	2	a								
LOR	3				b	c		e		f
moonArrival	4			b						
moonDeparture	5			c						
cmCrew	6							d		
lmCrew	7			e			d			
smFuel	8									
imFuel	9			f						

TABLE 14.4 | DSM representing the interconnection of decisions by the IMLEO metric in the Apollo case

		1	2	3	4	5	6	7	8	9
EOR	1									
earthLaunch	2									
LOR	3						I	I	I	I
moonArrival	4									
moonDeparture	5									
cmCrew	6				I			I	I	I
lmCrew	7				I				I	I
smFuel	8				I			I		I
imFuel	9				I			I	I	

In Table 14.5, the DSM of Table 14.3 has been partitioned and sorted so as to minimize interactions across blocks of decisions when possible. These blocks represent sets of decisions that should be made approximately simultaneously, because they have couplings with each other (for example, LOR with Lunar Module crew, Service Module crew, Service Module fuel, Lunar Module fuel). A partitioning procedure and example are given in Steward's papers. More generally, clustering algorithms can be used for partitioning (see Appendix B).

A DSM provides information in the representing layer and the structuring layer of decision support. It represents the decisions (but not their alternatives) and their interconnections. In combination with partitioning or clustering algorithms, a DSM can be sorted to show which sets of decisions are tightly coupled and which sets are less tightly coupled or not coupled. This information informs both system decomposition and the timing of architectural decisions, if the order of the decisions in the matrix is taken to represent the sequence in which they are made. [25]

TABLE 14.5 | Sorted DSM of the interconnections of decisions by logical constraints for the Apollo case

		1	2	3	4	5	6	7	9	8
EOR	1		a							
earthLaunch	2	a								
LOR	3				b	c		e	f	
moonArrival	4			b						
moonDeparture	5			c						
cmCrew	6							d		
lmCrew	7			e			d			
imFuel	9			f						
smFuel	8									

Decision Trees

A *decision tree* is a well-known way to represent sequential, connected decisions. A decision tree can have three types of nodes: decision nodes, chance nodes, and leaf nodes. Decision nodes represent decisions, which are controllable by the decision maker and have a finite number of possible assignments represented by branches in the tree from that node. Chance nodes represent chance variables, which are not controllable by the decision maker and also have a finite number of possible assignments, which are also represented by branches. The endpoints, or "leaf" nodes, in the decision tree represent a complete assignment of all chance variables and decisions.

When these decisions are architectural decisions, a path through the decision tree essentially defines an architecture. Thus, decision trees without chance nodes can be used to represent different architectures. Figure 14.4 shows a decision tree for the Apollo example concerning the five decisions related to the mission-mode. All nodes in this chart are decision nodes except for the final node in each branch, which is a leaf node. The first three of the constraints of Table 14.1 have also been implicitly included in the tree; for example, if earthLaunch = yes, there is no option of EOR = yes. This branch has been "pruned" from the tree by applying the logical constraint.

Note that this decision tree representation explicitly enumerates all the combinations of options (that is, all architectures), whereas the morphological matrix enumerated architectures only implicitly, by showing the alternatives for each decision. A limitation of decision trees is immediately visible in Figure 14.4: The size of a decision tree grows rapidly with the number of decisions and options, which results in huge trees, even for modest numbers of decisions.

In addition to representing architectures, decision trees can be used for evaluating architectures and selecting the best ones. This requires computing the metrics for each architecture. Sometimes, it is possible to compute a metric incrementally while following a path in the tree. For example, the probability-of-success metric in the Apollo example can be computed in this way, because the contribution of each decision to the overall probability of success is given by Table 14.2, and the metric is constructed by multiplying the individual contributions together.

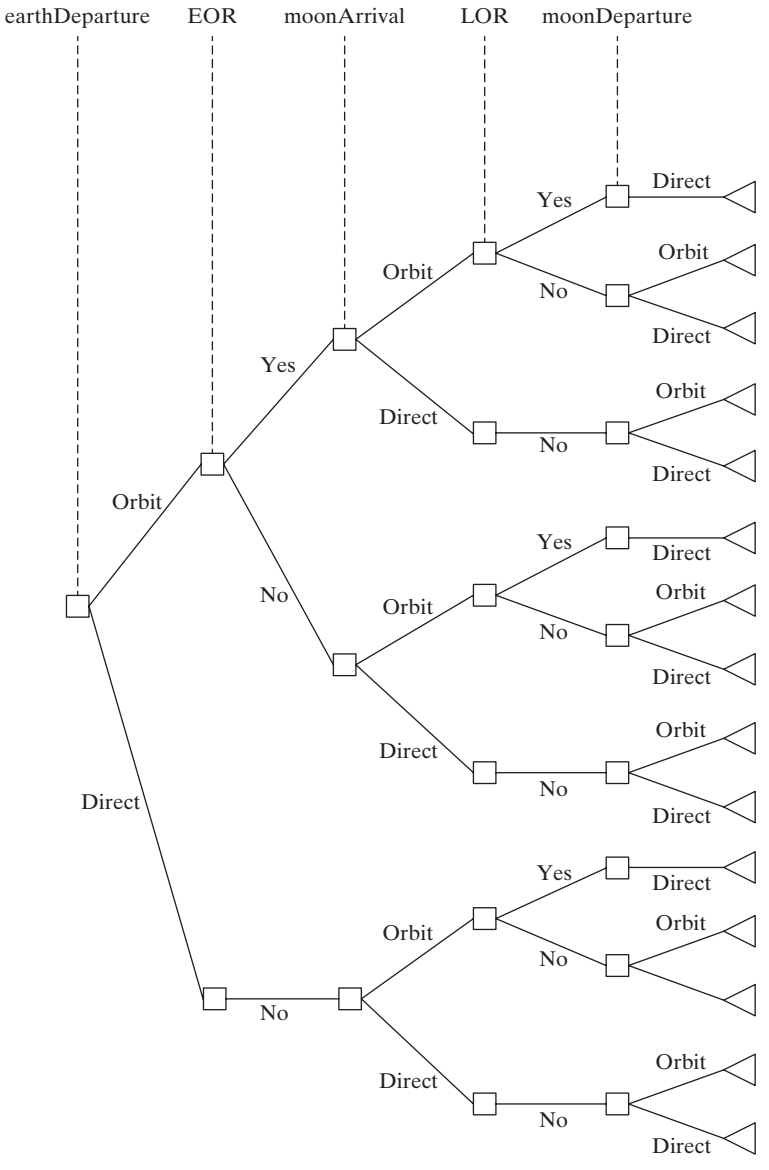

FIGURE 14.4 Simple decision tree with only decision nodes and leaf nodes (no chance nodes) for the Apollo mission-mode decisions.

In most cases however, there is coupling between decisions and metrics that is not additive or multiplicative, such as in the case of the IMLEO metric, a highly nonlinear function of the decision alternatives. In these cases, the metric for every leaf node (such as IMLEO) is usually computed after all the decision alternatives have been chosen. This approach may be impractical in cases where the number of architectures is very large, as we will see in Chapter 16.

The leaves or architectures represented on decision trees are usually evaluated using a single metric. In such cases, it is customary to combine all relevant metrics (such as IMLEO and probability of success p) into a single metric representing the utility of an architecture for stakeholders [such as $u = \alpha u\,(IMLEO) + (1 - \alpha)\,u(p)$]. The weight α and the individual utility functions $u\,(IMLEO)$ and $u(p)$ can be determined by means of multi-attribute utility theory. [26]

In decision trees that don't have chance nodes, choosing the best architecture is straightforward. It is more complicated in the presence of chance nodes, because leaf nodes no longer represent architectures but, rather, combinations of architectures and scenarios. It is thus necessary to choose the architecture that has the highest expected utility by working backwards from the leaf nodes. Let's illustrate that with an example.

Assume that, instead of having a single value for the risk factor of each decision in Table 14.2, we had two values: an optimistic one and a pessimistic one. What is the best architecture given this uncertainty? Figure 14.5 shows the addition of the chance nodes corresponding to the last decision in the tree for the Apollo case (only a very small part of the tree is actually shown). Note that the actual tree has effectively doubled in size just through the addition of chance nodes corresponding to the *moonDeparture* decision. We compute the risk metric for all the new leaf nodes by traversing the tree from start to finish and multiplying all the risk factors together (the results are the R_i values in Figure 14.5). At this point, we can't simply choose the leaf node with the lowest risk, because it does not represent an architecture; it represents an architecture for a given scenario—that is, a value of the risk factors. Instead, we need to assign probabilities to the branches of the chance nodes (such as 50%-50% everywhere), and compute the expected value of the risk metric at each chance node. Then, it makes sense to choose the Moon departure option with the lowest risk at each decision node. If we add chance nodes to all the other decisions, we can follow the same two steps. First, we compute the expected risk factor at each chance node, and then we pick the option with the best risk metric. We perform these steps for the *LOR* decision, then the *moonArrival* decision, and so forth, until all decisions have been made. This process yields the architecture that has best performance on average across all scenarios. This architecture is given by the combination of all the optimal decisions.

In short, the best architecture in a decision tree can be found by applying these two steps (expectation for chance nodes and maximization/minimization for decision nodes), starting from the leaf nodes and going backwards through the tree. [27]

Even though they are commonly used as general decision support tools, decision trees have several limitations that make them impractical to use in large system architecture problems. First, they require pre-computation of a *payoff matrix* containing the utilities of all the different options for each decision for all possible scenarios, which may be impractical for many problems. More generally, their size grows exponentially with the number of nodes in the problem.

Second, the method assumes that the payoffs and probabilities for a given decision are independent from the rest of the decisions, which is often unrealistic. For example, consider that instead of modeling the uncertainty in the risk factors, we wish to model the uncertainty in the ratio between propellant mass and structure mass for any of the vehicles used in the architecture. This is a very important parameter of the model, because it can drive the IMLEO metric. However, this parameter does not directly affect any of the decisions the way the risk factor did. It affects the IMLEO metric through a complex mathematical relationship based on the mission-mode

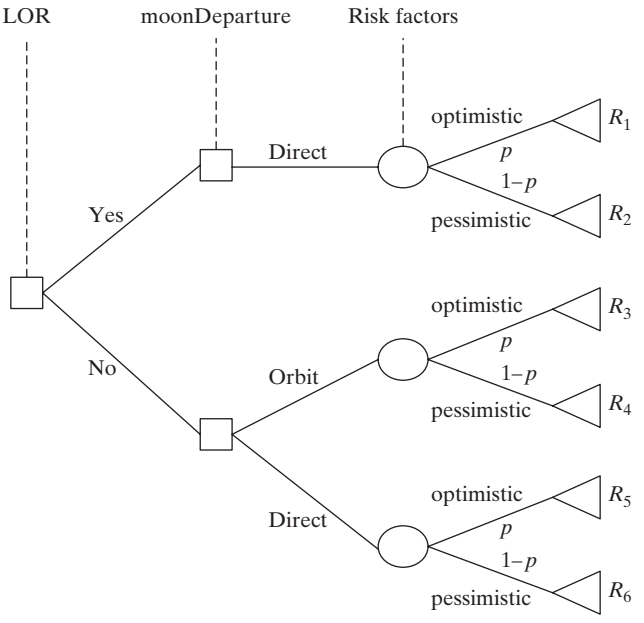

FIGURE 14.5 Fragment of the Apollo decision tree, including a chance node for the risk factors. In this tree, decision nodes are squares, chance nodes are circles, and leaf nodes are triangles. R_i are the values of the risk metrics for each architecture fragment, and p is the probability of being in an optimistic scenario for each individual decision. (Note that these probabilities could in general be different for different chance nodes.)

decisions. This ratio is thus a "hidden" uncontrollable variable that affects multiple decisions. Modeling this in a decision tree is very difficult, or else it requires collapsing all the mission-mode decisions into a single decision with 15 options (the number of leaf nodes in Figure 14.4).

In summary, decision trees provide a representation of the decision and can represent some kinds of structure, but they are of limited use in simulation.*

14.6 Decision Support for System Architecture

In the previous section, we discussed how three standard decision support tools can be applied to system architecture. At this stage, the reader may wonder whether there is anything special about system architecture decisions at all. How are architecture decisions different from other decisions?

Box 14.1 summarizes the characteristics of architectural decisions and metrics. The combination of these characteristics makes existing decision support tools very hard to apply to

*Decision networks are a more general version of decision trees where the tree structure condition is relaxed to allow arbitrary topologies between decision nodes, chance nodes, and leaf nodes.

Box 14.1 Insight: Properties of Architectural Decisions and Metrics

The following properties characterize architectural decisions and metrics and differentiate them from other types of decisions.

- **Modeling breadth versus depth.** Architecture decision support focuses on modeling breadth—that is, analyzing a large space of very different architectures at relatively low fidelity—whereas design decision support focuses on modeling depth—that is, analyzing a smaller number of designs with higher fidelity. Note that there is an inherent tradeoff between modeling breadth and modeling depth.
- **Ambiguity.** Ambiguity is present in both design and architecture problems. However, architecture problems suffer from larger and more varied sources of uncertainty (unknown outcomes due to random events) and ambiguity (inaccuracies or fuzziness in statements), simply because they occur early in the development process. This often makes the use of typical probabilistic techniques such as Monte Carlo simulation impractical or inadequate for system architecture. [28]
- **Type of variable.** In decision analysis and optimization, decisions are classified into three types: continuous, discrete (those that can take only integer values), and categorical (those that can take any value from a discrete set of symbols representing abstractions). [29] Architectural decisions are most often categorical variables, are sometimes discrete variables, and are rarely continuous variables. The reason is that architectural decisions often consist of choosing among different entities of form or function, or among different mappings between function and form, which are inherently categorical in nature. We will have a deeper discussion about classes of architecture decisions and how to model them in Chapter 15. For example, the Apollo mission-mode decision is really about choosing which vehicles (form) perform which maneuvers (function). Conversely, design decisions tend to be continuous, are sometimes discrete, and are less likely to be categorical.
- **Subjectivity.** Architecture problems often deal with some subjective metrics, which reflect the ability of the system to provide value to the stakeholders. Dealing with subjective metrics leads to the use of techniques such as multi-attribute utility theory or fuzzy sets. [30] Furthermore, subjectivity highlights the need for traceability. Architecture decision support should provide an explanation that justifies the assessments and can trace it back to expert knowledge or judgment if necessary.
- **Type of objective functions.** Architecture problems sometimes use relatively simple equations in their objective functions, but often they also tend to resort to simple look-up tables and if-then structures, such as the Apollo risk metric. This is because of (1) the breadth of architectures considered, which results in the need to apply different strategies to evaluate different types of architectures; (2) the low modeling fidelity, which leads to replacing complex computations with heuristics; and (3) the subjectivity of some metrics, which can be captured by rules that "set" some metric of interest for an architecture when a number of conditions are met.
- **Coupling and emergence.** Architecture decision problems typically have a relatively low number of decisions compared to design problems. However, the dimensionality of the corresponding architecture space is often extremely large due to combinatorial explosion. What this means is that architecture variables are often extremely coupled. This is important because it precludes the utilization of some very powerful methods that exploit decouplings in the structure of decision problems (for example, the Markov property in dynamic programming techniques).

architecture decision support, and this has motivated the development of decision support tools for system architecture. [31]

In response to the characteristics listed in Box 14.1, architecture decision support systems are more interactive and less automatic, and they often employ tools from the fields of knowledge reasoning and engineering (such as knowledge-based systems for incorporation of expert knowledge and explanation).

14.7 Summary

We began Part 4 by introducing a key idea: System architecting is a decision-making process, so we can benefit from decision support tools. We started off with an example based on the Apollo program to illustrate how we formulate a system architecting problem as a decision-making problem. We argued that the goal of these tools is to support—not replace—the system architect, given that system architecting requires creativity, holism, and heuristic approaches for which humans are much better suited than machines.

We described four fundamental aspects of decision support systems (representing, structuring, simulating, and viewing), and we showed how Parts 2 and 3 have used decision support tools mostly for representing, but not for structuring, simulating, or viewing. We discussed three basic decision support tools that support representing and that provide some limited structuring, simulating, and viewing capabilities. These are: morphological matrices, design structure matrices, and decision trees. We discussed in particular some of the limitations of these tools for system architecture.

This led us to ask about the differences between system architecture and other decision-making processes. We concluded that although some of the standard decision support tools remain applicable, there is a need for tools that can deal with subjectivity, ambiguity, expert knowledge, and explanation. The next chapters will discuss some aspects of the tools we use for the four aspects of decision support in the context of system architecture. We will first take a deeper look at architectural tradespaces like the one in Figure 14.2. We will be working mostly in the structuring layer of the Design Activity, and we will see how to obtain useful knowledge about the system architecture by applying simple data processing techniques to tradespaces.

References

[1] J.F. Kennedy, "Special Message to the Congress on Urgent National Needs." Speech delivered in person before a joint session of Congress, May 25, 1961.

[2] See I.D. Ertel, M.L. Morse, J.K. Bays, C.G. Brooks, and R.W. Newkirk, *The Apollo Spacecraft: A Chronology,* Vol IV, NASA SP-4009 (Washington, DC: NASA, 1978) or R.C. Seamans Jr., *Aiming at Targets: The Autobiography of Robert C. Seamans Jr.* (University Press of the Pacific, 2004).

[3] T. Hill, "Decision Point," *The Space Review,* November 2004.

[4] J.R. Hansen, *Spaceflight Revolution: NASA Langley Research Center from Sputnik to Apollo,* NASA SP-4308 (Washington, DC: NASA, 1995).

[5] W. von Braun, F.L. Whipple, and W. Ley, *Conquest of the Moon* (Viking Press, 1953).

[6] J.C. Houbolt, "Problems and Potentialities of Space Rendezvous," Space Flight and Re-Entry Trajectories. International Symposium Organized by the International Academy of Astronautics of the IAF Louveciennes, June 19–21, 1961, *Proceedings,* 1961, pp. 406–429.

[7] W.J. Larson and J.R. Wertz, eds., *Space Mission Analysis and Design* (Microcosm, 1999).

[8] V.A. Chobotov, ed., *Orbital Mechanics* (AIAA, 2002).

[9] J.C. Houbolt, *Manned Lunar-landing through Use of Lunar-Orbit Rendezvous*, NASA TM-74736, (Washington, DC: NASA, 1961).

[10] J.R. Hansen, *Enchanted Rendezvous*, Monographs in Aerospace History Series 4 (Washington, DC: NASA, 1999).

[11] C. Murray and C.B. Cox, *Apollo* (South Mountain Books, 2004).

[12] J.N. Wilford, "Russians Finally Admit They Lost Race to Moon," *New York Times*, December 1989.

[13] Hoffman, R. R., "Decision Making: Human-Centered Computing," *IEEE Intelligent Systems*, vol. 20, 2005, pp. 76–83.

[14] "Decision," *American Heritage Dictionary of the English Language*, 2004.

[15] S.O. Hansson, *Decision Theory: A Brief Introduction* (KTH Stockholm, 1994).

[16] H.A. Simon, *The New Science of Management Decision.* The Ford Distinguished Lectures, Vol. 3, (New York: Harper & Brothers, 1960).

[17] E. Turban and J.E. Aronson, *Decision Support Systems and Intelligent Systems* (Prentice Hall, 2000).

[18] C. Barnhart, F. Lu, and R. Shenoi, R., "Integrated Airline Schedule Planning," *Operations Research in the Airline Industry*, Vol. 9, Springer US (1998), pp. 384–403.

[19] C. Alexander, *A Pattern Language: Towns, Buildings, Construction* (Oxford University Press, 1977).

[20] D. Power, *Decision Support Systems: Concepts and Resources for Managers* (Greenwood Publishing Group, 2002), pp. 1–251; R.H. Bonczek, C.W. Holsapple, and A.B. Whinston *Foundations of Decision Support Systems* (Academic Press, 1981); and E. Turban, J.E. Aronson, and T.-P. Liang, *Decision Support Systems and Intelligent Systems* (Upper Saddle River, NJ: Prentice Hall, 2005).

[21] J. Banks, J.S. Carson II, B.L. Nelson, and D.M. Nicol, *Discrete-Event System Simulation* (Prentice Hall, 2009), pp. 1–640 and B.P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation* (Academic Press, 2000), pp. 1–510.

[22] F. Zwicky, *Discovery, Invention, Research through the Morphological Approach* (Macmillan, 1969).

[23] G. Pahl and W. Beitz, *Engineering Design: A Systematic Approach* (Springer, 1995), pp. 1–580; D.M. Buede, *The Engineering Design of Systems: Models and Methods* (Wiley, 2009), pp. 1–536; and C. Dickerson and D.N. Mavris, *Architecture and Principles of Systems Engineering* (Auerbach Publications, 2009) pp. 1–496. See also T. Ritchey, "Problem Structuring Using Computer-aided Morphological Analysis," *Journal of the Operational Research Society* 57 (2006): 792–801.

[24] D.V. Steward, "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Transactions on Engineering Management* 28 (1981): 71–74.

[25] S.D. Eppinger and T.R. Browning, *Design Structure Matrix Methods and Applications* cambridge. MA: The MIT Press, 2012), pp. 1–352.

[26] Multi-attribute utility theory was developed by economists Keeney and Raifa in the seventies. See: R.L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-Offs* (New York: Wiley, 1976), p. 592. Since then, it has been widely applied in systems engineering. See also: A.M. Ross, D.E. Hastings, J.M. Warmkessel, and N.P. Diller, "Multi-Attribute Tradespace Exploration as Front End for Effective Space System Design," *Journal of Spacecraft and Rockets* 41, no. 1 (2004): 20–28.

[27] An example of the application of this algorithm can be found in C.W. Kirkwood, "An Algebraic Approach to Formulating and Solving Large Models for Sequential Decisions under Uncertainty," *Management Science* 39, no. 7 (1993): 900–913.

[28] Alternative tools for dealing with ambiguity include again fuzzy sets, or simple interval analysis. See, for example, J. Fortin, D. Dubois, and H. Fargier, "Gradual Numbers and Their Application to Fuzzy Interval Analysis," *IEEE Transactions on Fuzzy Systems* 16, no. 2 (2008): 388–402. An application of interval analysis to system architecture appears in D. Selva and E. Crawley, "VASSAR: Value Assessment of System Architectures Using Rules," in Aerospace Conference, 2013 at Big Sky.

[29] Note that categorical variables are often implemented as integer variables in optimization problems, but that does not make them discrete variables. One cannot define a gradient-like concept in categorical variables unless some meaningful distance metric is defined within the discrete set of abstractions, which is often not trivial.

[30] The foundational paper on multi-attribute utility theory is R.L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-Offs* (New York: Wiley, 1976), p. 592, and an example of a recent application to system architecture can be found in A.M. Ross, D.E. Hastings, J.M. Warmkessel, and N.P. Diller, "Multi-Attribute Tradespace Exploration as Front End for Effective Space System Design," *Journal of Spacecraft and Rockets* 41, no. 1 (2004): 20–28. Concerning fuzzy sets, the foundational paper is L.A. Zadeh, "Fuzzy Sets," *Information and Control* 8, no. 3 (1965): 338–353. An application to conceptual design can be found in J. Wang, "Ranking Engineering Design Concepts Using a Fuzzy Outranking Preference Model," *Fuzzy Sets and Systems* 119, no. 1 (2001): 161–170.

[31] See, for example, Koo's development of the Object Process Network meta-language for systems architecture, focusing on the simulating layer: B.H.Y. Koo, W.L. Simmons, and E.F. Crawley, "Algebra of Systems: A Metalanguage for Model Synthesis and Evaluation," *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans* 39, no. 3 (2009): 501–513. Simmons's Architecture Decision Graph is a related tool that focuses on the structuring layer: W.L. Simmons, "A Framework for Decision Support in Systems Architecting" (PhD dissertation, Massachusetts Institute of Technology). Selva's VASSAR methodology is a third example, focusing on the structuring and simulating layers: D. Selva and E. Crawley, "VASSAR: Value Assessment of System Architectures using Rules," in Aerospace Conference, 2013 at Big Sky.