ENGINEERING SYSTEMS MONOGRAPH

THE INFLUENCE OF ARCHITECTURE IN ENGINEERING SYSTEMS

The ESD Architecture Committee

*Edward Crawley, Olivier de Weck, Steven Eppinger, Christopher Magee,
Joel Moses, Warren Seering, Joel Schindall, David Wallace, Daniel Whitney (Chair)*

MIT esd

March 29-31, 2004

THE INFLUENCE OF ARCHITECTURE IN ENGINEERING SYSTEMS

The ESD Architecture Committee

Edward Crawley, Olivier de Weck, Steven Eppinger, Christopher Magee, Joel Moses, Warren Seering, Joel Schindall, David Wallace, Daniel Whitney (Chair)

ABSTRACT

The field of Engineering Systems is distinguished from traditional engineering design in part by the issues it brings to the top. Engineering Systems focuses on abstractions like architecture and complexity, and defines system boundaries very broadly. It also seeks to apply these concepts to the process of creating systems. This paper summarizes the role and influence of architecture in complex engineering systems. Using the research literature and examples, this paper defines architecture, argues for its importance as a determinant of system behavior, and reviews its ability to help us understand and manage the design, operation, and behaviors of complex engineering systems.

A. INTRODUCTION

Typical engineering design education focuses on specific aspects of design, such as the technical behavior of a set of elements interconnected in a certain way. By contrast, Engineering Systems focuses on a number of abstract concepts first because they provide a general framework for guiding the development of many diverse kinds of systems, so that these systems will provide the desired functions in the desired ways. Among these abstract concepts is that of system architecture. In this paper, we explore this concept and provide a number of ways of appreciating system architecture’s importance in both the practical aspects of system design and in the intellectual aspects of understanding complex systems from a variety of viewpoints.

The paper begins with a definition of architecture and its influence on functional behavior, extra desired properties like flexibility and reliability (collectively called “ilities”), complexity, and emergent behaviors. Architectures are not static but instead evolve over long periods as technologies mature. They also evolve during the normal course of designing an individual system. These evolutionary patterns are useful in understanding architecture’s importance.

The paper next provides several examples of architectures and illustrates how architecture affects the way systems are designed, built, and operated. The examples include aircraft, automobiles, infrastructures, and living organisms.

The importance of architecture is framed in three domains of importance: as a way to understand complex systems, to design them, to manage them, and to provide long-term rationality by means of standards. The abstract concepts of modularity and integrality are shown to be useful for categorizing systems and illustrating how architectural form can influence important system characteristics. Several contrasts are noted between relatively small, deliberately designed products and evolutionary, less-managed large infrastructures.

Architecture’s ability to influence the functions and allied properties of systems is shown to extend to robustness, adaptability, flexibility, safety, and scalability. Examples from recent research are given to show how some of these properties might be measured using network models of particular architectures.

Finally, the paper identifies a number of important near- and long-term research challenges regarding the potential for understanding architecture to the point where a system's behaviors can be completely determined. Among the barriers are complexity, bounded human rationality, and human agency.

B. WHAT IS SYSTEM ARCHITECTURE?

System architecture is an abstract description of the entities of a system and the relationships between those entities. Architecture is important in most technical fields, including not only civil architecture of buildings but of physical products, software, computer networks, large engineering systems, and infrastructures. The architecture of a system has a strong influence on its behavior. Every system has an architecture. Architectures may arise in the process of deliberate *de novo* design of a system; by evolution from previous designs with strong legacy constraints; by obeying regulations, standards, and protocols; by accretion of smaller systems with their own architectures; or by exploration of form and behavioral requirements via dialogue between users and architects, to name a few known mechanisms. While natural architectures may hold lessons for us, including the influence of evolution under constraints, we are mainly concerned here with man-made architectures of complex engineering systems.

Man-made system architectures are created as part of the process of creating and designing systems. These systems are intended to have certain primary functions, plus other properties that we call "ilities:" durability, maintainability, flexibility, and so on. The primary functions have immediate value while the ilities tend to have life-cycle value. Like the ilities, the architectures themselves are long-lived either because they determine the design of several generations of products or because the resulting systems are themselves long-lived. In most cases, it is very challenging to design a complex system that achieves all of its primary functions and all of its ilities. In some instances one has to resolve tradeoffs between desirable properties for the short term versus desirable life-cycle properties. An example is the life-cycle property of extensibility, which might require including interfaces for future system elements that are not present in the original version. Such interfaces must be designed, will generally require additional resources, and might increase initial system complexity. The benefits of such architectural decisions are uncertain and might only be realized in the future, or not at all. Methods for evaluating uncertain events and providing for them in advance are discussed in De Neufville (2004).

Some systems, such as familiar products of industry (cars, aircraft, computers) are designed according to a deliberate process that includes carefully thinking through what their architectures should be, although the thinking and the resulting architectures can vary widely in appropriateness for their intended purposes. Other systems, such as large infrastructures, may grow by accretion or annexation from smaller systems. Even when the smaller systems have deliberate and well-conceived architectures, the resulting agglomerated system may not have a cohesive or consistent architecture, a fact that may inhibit the system's ability to function. Regional electric power grids are an example.

Complex systems have behaviors and properties that no subset of their elements have. Some of these are deliberately sought as the product of methodical design activity. While achieving these behaviors, the designers often accept certain undesirable behaviors or side effects, In addition, systems have unanticipated behaviors commonly called emergent. Emergent behaviors may turn out to be desirable in retrospect, or they may be undesirable. Emergent behaviors are similar to incidental interactions identified in Ulrich and Eppinger (2000). Automobiles not only enabled personal transport but revolutionized society in many unexpected ways, such as growth of suburbs and shopping malls, courtship habits, and a sense of personal freedom. Other examples of this are in Table 1.

	Anticipated	**Emergent**
Desirable	Electric power networks share the load. Hub-spokes airline routes shorten the length of trips.	Blackouts are associated with increased births. Hub-spokes plus waiting time creates a business opportunity in airport malls.
Undesirable	Power networks can propagate blackouts. Hub-spokes causes huge swings in workload and resource utilization at airports.	Blackouts are associated with increased births. Airport operators become dependent on mall rental income, making it difficult to modify airline route structures.

Table 1: Examples of Desirable and Undesirable Anticipated and Emergent System Properties Influenced by Architecture

The desirability/undesirability as well as the anticipated/emergent nature of these examples are debatable and are offered for discussion purposes only. (Hub-spokes example from Allen, Nightingale, and Murman 2004.)

Finally, the architecture of a system is an important determinant of its complexity, for good or ill. Sometimes, architectures are designed or evolve to minimize complexity, but, as systems grow in size, a point is usually reached where a system's complexity becomes overwhelming, imposing a limit on what one can do to operate the system, predict its behavior, or change it. Many systems gain both their benefits and their vulnerabilities from what would appear to be complexity, such as the interconnections in the nation's electrical grid. These interconnections permit power to flow from regions with excess to those with shortages, a common occurrence. If each region had its own grid, there would be no way to share the load. But exactly the same complexity works in the opposite direction as well. When the shortage in one region is too great and that region's grid breaks down, this breakdown can propagate along the same connections and bring down other parts of the grid that have no problems. Empirical evidence for the influence of complexity is given by Sterman (2000), who describes instances of "policy resistance:" systems whose behavior gets worse as people doggedly apply what they think is the correct policy.

The above points are summarized in Figure 1.

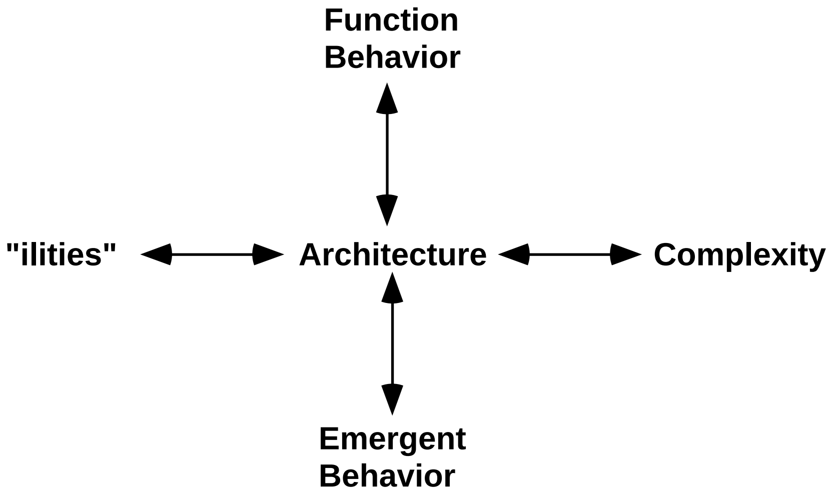

Figure 1: Architecture Plays a Central Role in Giving a System Its Behavior and "Ilities," as Well as Generating Emergent Behavior and Complexity

System engineering theory teaches designers to create a hierarchy of functions and physical objects. In most cases, these are system behaviors or characteristics, such as safety, handling, or fuel economy, that are visible to the customer. During system design, the requirements of upper levels in the hierarchy are decomposed and flowed down to the lower levels. This is intended to create separate manageable pieces that can be worked on independently. Carried to its extreme, this process is called “reductionism.” Major challenges include remembering all the requirements, keeping them consistent, and understanding the many interactions between branches of the hierarchy. These interactions cause problems during integration at the end of product development and challenge the basic assumptions underlying reductionism.

System engineering theory works most smoothly when the product can be broken into modules that are relatively independent. Such products or systems are called modular. When products cannot be decomposed simply, or when their behaviors interact, they are called integral. There are good technical reasons, such as weight or energy efficiency, that some systems must contain some measure of integrality. Correspondingly, there are good non-technical reasons, such as the ability to upgrade or customize a product, that systems must also contain some measure of modularity. A continuing tension in system design is the result.

More generally, reductionism relies on the assumption that a divide-and-conquer strategy will really work, that understanding the behavior of each element and defining each interface correctly and completely will assure a properly working system. This assumption brings with it a host of other attitudes and methods, generally called top-down, that assume that things can be preplanned and scripted, and that following the script is the way to get a successful result.

In contrast to top-down is bottom-up, in which requirements and system design are expected to emerge over time and by means of trial and error. Under these assumptions, no complete script can be written, not all of the events and decisions can be anticipated or scheduled, and the final result is unknown. On this basis, a step-by-step design process beginning with definition of the architecture is impossible. At the very least, the early steps will be revisited. Even if the design of each individual system proceeds more or less top-down, the architecture of an industry or class of system has historically evolved on a bottom-up basis. This is especially true in complex situations. A nuanced procedure for balancing these approaches is given by Cutcher-Gershenfeld, Field, Hall, Kirchain, Marks, Oye, and Sussman (2004).

Creating an architecture is often called architecting. Rechtin’s description of this process broadly represents the *de novo* design situation plus intense dialogue between architect and customer (Rechtin 1990). It involves determining what the system is supposed to do and how specifically it will do it. Rechtin’s view of architecting is similar to the embodiment stage (Pahl and Beitz 1991) in classical engineering design, when functions are expressed as objects arranged in space so that they can accomplish the desired functions. Achievement of the ilities is also a factor during the embodiment stage, according to Rechtin and Pahl and Beitz. The process of generating form from function usually does not deliver a unique form. The final choice is guided by application of the principles of engineering design as well as by the desire to conform to the ilities. Furthermore, in traditional engineering design, the ilities are derived from careful consideration of customers or users of the system. In the enterprise view of complex engineering systems (Allen, Nightingale and Murman 2004), Ilities also arise from enterprise culture and values, such as the safety-consciousness of Volvo.

The process of creating an architecture often follows a process of decomposition, in which a top-level concept of the system’s required functions is broken down into subfunctions, and at the same time the most abstract version of its physical form is broken down into subsystems capable of performing the subfunctions. The process of decomposition continues in this way until single parts are reached. Decomposition at lower levels often can be accomplished only by choosing particular concepts or instantiations of the system. System design does not always follow a relentless top-down decomposition process to the single-part level but may stop partway when standard or previously used items are adopted in full. It is also frequently true that architects and designers iterate between upper and lower levels of the system decomposition as the designers learn more about the implications of their architectural decisions.

Within the architecting process, several types of architectures are involved (Levis 1999). These are

- The functional architecture (a partially ordered list of activities or functions that are needed to accomplish the system's requirements)
- The physical architecture (at minimum a node-arc representation of physical resources and their interconnections)
- The technical architecture (an elaboration of the physical architecture that comprises a minimal set of rules governing the arrangement, interconnections, and interdependence of the elements, such that the system will achieve the requirements)
- The dynamic operational architecture (a description of how the elements operate and interact over time while achieving the goals)

Furthermore, architectures may evolve over time. This is true of typical products like cars and aircraft and is particularly important with respect to long-lived systems like infrastructures. An infrastructure's evolution is governed by many factors, such as its geographic dispersal and its high societal impact. A system like the telephone network or the organization chart of a company begins simply, with a few elements and simple relationships. As it grows, elements are added, new complexities arise, and the system may have to be re-architected. Wise is the architect whose system can grow within the original rules, element types, and structural arrangements. Most companies must be repeatedly reorganized. Other systems that involve large investments in physical plant cannot be re-architected. The goal of the architect is to minimize the severe constraints imposed by legacy, some of which are highly disadvantageous.

The life history of a decomposition process, both for familiar products and long-lived infrastructures, comprises at least two evolutions. One evolution defines the architecture of an entire technological class of entities and follows the "S" curve (Utterback 1994). The other evolution follows a contingent series of choices during the design of a given entity in that class.

The "S" curve history starts with an interactive search by users and designers for requirements and matching architecture. A pure top-down process cannot succeed in the early phases of a technology or industry. Even after users and designers understand each other, systems evolve to meet requirements in new ways. In aircraft propulsion, the top-level architectural decision between piston (radial geometry) and turbine (axial geometry) mechanizations was made decades ago. We do not often go down the piston side of the choice tree any more. Within the turbine choice tree, many decisions are no longer looked at seriously because technology, materials, or safety considerations have cemented those decisions into place. Thus, as technologies mature, the active choices are pushed lower and lower, ultimately to the component level. Sometimes hybrid architectures, such as turboprops, will emerge to exploit new markets or to backfill market segments abandoned by earlier architecture/technology concepts. This reflects the Henderson-Clark (1990) breakdown of innovation (radical, architectural, component, and incremental). When a major/disruptive break occurs, it is necessary to undo choices at increasingly higher levels of the decomposition tree.

In an individual system design project, a decomposition is made in the form of a series of choices, any one of which represents a form-function realization, and any one of which could run into a serious problem. If the problem threatens the whole design, a series of decisions above must be revisited. This can be especially difficult if the budget or schedule available to these decisions has been exhausted. The inability to operate confidently at lower levels of this tree, free of the fear of the whole thing unraveling, constitutes one of the hazards of designing complex systems.

C. EXAMPLES OF ARCHITECTURES

Architecture comprises entities and the structure of relationships and interfaces between them. The relationships can be geometric, temporal, logical/informational, or based on exchange of matter, information, energy, or value. Architectures can describe end items like cars or highways, or they can describe patterns and rules by which such end items should be designed or built. Architectures must also contain a plan for how the system should be operated under nominal and off-nominal conditions.

Thus, aircraft architectures can be classified as fixed wing or rotary wing. Within fixed wing, we have straight or angled wings distinct from the fuselage, delta wings, and blended wing architectures. We also used to have mono-wing, bi-wing and tri-wing configurations, but the latter choices were gradually abandoned. Each of these is suited to a different class of operation, speed, flight pattern, payload, or flying characteristics. The architect chooses wing and body shapes in order to achieve the desired functions.

The architecture of wheeled vehicles began as "body on axles," the typical form of chariots and farm wagons. Sometime in the past, springs were added and the architecture became "body on frame separated by springs." This architecture persisted into the era of automobiles until the mid-1960s when the technology of steel stampings made reinforced shell bodies feasible, giving rise to the unitized body in which there is no separate frame. Sloan (1996) describes several intermediate forms that appeared and disappeared in the 1920s and '30s, such as radiator and rear seat over axles, and low-slung frame.

Jet engines have two dominant forms today, the low-bypass and high-bypass types. See Figure 2. The high-bypass type gains its thrust from a jet-driven propeller called the fan, and has a large frontal area and diameter. It is suitable for commercial planes when situated under the wings. The low-bypass type is more like a rocket that gains thrust by emitting hot gas out the back. It is long and thin and is suitable for mounting inside the fuselage of a combat jet. The cockpit of such a plane is mounted ahead of the engine. A high-bypass engine could not be mounted in this way on a fighter jet. It is also more difficult to rapidly change the RPM and thus the thrust of a high-bypass engine as is required by combat maneuvers. For a history of these engines, see Smith and Mindell (2000).

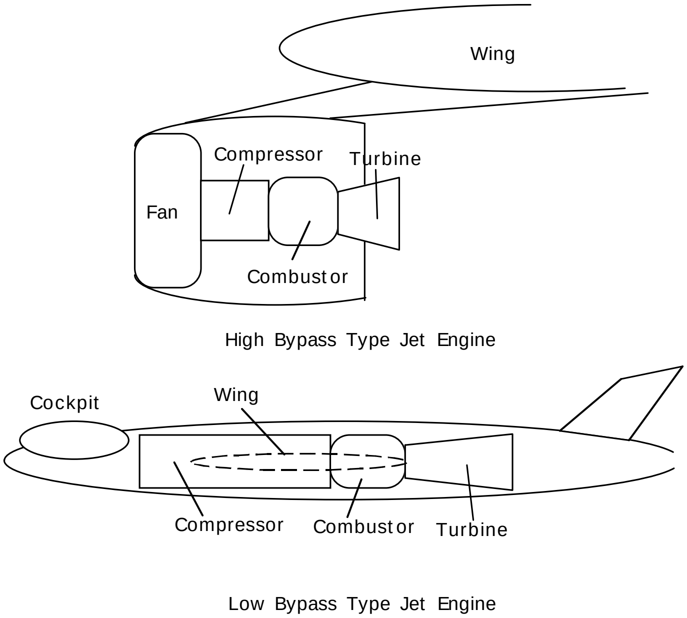

Figure 2: Schematic Comparison of Architectures of Two Types of Jet Engines

Aircraft architectures are also described by product families such as the Boeing 737-xxx series. Members of such a family share a common fuselage diameter and certain other characteristics such as control systems and cockpits, but may differ in fuselage or wing length. New members of the family must conform to certain rules of design and production, so that they can be made in the same factory and flown by the same pilots without undue retraining. These rules also comprise a kind of architecture—that of the product family itself. They comprise standards that govern some features of the entities (fuselage elements) as well as the interfaces (main fuselage joints) that tie them together.

Aircraft manufacturing also presents architectural choices. Figure 3 shows two methods for assembling aircraft fuselage and installing distributive and interior systems. Each has its advantages and disadvantages. "Trough and drape" is more feasible for small aircraft than for large, for which "build and join barrel sections" is commonly used.

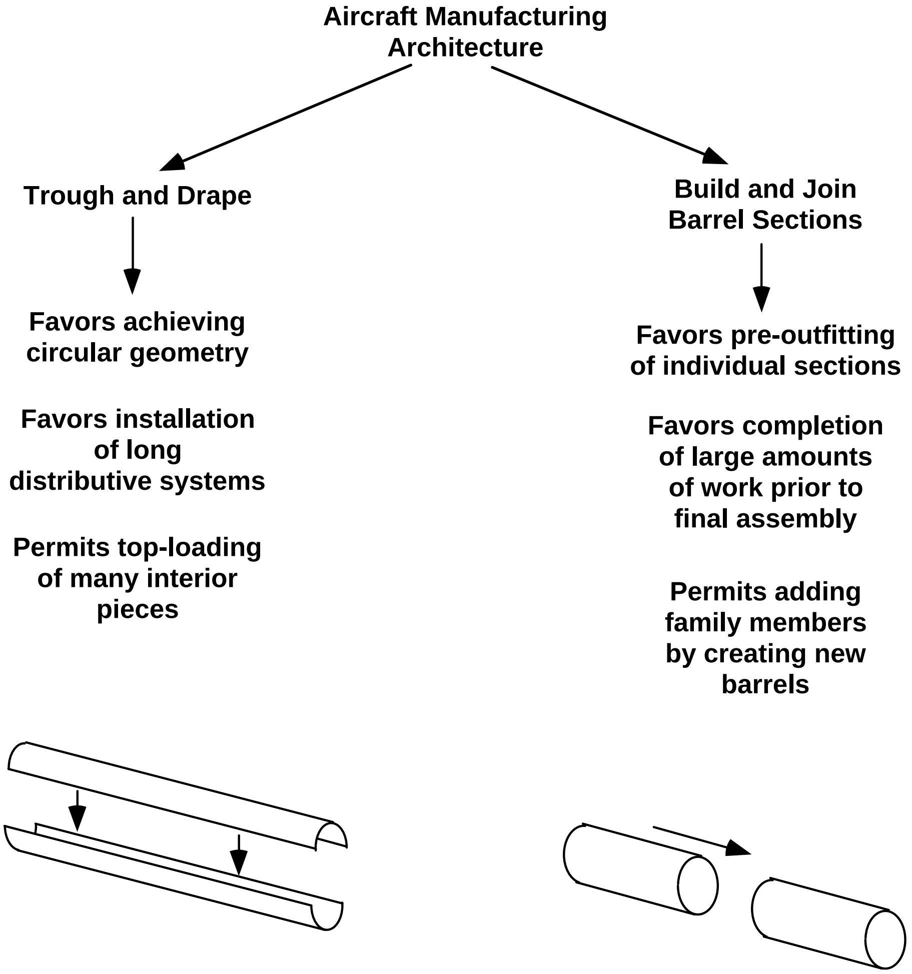
Aircraft Manufacturing Architecture
Trough and Drape
Favors achieving circular geometry
Favors installation of long distributive systems
Permits top-loading of many interior pieces
Build and Join Barrel Sections
Favors pre-outfitting of individual sections
Favors completion of large amounts of work prior to final assembly
Permits adding family members by creating new barrels

Figure 3: Alternatives for Building Aircraft Cylindrical Fuselage Sections

Another example of an architecture that governs the creation of multiple systems is the wired national telephone network. This system has three layers, each with its own architecture. At the local level, there is a central exchange that can link any of 9,999 phones to each other or can link one of them to a phone in a nearby exchange in the same region. Inter-regional trunks link the regions in the layer above the local layer. At the top is a layer of long distance links that can be switched based on current loads to create alternate routes for long distance calls between distant regions. Each layer has multiple connections within it and a few connections to the other layers.

Within the top two layers, it is possible to create many alternate paths for calls to follow. Thus, the system can be scaled to carry more calls and is robust against disruptions to major links and hubs.

Figure 4 is a map of the California electric transmission grid, showing the main high voltage lines. It is an accretion of the investments of at least nine separate entities. Connections, not shown, to other states obviously add links and loops to what is shown.

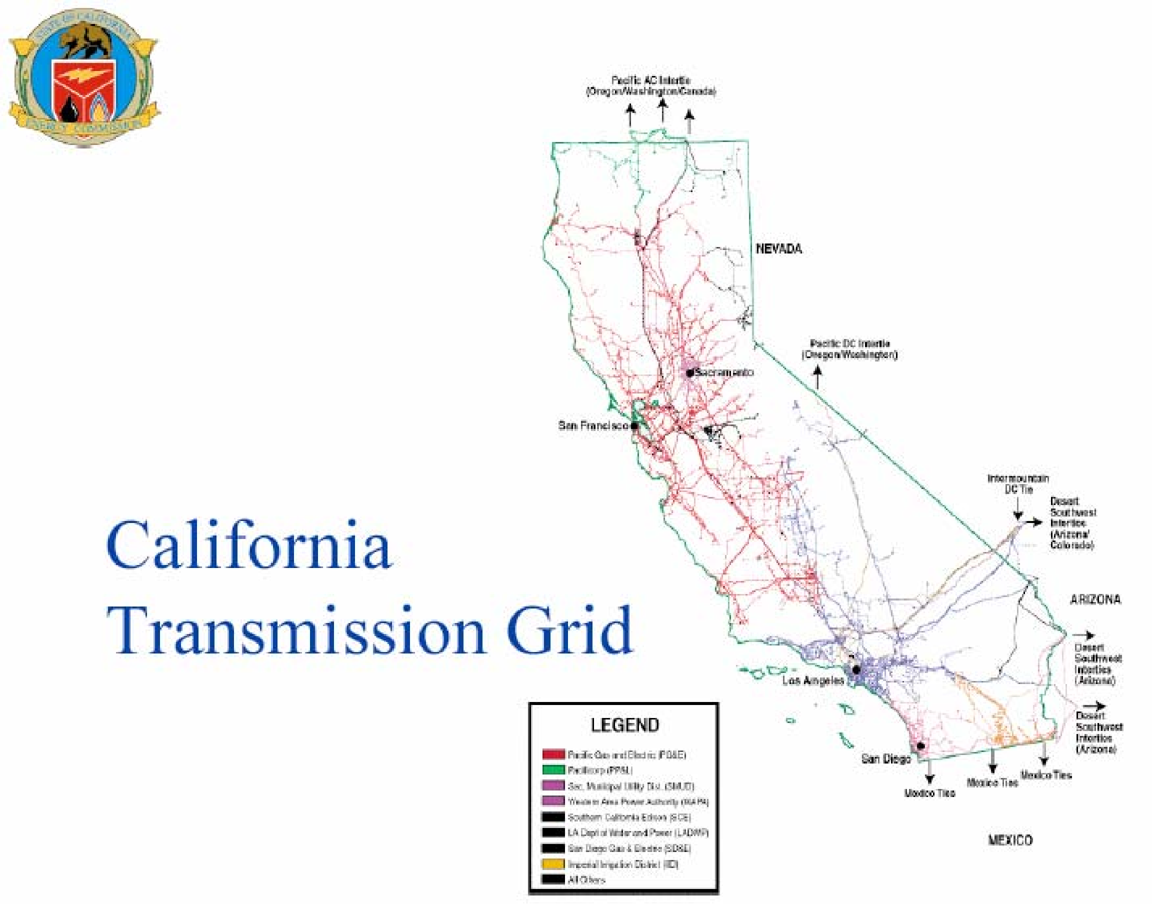
California Transmission Grid
Pacific AC Intertie (Oregon/Washington/Canada)
NEVADA
Pacific DC Intertie (Oregon/Washington)
Sacramento
San Francisco
Intermountain DC Tie
Desert Southwest Intertice (Arizona/ Colorado)
ARIZONA
Desert Southwest Interties (Arizona)
Desert Southwest Intertice (Arizona)
Los Angeles
San Diego
Mexico Ties
Mexico Ties
Mexico Ties
MEXICO
LEGEND
Pacific Gas and Electric (PG&E)
Pacificorp (PP&L)
Sac. Municipal Utility Dist. (SMUD)
Western Area Power Authority (WAPA)
Southern California Edison (SCE)
LA Dept of Water and Power (LADWP)
San Diego Gas & Electric (SDG&E)
Imperial Irrigation District (IID)
All Others

Figure 4: California Electric Transmission Grid
At least nine distinct grids are shown. Line layouts and generator locations are influenced by geography, population centers, location of water, and regulations, to name a few (von Meier 2003).

The fact that architectures can arise in different ways presents some challenges to academics wishing to understand architecture. The *de novo* case is the least constrained and thus allows analysis of a sort of "pure" situation similar to removing friction in order to understand the basics of dynamics. Understanding the pure case allows us to define an ideal process and perhaps some ideal architectures, as well as to generate benchmarks that can be used for comparison to real architectures that conform to the constraints mentioned above. Yet few if any architectures are actually created in the pure environment. Thus, the next intellectual challenges are to understand the interactions of form, functions, and constraints that practicing architects must deal with, and to understand how to balance these factors to create a "good" architecture.

Summing up, we may observe that a wide variety of systems have architectures and that architecture is what gives systems their behaviors, both good and bad. The process of creating architectures is central to creating working systems that fill defined needs in defined ways within certain constraints.

There are, however, no algorithmic procedures for creating architectures, that is, for choosing elements or choosing how to link them together, so that the desired behaviors will definitely be achieved. Furthermore, there are no tools that permit us to reason out what unintended behaviors will emerge just by inspecting the arrangement or understanding the behaviors of the individual elements. Thus, architecture is a necessary but incompletely understood property of complex engineering systems, and architecting is a necessary but incompletely understood step in creating them. This does not mean that more rigorous ways of analyzing and synthesizing architectures are beyond our collective reach. One of the goals of research in system architecture is precisely this—to uncover a set of principles, methods, and tools that will help system architects in the future.

D. WHY IS SYSTEM ARCHITECTURE IMPORTANT?

Architecture is important practically and intellectually. It is necessary in order to design systems well and to understand their behavior. Some architectures are easier to manage during design, others easier to manage during operation. Some are more robust to deliberate attack while others are more robust to random failures. Some of these aspects of architecture are discussed next.

1. ARCHITECTURE IS A WAY TO UNDERSTAND COMPLEX SYSTEMS

Architecture as “arrangement of entities and relationships between them” conveys many of the ways the system will behave. In a highway system, one can count the number of alternate routes and the capacity of each, and thus can predict the system’s overall capacity. The structure of an airline’s service network can be set up as hubs and spokes, permitting the airline to fly passengers between far-separated cities via short routes and few plane changes.

Architecture as “rules to follow when creating a system” conveys coordination rules, so that different people at different times and places can create systems that are compatible in various ways. This is efficient not only because of the advantages of coordination but also because elements and interconnection patterns with known behavior can be reused, increasing the speed with which such systems can be designed and put into operation.

Some architectural designs can follow canonical patterns whose behavior is fairly well understood. Hub and spokes mentioned above is one example and is one case of a network. Token rings and buses are other forms of architectures, particularly popular in computer networks. A tree structure is another standard architectural form. Here each element is linked to each other element by one route only. Formal organizations are usually trees, but informally they behave like more general networks in which people find alternate routes by which to communicate with each other.

Not only are a system’s desired operating modes influenced by its architecture, but so are some of its failure modes. Thus an architecture that permits only one path between elements may fail if a leg of any path breaks. All of a tree below a broken node is isolated from the rest of the tree.

Some architectures can be represented fairly completely as networks. In such cases, a lot can be determined about their behavior from graph theory. Current literature usually assumes identical nodes and identical links. Various authors (Doyle and Carlson 2000; Barabasi and Albert 1999; Watts and Strogatz 1998; Strogatz 2001) have studied the properties of such graphs in order to determine their behavior if certain nodes are removed, or if control of the network’s routes is decentralized. A general network is vulnerable to random failures while a hub and spokes network is not, since hubs can easily carry traffic routed to them around broken nodes. But a deliberate attack on one or more hubs will disable this kind of network fairly quickly whereas a general network does not present opportunities for efficient deliberate attack. The time taken by a message to go between random nodes and the vulnerability of a forest to lightning strikes are other examples of analyses that can be performed on such general models.

2. ARCHITECTURE IS A WAY TO DESIGN COMPLEX SYSTEMS

Designers can use architecture to help create systems with desired behaviors and can structure those architectures to make the task of design and manufacture easier, although these goals sometimes can be in conflict. Architecture as “rules to follow when creating a system” has already been discussed. In addition, architects speak of integral and modular architectures as differing styles with their own advantages and disadvantages.

Roughly speaking, modular architectures consist of modules, each with one or a few distinct functions, connected to each other with a few simple, well-defined interfaces. In the ideal limiting case, all interactions between modules occur over these predefined interfaces, and all system behavior is encompassed by module behavior and interactions across the defined interfaces. On the other hand, integral architectures contain modules that perform multiple functions and interact over many interfaces, some defined by the architect and some emergent as a result of module behavior or opportunistic interactions. In some limiting cases, there are no discernable modules. Most real systems are in between the limiting cases.

Modular architectures are the easiest to decompose, sometimes according to the functions performed, sometimes according to how they are designed and built, sometimes according to how they are used or perceived by users (Ulrich and Eppinger 2000; Baldwin and Clark 2000). In the ideal limiting case, the system can be built simply by plugging the modules together. The literature on drivers of modularity and decomposition is large. The notion of “elegant” architecture applies when the system architecture is very similar across multiple decomposition criteria. This is the case, for example, when the system modules are identical for design, manufacturing, procurement, and operational considerations. An example of an elegant architecture is the Apollo Program. Here, the command module, service module, lunar excursion module, as well as the Saturn rocket represented modules of a larger architecture, with many interconnections within the modules and few between them.

Conventional aircraft comprising separate wings and fuselages accomplish the functions of providing lift, carrying fuel, and housing passengers using separate portions of the aircraft. Typically wings and fuselages are designed by different engineers and made in different factories. The Airbus Consortium was structured to take advantage of this architecture. Wings are made in the UK, fuselage barrel sections in Germany, tail sections in Spain, and final assembly and integration take place in France. But there are some disadvantages in terms of coordination as well as transportation of large subassemblies. For example, the International Space Station may have suffered from certain mismatches between physical and organizational architectures.

On the other hand, blended-wing and delta-wing aircraft combine many functions in the same structural regions. Such systems are integral. Their design and construction can be more difficult, but their operational efficiency can be superior to that of more modular systems. They may contain fewer interfaces or may require only one element to carry out several functions, whereas modular architectures tend to have distinct elements assigned to each function. The extra interfaces between modules add weight and occupy space that can be used to accomplish other goals in an integral system. Extra manufacturing and assembly time may also be required of modular systems.

The option to modularize is often limited by the absolute level of power used or conveyed by system elements. Portions of jet engines may have to be made integral simply in order to manage the power; however, elements of microelectronic products can be modularized almost at will because their very low power transmission is incidental to the main function of processing information. As modularity is enforced, some of this power can be wasted—something that cannot be done in jet engines (Whitney 1996).

Also, for many systems, such as vehicles (cars, airplanes, spacecraft, submarines), there exist important packaging constraints that force all subsystems and modules to fit within a constrained volume in physical space. Vehicles have to carry much of the energy for their own travel within that constrained volume. One may argue that the existence of such constraints pushes architects and designers to make choices that promote volume and mass efficiency at the expense of purely decoupled, uncoupled, or elegant architectures. Fixed infrastructures and various kinds of non-mobile industrial machines might not have to satisfy such constraints, and, as a result, their architectures might be different. The independence axiom of axiomatic design promotes decoupling functional and physical elements of the architecture (Suh 2001). The virtues of such practices stem mainly from reduced complexity of design, manufacture, and system integration. When performance, efficiency, and packaging constraints dominate, such clean, decoupled architectures might not always be feasible.

Modular architectures can also be designed and built as distinct subassemblies, and some of this work can be distributed among many companies in a supply chain. The coordination of such work is aided by careful definition and oversight of the interfaces and relationships between these subassemblies, along with maintenance of standards for measurement. In many cases it is desirable to make the functional boundaries the same as the physical boundaries, an additional constraint on the architecture that permits subassemblies to be functionally tested prior to final assembly.

Commercial products are often given modular architectures that permit them to be customized on the spot in response to customer orders. Assembly sequences and supply chain dynamics are arranged, so that most of the product can be built in common and customization can be delayed to an economically or logistically advantageous point (called the decoupling point or the push-pull boundary) (Simchi-Levi et al. 2002).

Baldwin and Clark (2000) have studied modularity and conclude that there are three kinds.

- Modularity in design (each function is designed separately and placed in one physical object, or several functions and their objects are combined and designed together)
- Modularity in production (a group of functions or physical objects is built or bought as a package)
- Modularity in use (the customer can combine several functions or physical objects and use them, choose them when buying, or upgrade them together)

The important thing to understand is that the three different kinds of modules may be different. As illustrated in Figure 5, a module that exists in production may include parts of many modules that exist in design, This can cause confusion, cut systems in two, and create disconnected pieces of requirements that must be satisfied by different suppliers who may not have the skills. The example in the figure was provided by Francois Fourcade, who was product line manager for a front-end module for a French car supplier. Its customer cancelled the program, and the supplier, seeing no way to make a profit in view of the technical and business complexities, has not bid on this kind of item since.

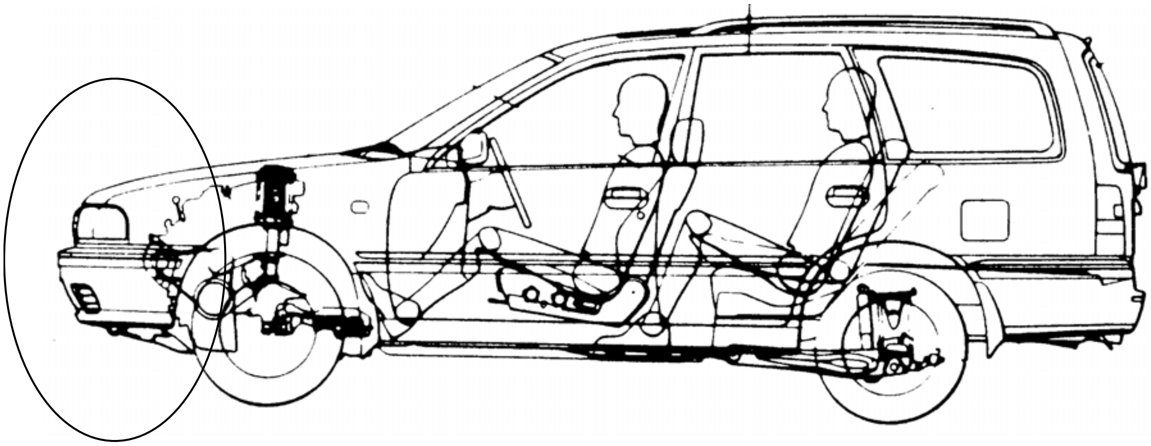

Figure 5: A “Module in Production” Consisting of Front Bumper, Bolster, Grill, and Lights
Contains portions of several systems but not all of any one. It is easy to install on the car but hard to design and test.

As shown in Figure 6, some kinds of products are easier to modularize than others. More integral products need to be designed at the highest level of the hierarchy, or their design requires a lot of coordination of the “modules” at lower levels.

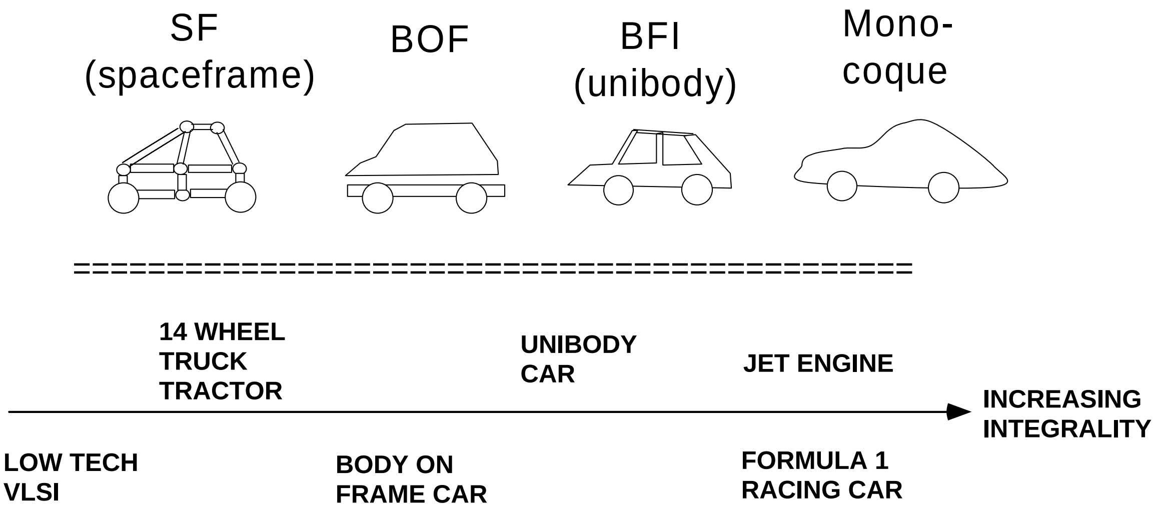

Figure 6: Examples of Products with Different Degrees of Integrality
Top: Dominant architectures of automobiles. (Illustration provided by Prof. Olivier de Weck, MIT ESD) Bottom: As integrality increases, it is harder to divide the product into independent modules. (Illustration provided by Prof Jasper Steyn, University of Pretoria.)

According to the theory of axiomatic design, the best design is one where each function is implemented in a way that is independent of implementation of any other function (Suh 2000). This permits the maximum in independence and simplicity. It is impossible to define all of these relationships at once, however. Instead, one has to start with the top-level functions and define some top-level technological choices or implementations. These give rise to a second layer of functional requirements, which in turn are linked to more detailed or subordinate technical implementations. This decomposition process continues in a zig-zag fashion to the lowest level in the hierarchy. While there is general agreement that simpler designs with independent functions are desirable, there is less agreement about whether this is possible in complex systems.

The processes for designing complex systems can also be said to have architectures, as can the organizations that carry out the design. Processes can have modules and interconnections between them, and these can all evolve over time. Steward (1995); Allen, Nightingale, and Murman (2004); and Browning et al. (2002) discuss alternate organizational patterns and show the influence of conforming or not conforming organizational structures to product and system architectures.

There is some evidence that complex phenomena become understandable only after their essential modules have been identified and characterized. A corollary requirement for understanding is that these modules act the same way in all combinations and that the types of interactions and interfaces can be enumerated and characterized.

3. ARCHITECTURE IS A WAY TO DESIGN STANDARDS AND PROTOCOLS TO GUIDE THE EVOLUTION OF LONG-LIVED SYSTEMS

Standards provide a way for systems to be designed by different and distant sets of designers using the patterns set by the standards. An early example is standard weights and measures, which permit decomposing and outsourcing system design to a supply chain. Later examples include pipe threads, electrical quantities, communication codes and protocols, and software file interchange standards. Standards often apply to interfaces, a prime focus of system architecture. Once in place, these standards provide the ability to evolve systems over time, design upgrades to existing systems, and to innovate a class of systems under the umbrella of the standard. The standard allows the system designers to skip an often difficult step in design and focus their efforts on the immediate challenges. Standards also permit systems to be made of a judicious combination of standard components and bespoke ones.

An example of a long-lived standard that gave rise to a whole set of avionics architectures in aircraft, spacecraft, and military communications systems is MIL-STD-1553.

MIL-STDs comprise, among other things, standard procedures for doing repetitive things. Even the design of products and systems can be subjected to standards, such as the German DIN system. The DIN for product development was based on Pahl and Beitz (1991).

In some domains, such as VLSI, standard components have been adopted as one way to reduce uncertainty in complex semiconductor product development. These components, either stand-alone chips or nano-scale circuit devices within a chip, act as the main function carriers of such systems. In typical high-power mechanical systems, the standard components are more often support items like fasteners. The most important main function carriers are usually designed to suit. Whitney (1996) discusses the reasons for this.

4. ARCHITECTURE IS A WAY TO MANAGE COMPLEX SYSTEMS

System architects often seek to design the systems so that they will be easily managed after they are built. System modules or segments may be defined by where they are or how easily they can be observed or controlled rather than by what they do. What they do may be perceived quite differently by operators than by designers. Thus, elements that might be designed or manufactured more efficiently as a single integrated unit might be nevertheless divided into two or more distinct units in order to make operation or repair easier.

Complex products and systems are hard to design and operate because a) they contain complex components, and b) those components interact in complex and sometimes unpredictable ways. It takes a long time for someone to learn all the interactions that are known, much less learn how to find the hidden ones. Data from several lines of research indicate that individual attributes are supported by six to twelve underlying and interacting items, and sometimes more (Whitney et al. 1999). The interactions comprise long chains that can snake through assemblies, tooling, organizations, and supply chains. Only the most senior employees, given the chance, are likely to understand the whole chain in any given situation. Management incentives and career-path planning are needed to ensure that a critical mass, perhaps 10 percent of all technical employees, have a chance to develop this kind of knowledge and put it into use.

A user or operator of a system may see it in a way that is totally different from the way its designers or builders see it. Its “modules” may be operating or failure modes, rather than physical elements. Such modes may combine elements that were not necessarily designed or built together.

The impact of complexity in an architecture may be felt most strongly by users or operators. If the interconnections or interactions are many, hidden, or changing, the operators may be unable to understand the system and operate it properly. Users may drive the system into unstable operating regimes, as happened at Chernobyl, or they may do the wrong thing because they do not know the state of the system, as happened at Three Mile Island. Thus, one must distinguish between interface complexity and behavioral or structural complexity. Interface complexity is a subjective measure of complexity as experienced by users, operators, or assemblers of the system. This subjective perception of complexity is often expressed by the adjective “complicated” rather than the adjective “complex.” Structural and behavioral system complexity are theoretically measurable, independently of the observer, provided one can agree on the metrics. It is good systems architecting practice to think about the relationship between interface and structural/behavioral complexity in the context of the needs of downstream stakeholders.

It is important to note that the designers of such systems may be unaware that they are creating complexity. The system may grow by individually understandable increments that lead to an incomprehensible system in due time. Careful planning can prevent this, but such planning must be centralized and managed consistently over the entire life of the system. This can happen under monopoly conditions as existed in the US telephone system for decades. The US electric power grid was not built by a central planning agency or a monopoly.

E. TYPES OF ARCHITECTURES

This section expands on some of the issues raised in earlier sections. Here we deal with structures or architectures for architectures and try to make some general observations.

The following figures propose a structure for architectures. Figure 7 displays the first-level decomposition of things with architectures. Each subsequent figure expands a portion of Figure 7 to lower levels of hierarchy. In many engineering systems, several of these types of entities will be involved. Thus, not only are their individual architectures important, but so is matching, or the consequences of mismatching, the architectures of the entities that interact.

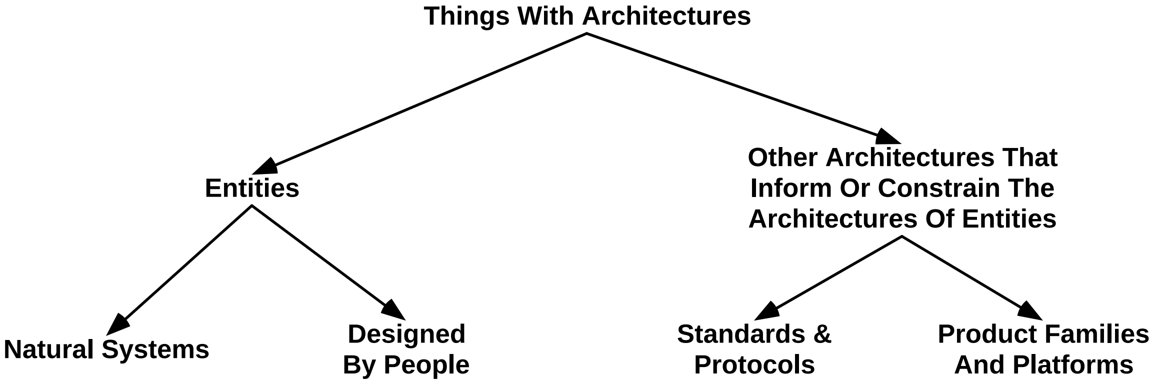
Things With Architectures
Entities
Other Architectures That Inform Or Constrain The Architectures Of Entities
Natural Systems
Designed By People
Standards & Protocols
Product Families And Platforms

Figure 7: First-Level Decomposition of Architectures

In Figure 7 we make a few main distinctions. First, entities are distinguished from architectures themselves, since both can have architectures. Not every possible architecture on the right is listed, but a couple of importance (standards and protocols, and product families and platforms) are noted because they provide constraints on the architecture of things on the left designed by people that are intellectually or commercially important. Natural systems are included because their architectures are the subject of high interest (carbon cycle, for example) or because they may offer exemplars for the design of things that might as a consequence have certain desirable properties.

Figure 8 pursues the branch in Figure 7 that deals with entities designed by people. On the left are distinguished two kinds of entities whose internal constraints make their possible architectures quite different. These differences extend to the ways they are designed and how they behave. On the right (with abstractness increasing to the right) are things with architectures that have no necessary physical existence. The lack of physical existence frees them of many constraints that inhibit architectural choice or implementation in physical things. For example, thinking in network terms, nothing limits the number of arcs that can emanate from a node in a non-physical system, but heat, weight, or mechanical constraint are examples of influences that limit this number in physical things. (Bounded human rationality, however, may provide a limit in non-physical things.)

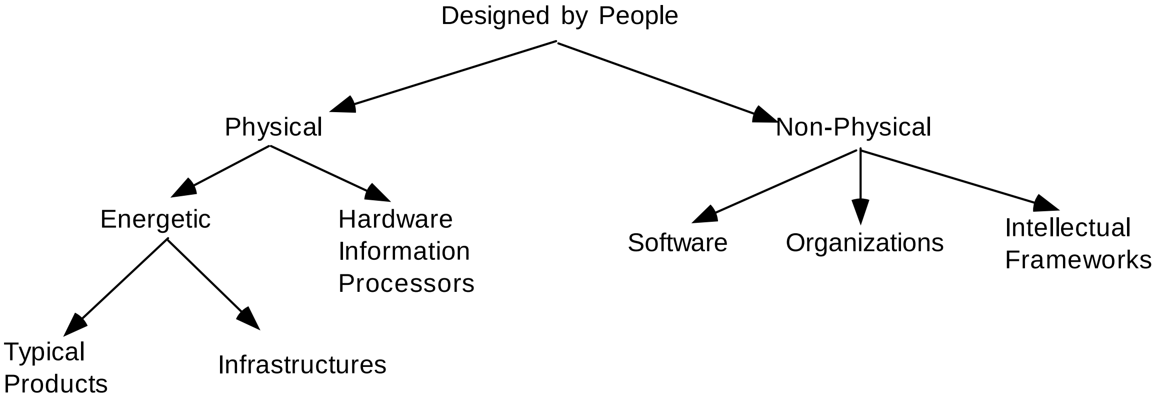

Figure 8: Several Important Things Designed by People

Figure 9 displays an expansion of one important node from Figure 8, namely that of organizations. A few important kinds are shown. Government and non-governmental organizations are shown mainly because they provide contexts such as regulations that constrain architectures of other designed things, both physical and non-physical. Companies and enterprises are important in themselves because they can have many possible architectural forms and because those forms interact with and encourage or inhibit the design of other entities, such as products. At the lowest level in this figure there is increasing abstractness, diversity, and lack of structure proceeding from left to right.

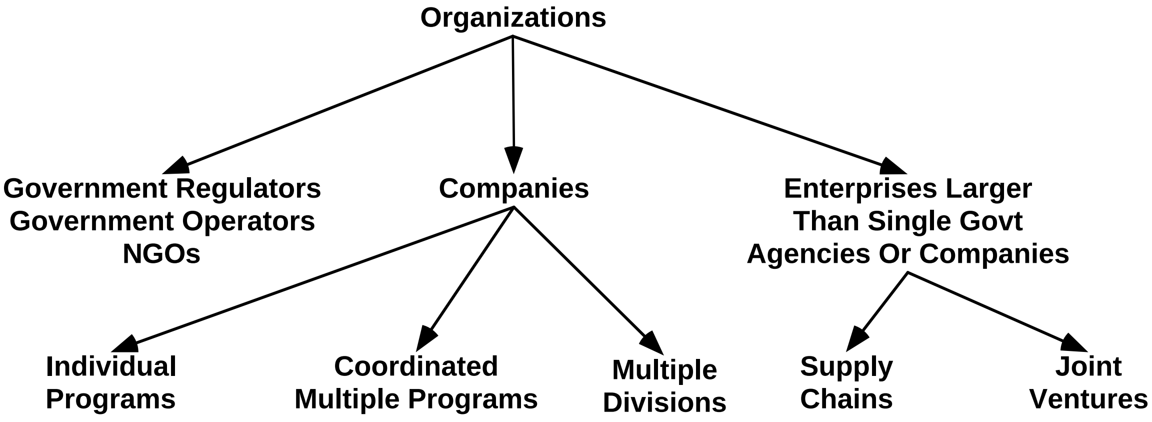

Figure 9: Two Levels of Decomposition of Organizations as Examples of Non-physical Things with Architectures That Are Designed by People

The last section of Figure 7, intellectual frameworks, appears in Figure 10. Pursuit of this domain is as old as philosophy, but it is included here because our committee has engaged in it, and those who design organizations and their knowledge have done so as well. Again, these architectures can encourage or inhibit the creation of important entities listed elsewhere in the figures. Also, there is again a somewhat increasing level of abstractness proceeding from left to right.

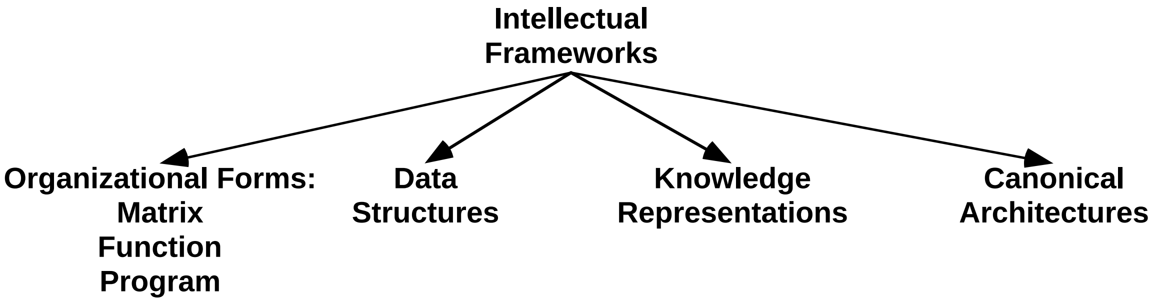

Figure 10: A Decomposition of Intellectual Frameworks

For example, it is known that organizational forms differ widely and are variously suited or unsuited for managing situations in Figure 9, such as individual programs or coordinated multiple ongoing programs (overlapped car development activities, for example).

F. DOMAINS WHERE ARCHITECTURE IS IMPORTANT

Architecture is important to both naturally occurring and overtly designed systems since architecture conveys behavior.

Examples of naturally occurring systems whose architectures have been studied are biological systems (ecosystems, cells and small organisms—see Figure 11, the carbon cycle, and the etiology of diseases) and social systems (villages, military units, workplaces). Among the models built are networks, dynamic simulations, and control systems. Network theory and graph theory have been used to measure the networks, discover connections and cliques, and determine how information, energy, and material flow between elements (Kitano 2002; Sterman 2000). Naturally occurring systems are of interest in their own right as well as because they may tell us something about how to provide overtly designed systems with desirable properties such as robustness to element failure.

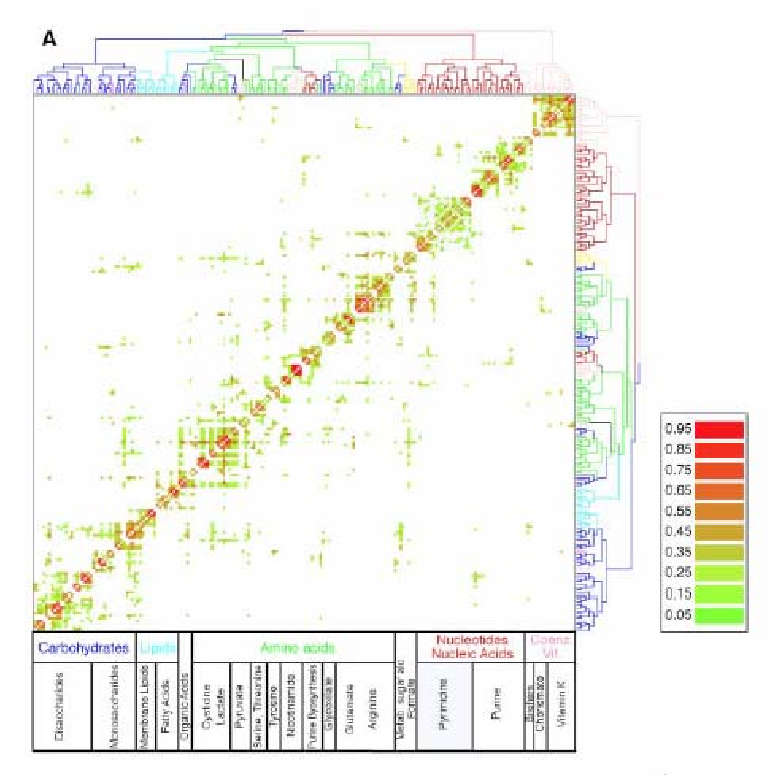

Figure 11: The Topologic Overlap Matrix Corresponding to *E. coli* Metabolism, with the Corresponding Hierarchical Tree That Quantifies the Relation between the Different Modules (Ravasz et al. 2002)

Some physical systems have emerged without design but perform complex functions and have distinct architectures. These include informal social networks that spring up within defined organizational structures. Another well-known example is the Internet, which has no central control or growth rules. It is relatively efficient and effective but is hard to manage when uniformity is needed, such as for diffusing security precautions.

Some infrastructure systems conform to natural constraints, such as railroads and highways that follow terrain, river valleys, or mountain passes. In some cases the routes are thousands of years old and have been repeatedly reconstructed using new technology. In some cases, new technologies allow designers to push back on some constraints (tunnels through the Alps, Chunnel, bridge connecting Denmark and Sweden), albeit at significant cost.

Finally, there are overtly designed physical systems, such as cars, planes, airports, factories, shopping malls, and supply chains. Each of these seeks to perform certain functions and must conform to a variety of physical and administrative constraints during construction and use, including the laws of nature, building codes, industrial standards, safety regulations, and work rules, to name few. Some such systems, like the DC-3 aircraft, fulfill their intended role and do so unchallenged throughout a long life cycle. Others, like the B-52, outlive their original function and repeatedly take on new ones not conceived of when they were designed. Yet others, like the B-58, barely serve their original function and go out of service quickly, unable to be used for anything else. The deeper architectural reasons for these different outcomes are not well understood. The word “flexible” is used to describe such systems, but a universal formula for creating flexible systems does not exist.

In some cases, flexibility comes at a price—namely, efficiency in some form. Flexibility may require over-design, generic components, extra interfaces, or changeover time. A less flexible system might have more focused components, fewer interfaces, and no loss due to changeover. The need for flexibility shows that the process of architecting requires a model of future usage of the system, including an understanding of uncertainties in the environment, competition, regulations, and future user needs. This is a central issue addressed by a parallel ESD committee on uncertainty management.

The architectures of most overtly designed systems usually emerge from long iterations between potential users and the system architects and engineers. The DC-3 followed the rejected DC-1 and DC-2 as the engineers sought to meet the specifications and the users saw what was possible economically and physically. The B-52 might be seen as a successful iteration following the less successful B-48 and B-50. The structure of General Motors’ products in the 1920s emerged as the company discovered how to implement the strategy of “a car for every purse and purpose” while at the same time maintaining some rationality of families and reuse of expensive components like engines (Sloan 1996).

Similarly, different architectures of production systems have emerged successively over time in response to different needs. The principle of division of labor permitted specialization and economy of scale. The assembly line brought the work to the worker to cut wasted motion and worked best when identical products were made over and over using the same route and sequence of work. Flexible manufacturing systems sought to combine efficient use of individual workstations with flexible transport systems, permitting different routes for different products in the same factory. Common car body architectures are intended to permit different kinds of cars to be made on the same single route-single sequence assembly line. Each of these architectures meets, in different ways, different needs for efficiency and flexibility, two of many ilities.

As with the aircraft cited above, each of these architectures emerged and was not successfully designed on the first try. Each has its own strengths and vulnerabilities as well as emergent behaviors. Assembly lines led to boredom and, combined with hierarchical management, to industrial strife, while flexible manufacturing systems presented difficult scheduling problems. Both were unanticipated, and both have been mitigated by additional architectural and managerial initiatives.

Another reason why architectures of designed systems emerge is immaturity of technology and product alike. Utterback (1994) and others (for example, see Christensen 1997) have noted the cycles of industrial evolution in domains like typewriters and disk drives. When a technology is new, no one knows what it is or what it can do. There is an exploratory phase when many innovators and users try different things. At some point, for various reasons, a so-called dominant design emerges. This design is usually an architecture. An example is the 1920s single metal wing aircraft with stressed metal skin, which was adopted after twenty-plus years of exploration with multiple wings, cloth wings, wings with struts and guy wires, and so on. Only now is this architecture being questioned by both Airbus and Boeing. Once the dominant design emerges, innovation switches from product architecture to process architecture, including production, supply, and distribution systems.

Thus requirements themselves are apparently emergent properties of systems, making their design necessarily an exploratory process, at least when the system's technology or the desired functions have not been implemented before.

G. EMERGING PRINCIPLES OF ARCHITECTING

The previous discussion highlighted the importance of architectures in many different domains from product development to large infrastructures. One may ask whether there are generalizable insights to be gained across these domains. One may even go further and attempt to uncover or formulate "principles of architecting." A principle is a statement that is almost universally true and is long enduring. We contrast principles of architecting with methods and tools, which might become obsolete on shorter time scales.

Prof. Ed Crawley has been collecting principles of architecting over the last five years in his course on systems architecture at MIT, and there is empirical evidence that such principles might exist, regardless of whether one considers, civil, computer, product or other architectures. Principles can be either descriptive (how things are) or prescriptive (how things ought to be).

An example of a principle of systems architecting is: Robust functionality drives essential complexity. This is a descriptive statement that introduces a third notion of complexity, aside from structural/behavioral and interface complexity as described above. Essential complexity is a theoretical lower bound to the complexity of a system, required to achieve a specified function or set of functions with desired precision, efficiency and repeatability. The existence of such a lower bound has not been proven. This is in sharp contrast to lower bounds in other disciplines, e.g., Shannon's bound in information theory.

Let us consider a very simple example to explore the proposed essential complexity principle of systems architecting. An example of simple systems of varying complexity are bottle cork removers. They come in various shapes and sizes, and use a variety of working principles (spiral screws, vacuum pumps, gas injectors…). They are all designed to fulfill essentially the same function. Expressed in its essential form, this function is to get the cork out of the neck of the bottle. Most commercial realizations operate by exerting an axial force on the cork, reacting this force axially on the bottle. Those that tend to fulfill the function more robustly, e.g., requiring less force, also tend to be more complex. That is, they have more system elements and interactions, and may require more steps to operate. Good system designers attempt to design for simplicity, but usually some amount of gratuitous complexity is introduced. This unnecessary amount of complexity is the difference between the (unknown) essential complexity and the actual complexity of the final product or system. Good architectures, according to this principle, have minimal gratuitous complexity.

Further principles of architecture might emerge over time, but they can only be viewed as heuristic guidance until they are proven by rigorous mathematical proofs and physical evidence in actual systems. There is debate whether such rigor is achievable or necessary in practice.

Table 2 categorizes some of the activities involved in architecting a complex engineering system. These are partitioned into two phases and two types of considerations. While these activities do not rise to the level of principles, it is felt that these steps are involved in the creation of many—if not all—architected systems.

	Technical Matters	Non-Technical Matters
“Direct” design aimed at achieving the main functions—these are the functions required immediately upon initial fielding of the system	Determine structure that achieves properties and behavior Manage complexity “in the box” using clustering, interface clarity, and other principles of architecting Be mindful of uncertainty Prepare for verification at multiple levels	Consider multiple stakeholders Manage complexity at the human interfaces Match system architecture to the architecture of the context and stakeholder organizations Prepare for validation at multiple levels
Planning for the life cycle (manufacturing, operations, upgrade, retirement) plus achieving the ilities (reliability, flexibility, operability, etc.)	Be mindful of emergence, including technological evolution and disruptions Prepare for upgrades, customization, repair See to reliability, durability, scalability	Consider multiple stakeholders Be mindful of emergence, including human agency Be mindful of changing missions and circumstances

Table 2: Considerations Applicable to Architecting Complex Engineering Systems
The term “be mindful” is an acknowledgement that systematic procedures do not exist for obtaining many desirable features of a complex system.

Direct design encompasses traditional engineering that is aimed at achieving the system’s main recognized functions. Life-cycle planning addresses the ilities as well as activities normally associated with the system’s life. Technical matters begin with the act of architecting itself (structure > properties > behavior) but extend to other requirements. Non-technical matters include non-physical architectures and the activities of people as users, passive participants, or observers of the system.

H. CRITICAL PROPERTIES OF SYSTEMS AFFECTED BY ARCHITECTURE

1. DELIVERY OF BASIC FUNCTION

Architecture as “arrangement of entities and relationships between them” or as “relationship between form and function” is the designer’s solution to the problem of finding a physical embodiment that will deliver the required functions. Example decisions include that cited above between the basically radial style of piston and high-bypass jet engines on the one hand and the basically axial style of low-bypass jet engines. Each of these delivers thrust but in very different functional ways.

Traditionally, design to achieve basic function focuses on performance, time to market, cost, and risk. Maier and Rechtin (2000, figure 5.2) show these four quantities as being in tension with each other. One of the main jobs of architects and product designers is to understand and resolve this tension. Traditional engineering education tends to focus on performance. Adding time and cost creates a complex interaction in which elements of each must be traded off in order to arrive at a design. Risk is the most difficult dimension to understand and address. A wide array of methods, intellectual frameworks, and computerized tools has emerged to support this process. These include traditional engineering design (Pahl and Beitz 1991), axiomatic design (Suh 2001), product design and development (Ulrich and Eppinger 2000), constraint management (Goldratt 1991), computer-aided design and manufacturing software, the design/dependency structure matrix (Steward 1995), time to market and time to profit metrics, among others. It can be debated whether risk should be treated as a separate entity, or whether it acts as a qualifier on performance, schedule and cost (Browning et al. 2002).

2. OTHER PROPERTIES

From a traditional point of view, it is comparatively easy to say what the functions of a product, system, or enterprise should be. An airplane must fly, a manufacturing company must design and deliver products, and so on. But it has to do these things well, and here is where properties called ilities come into play. Examples include robustness, adaptability, flexibility, safety, and scalability, to name a few. The architecture has a strong influence on how the ilities are achieved and how their achievement interacts among themselves and with the basic function.

Doyle says that comparatively little effort is devoted to assuring that an entity like an airplane will provide the required functions of lift and forward speed. The rest is overwhelmingly large and comprises providing safety, redundancy, amenities, automated functions, and so on. But this statement must be tempered by the thought that lift and forward speed may not be a complete statement of the required function. Only by considering the aircraft in the narrowest sense can this list be considered complete. At a minimum it must deliver lift and speed with sufficient safety and fuel efficiency. It also must be able to take off and land on required or available runways and must be able to make an emergency landing right after takeoff. These additional basic functions add severe constraints to speed and lift specifications. Things are further complicated when various types of internal and external payloads and delivery processes are considered.

Boeing manufacturing employees acknowledge that the design engineers mainly focus on assuring that the aircraft will not physically break during takeoff, flight, and landing. The rest, including manufacturability, is addressed by others who petition the design engineers to pay attention to these other needs. Similarly at Ford, the body engineers focus mainly on structural integrity, stiffness, crash-resistance, and squeaks and rattles. The manufacturing engineers perform a function similar to those of their Boeing counterparts.

More generally, the requirements must be stated in the context of the enterprise. This means that the product has to satisfy a combination of interlocking technical and business requirements. The aircraft must be able to take off fully loaded with revenue cargo and over its lifetime must return a profit to its owner who will pay ten times more for fuel and maintenance that its original purchase cost. Not only must the gravity-lift equation balance but the cost-revenue equation must do so as well. At Boeing this is called “making the business case close,” and failure to close the business case is often the reason why aircraft programs are cancelled or aircraft do not succeed in the market.

The architecture of the product has a large influence on how these other properties are achieved, as well as how well and how efficiently. An example was given in Figure 3.

The next few subsections discuss specific ilities.

a) Robustness

Robustness is defined as “the demonstrated or promised ability of a system to perform under a variety of circumstances, including the ability to deliver desired functions in spite of changes in the environment, uses, or internal variations that are either built-in or emergent” (ESD 2002).

Robustness may be implemented in several ways, including by paying attention to the reliability of system elements as well as reliability of interconnections. In addition, robustness may be enhanced by the structure of the system, such as by providing alternate paths or backup systems. An intellectual framework for designing robust systems revolves around the P-diagram, which is a discipline for identifying "noises" from within or outside the system (see Figure 12). Engineers are asked to create a P-diagram for their component, subsystem, or system. Identification of the noise factors is the most difficult part of this process, since this identification requires engineers to think beyond the boundaries of their part. A lot of knowledge about interactions from other systems, including customers, is needed. At Boeing, the pilot is considered a possible noise source. As more comprehensive subsystems and systems are considered, developing their P-diagrams involves more people. There is some question as to whether the noise factors for an entire system can be identified.

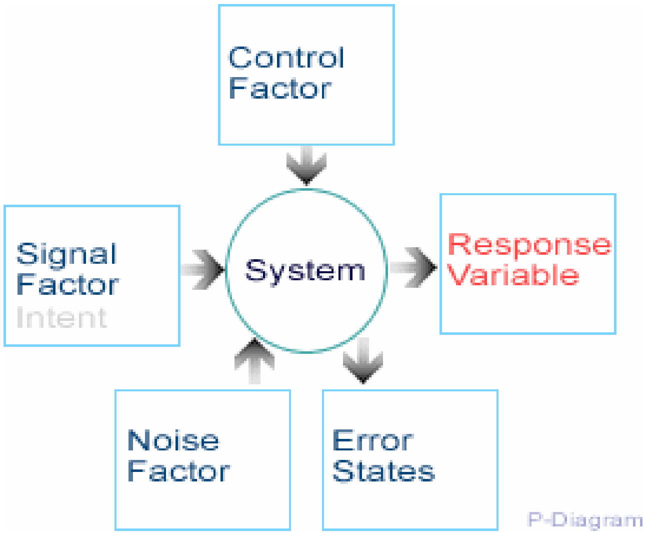

Figure 12: The P-Diagram.
(Source: http://thequalityportal.com)

The P-diagram approach, however, does not address the role of architectural form in providing robustness. The network theorists (Barabasi, Doyle, and others) note that different network architectures are differently able to resist attack. Doyle, however, says that additional effort at providing robustness only adds layers of system elements and connections, providing only additional complexity and points of potential failure (Doyle and Carlson 2000). Perrow (1999) makes the same observation. Thus there is no consensus on whether architectural form by itself can impart robustness to a system. Nevertheless, it is known that certain forms are easier to understand, partition to avoid cascading failures, diagnose, and maintain. These generally have a more modular structure and were designed with careful attention to keeping the modules separate.

b) Adaptability

Adaptability is defined as "the ability of a system to change internally to fit changes in its environment," usually by self-modification to the system itself (ESD 2002). Such systems contain sensors, control algorithms, and human operators. The technical portions must be designed to be stable while the human elements must be trained to distinguish desired signals from distracting noise. The technical architecture of such control systems is well developed but holds some risks when decisions must be made quickly and the behavior of the system is not well understood, or when the human interface is not well designed (Perrow 1999).

c) Flexibility

Flexibility is defined as “the property of a system that is capable of undergoing classes of changes with relative ease. Such changes can occur in several ways: a system of roads is flexible if it permits a driver to go from one point to another using several paths. Flexibility may indicate the ease of ‘programming’ the system to achieve a variety of functions. Flexibility may indicate the ease of changing the system’s requirements with a relatively small increase in complexity and rework” (ESD 2002). Flexible systems include product families that can be configured to meet changing markets. An example is a family of aircraft where the fuselage can be lengthened merely by adding short sections fore and aft of the center section. At an enterprise level, a flexible system can deliver products or services quickly and accurately in response to real-time demands. An example is Dell’s manufacturing, supply chain, and fulfillment system. In both example cases, the flexibility arises from the architecture of the system itself, which exploits defined interfaces to permit substitutions at predefined points in the product or process.

Flexibility can be thought of as a means of managing risk, specifically risk that the system as designed will not meet its requirements some time in the future. The flexibility committee’s paper (de Neufville et al. 2004) notes that financial options theory can be applied to determine the value of providing (at some cost) the flexibility to respond to future contingencies. These options permit some decision about the system to be delayed until more information is available. Financial options have been used this way for decades. They have the advantage that future value can be calculated. In addition, there is some equivalence of value between different options since they all can be reduced to money and can be readily substituted for one another. Whether this can be accomplished in physical systems is the subject of future research.

d) Safety

Safety is defined as “the property of being free from accidents or unacceptable losses.” Associated with this definition are several others: An accident is “an undesired and unplanned (but not necessarily unanticipated) event that results in a specified level of loss” (human, economic, etc). A hazard is “a state or sets of conditions that, together with worst-case external conditions, can lead to an accident.” Risk is “the level of hazard combined with the likelihood of the hazard leading to an accident, and the duration of exposure to the hazard” (Leveson 1995).

Endowing a system with safety involves determining, among other things, how safe is safe enough. This question is beyond the scope of architecture alone; however, architecture can improve or reduce safety in various ways. Systems whose architecture is transparent are easier to diagnose, and hazards may be easier to detect and prevent. No method exists, however, to compare the safety of two architectures.

e) Scalability

Scalability is defined as “the ability of a system to maintain its performance and function, and retain all its desired properties when its scale is increased greatly, without causing a corresponding increase in the system’s complexity” (ESD 2002). Ways of increasing the scale of a system include increasing the number of elements, increasing the number of possible paths or connections between elements, increasing the “traffic” capacity of links or connections, or all of the above. The degree to which the complexity of the system increases depends greatly on how the structure is designed and how or if it changes as the scale is increased. Some architectures permit additional scale by adding identical subsystems. Short and long truss-style bridges and short and tall buildings can be made this way. Others permit additional scale by adding differently structured layers. An example is the US national wired telephone system.

Most systems incur increased complexity as their scale increases. Taller buildings require segmented elevator systems since a single elevator cannot be built taller than a limit that is now less than building height. Coordination of these elevators to permit people to travel the full height of the building can be complex. Some plans for mega-buildings envision creating separate stacked modules for full-time living that do not require residents to move beyond their module. This reduces the complexity problem to that of the identical layers method.

Really large systems pose particular challenges. Each has its own architecture and scaling properties. Some are built in an evolutionary way rather than being manufactured repetitively according to a standard pattern that was established before construction begins. Some, like the International Space Station, cannot be assembled and tested before being put into use. They are so complex, and their operating environment is so hard or costly to duplicate, that they cannot be tested end to end prior to deployment. In some cases, very large infrastructures can be implemented in pieces and each piece can be tested before it is added to the existing system. This approach is sometimes referred to as “staged deployment.” Staged deployment for large constellations of communications satellites has been proposed as a way to reduce economic risks for systems such as Iridium and Globalstar, which were designed “all at once”, rather than in a staged manner (de Weck et al. 2003). But success in these tests does not guarantee that unexpected behaviors will not emerge after the larger system begins to operate. This and other issues of scale remain unresolved.

I. OPEN RESEARCH QUESTIONS

In this section, we list a number of research questions that may have definite answers or may become recurrent themes without final resolution. The list is not intended to be exhaustive, but rather to be illustrative and provocative. Our hope is that the field of engineering systems architecture will create an effective way of thinking about complex systems, generate a shared vocabulary, develop and validate research methodologies, and improve the practice of generating and evaluating system architectures.

The list of questions follows:

Systems that are complex enough to be of interest to the Engineering Systems Division exhibit emergent behavior. Some emergent behavior may be beneficial because it gives the system desired qualities that the parts of the system do not have by themselves. But some emergent behavior is detrimental because it represents sneak paths or other unpredicted behavior that is not wanted. How can we get the good without the bad, or at least predict and mitigate the bad? Perhaps this should be called “emergence management.”

Is emergence management possible, or do the methods we use now (testing, redundancy, sensors, etc.) only add complexity and introduce emergence elsewhere in the system?

Is complexity inevitable in systems that are commercially meaningful or that do enough good things well to make them worth building? Are there less complex alternatives that could do the job just as well, such as locally generated electricity as an alternative to central generators and complex distribution grids?

In principle, all the behaviors of a system could be predicted if we pursued reductionism to its limit, modeling and testing every element and every combination of elements, but this limit is unachievable in practical cases. Emergence is the opposite of reductionism.

Cognitive limits prevent us from thinking of everything. They also prevent us from making perfect models of real systems. Incomplete or inaccurate models are another source of emergence.

Are there architectures that can minimize the effects of cognitive limits? Hierarchies are one such architecture, but can they be used in all systems and do they apply to all types and consequences of cognitive limits?

Will human agency inevitably create emergent behavior? Agency can arise due to cognitive limits or willful intent. Perrow (1999) argues that some systems will always be too complex to design or operate properly and thus should not be built. Weick et al. (1999) argue that people's ability to improvise and use their accumulated experience is the only known barrier to emergence, but this argument applies only in emergencies in the examples they cite.

Systems where reductionism appears to work are often those that exhibit a great deal of modularity. An example is low-power micro-electronics. In these cases the individual modules can be modeled and their models validated separately. Additionally, their behavior when connected into systems is reasonably predictable. This is accomplished by several means, such as enforcing standardization and forcing interfaces to be one-way, via huge impedance mismatches. What are the limits of extending this approach to all systems?

How many kinds of things can usefully be said to have architectures? Examples include systems, products, organizations, and processes.

What are ilities, and can they be classified? A suggested classification comprises two types: those used during system design to anticipate future uncertainties, such as manufacturing, normal use, abnormal use, technological or market evolution, and so on; and qualifiers on performance or fitness, such as cost, robustness, safety, or reliability.

Can systems be classified? Classes could include primary function delivered, logical or physical structural type, operating power level, type of enterprise (public/private), type of use pattern (product, infrastructure), or degree of evolution over time. What characteristics do all classes share, and what characteristics distinguish them?

How can the relative or absolute complexity of a system be measured, or can systems even be ordered according to complexity? Can other system characteristics, such as reliability, be normalized with respect to complexity in order to create a scale of expectations against which the reliability per unit of complexity (for example) can be judged?

How do we evaluate the ilities content of a system?

More generally, how can the "goodness" of a system be evaluated?

What is the difference between designing a new system and designing one to fit legacy systems, predefined and populated product families, existing standards of product, interface, or process? How much in terms of functions or ilities is lost when these constraints are applied, what is gained, and is the gain worth it?

To what degree do these constraints inhibit the inclusion of new technologies, or do new technologies simply invalidate old architectures?

How do we represent the architecture of a system? How many things need to be in the representation? For example, a network can represent the simple binary relationships between system elements, but it may require another network to describe some of the strengths or histories of these relationships. Some relationships might be static (i.e., they are permanently present); others might only exist temporarily or during certain operating or failure modes. What aspects of an architecture cannot be represented by a network or networks, and how should they be represented? Can there be a unified representation of system architectures that captures simultaneously both structure (objects) and behavior (processes), or must we inevitably resort to multiple, related views?

If we say that systems have properties, structure, and behaviors (alternately structure, functions, and organization), what methods exist for describing, modeling, testing, and synthesizing these?

Are certain kinds of structures or architectures known to be good and would this apply across the variety of domains discussed here? It is often recommended that functional and physical structures should correspond, having similar clusters. This is obviously a cognitive aid, but is it always a good thing in other dimensions of function or ilities? For example, combat systems have distributed physical structure in order to be robust against point damage. Another example is highly clustered networks: they provide shorter paths and are much more robust to random node damage but much less robust to deliberate attack on the hubs than are random networks.

Are natural systems good exemplars for designed engineered systems? Some natural systems have ilities that we would like engineered systems to have, such as flexibility and robustness, even when individual modules are not very reliable.

Do different observers or participants in an architecture require different representations? Examples include the designer, the user, the manager of a family of items in an architecture, a safety or risk analyst.

Is there a process for generating an architecture that will have certain desired properties while avoiding certain undesired ones? Typically, architectures themselves emerge from an evolutionary process that weeds out unfit ones according to market pressures on functions or pressures on the ilities. There are heuristics but no theory or guaranteed algorithms.

How determinative is architecture (compared to detailed design, module behavior, and behavior of users or operators) in deciding how a system will behave, how it comes out on the scales of ilities, etc.? For example, Doyle and Carlson (2000), Barabasi et al. (2002), Strogatz (2001), and Watts (1999) argue in various ways that highly clustered networks are better in some respects than random ones even if the nodes are the same. But up to now these analyses have assumed identical nodes and arcs. What if these are allowed to be different? Is structure still dominant?

J. CONCLUSIONS

> The architecture is the form of the system and is the dominant factor in its behavior. In some cases the function can be deduced by inspecting the form while in others (for example, software), the form conveys nothing about the function.

> Systems have behaviors that no subsets of their elements have. These behaviors are products of the interactions between the elements. They may be anticipated and designed in, or they may be unanticipated, in which case they are called emergent. Both anticipated and emergent properties may be desirable or undesirable.

> Emergent properties, desirable and undesirable, exist because we do not understand the system or its interactions with its context completely.

> To the extent that emergent behavior is caused by unpredictable factors such as human agency, future changes to the system, or the inability to model every possible system state, it may never be possible to prove that a given architecture has or does not have a particular behavior.

> The aim of system design, and of architectural design within system design, is to obtain the desired behaviors (functions plus ilities) while suppressing undesirable behaviors. The system's architecture is chosen to enhance achievement of these goals. A key issue for future research is to learn how to create systems with the desired behaviors and to predict and suppress the undesired ones.

> Systems have multiple architectures and hierarchies of architectures. This can occur because of the choice in defining the system's boundary. It also occurs because the system has both a physical architecture and numerous virtual

architectures that seek to capture important aspects or views of its behavior. Many of these virtual architectures correspond to mental models of different behaviors.

> Multiple representations are needed to describe systems and their architectures. Many aspects of system architectures can be described by networks of one kind or another, but not all architectures can be described this way (houses, microprocessors). Network analysis is the most abstract architectural modeling method and holds promise for elucidating general properties of architectures; however, typical networks represent only connectivity. Hence, more specific models than those currently in use will likely be needed.

> There are some basic architectural elements from which more complex architectures can be built. Some kinds of desirable system behaviors can be obtained by using these primitives singly or in combination. Thus it is desirable to understand these elements more completely in order to understand how to use them in architectural design. (This is a version of the statement that structure begets properties which beget behavior.) Standards and standard components and procedures can be thought of as primitives.

K. SELECTED BIBLIOGRAPHY

Albert, Reka, and Albert-Laszlo Barabasi. 2002. Statistical Mechanics of Complex Networks. *Rev Mod Phys* 24 (January):47-97.

Allen, Thomas, Deborah Nightingale, and Earll Murman. 2004. Engineering Systems: An Enterprise Perspective. Paper presented at the MIT Engineering Systems Symposium, 29-31 March, at Cambridge, Mass. http://esd.mit.edu/symposium/monograph.

Engineering Systems Division (ESD). 2002. ESD Symposium Committee Overview. In *Proceedings of MIT ESD Internal Symposium*. Cambridge, Mass.: ESD. http://esd.mit.edu/WPS.

Baldwin, Carliss Y., and Kim B. Clark. 2000. *Design Rules: Volume 1. The Power of Modularity*. Cambridge, Mass.: MIT Press.

Barabasi, A. L., and R. Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286(5439):509-512.

Barabasi, A. L., H. Jeong, et al. 2002. Evolution of the Social Network of Scientific Collaborations. *Physica A* 311(3-4):590-614.

Browning, T. R., J. J. Deyst, S. D. Eppinger, and D. E. Whitney. 2002. Adding Value in Product Development by Creating Information and Reducing Risk. *IEEE Trans on Engineering Management* 49(4):443-458.

Carlson, J. M., and J. Doyle. 2002. Complexity and Robustness. *Proceedings of the National Academy of Sciences of the United States of America* 99:2538-2545.

Carlson, J. M., and J. Doyle. 1999. Highly Optimized Tolerance: A Mechanism for Power Laws in Designed Systems. *Phys Rev E* 60(2):1412-1427.

Christensen, Clayton M. 1997. *The Innovator's Dilemma*. Boston: Harvard Business School Press.

de Weck, O., R. de Neufville, and M. Chaize M. 2003. Enhancing the Economics of Communication Satellites via Staged Deployment and Orbital Reconfiguration. Paper AIAA-2003-

6317, presented at the AIAA Space 2003 Conference and Exhibition, 23-25 September, at Long Beach, California.

De Neufville, Richard, et al. 2004. Uncertainty Management for Engineering Systems Planning and Research. Paper presented at the MIT Engineering Systems Symposium, 29-31 March, at Cambridge, Mass. http://esd.mit.edu/symposium/monograph.

Doyle, J., and J. M. Carlson. 2000. Power Laws, Highly Optimized Tolerance, and Generalized Source Coding. *Physical Review Letters* 84(24):5656-5659.

Fine, C. H. 1998. *Clockspeed*. Reading, Mass.: Perseus Books.

Goldratt, E. M. 1992. *The Goal: A Process of Ongoing Improvement*, 2nd ed. Great Barrington, Mass.: North River Press Publishing Corporation.

Henderson, R. M., and K. B. Clark. 1990. Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly* 35(1):9-30.

Kitano, H. 2002. Systems Biology: A Brief Overview. *Science* 295(1 March):1662-1664.

Leveson, N. 1995. *Safeware*. Boston: Addison-Wesley.

Levis, A. 1999. System Architectures. Pages 427-454 in *Handbook of Systems Engineering and Management*, edited by A. P. Sage and W. B. Rouse. New York: John Wiley & Sons.

Maier, M. W., and E. Rechtin. 2000. *The Art of Systems Architecting*, 2nd ed. Boca Raton, Fla.: CRC Press.

Pahl, G., and W. Beitz. 1991. *Engineering Design*. London: Springer.

Perrow, C. 1999. *Normal Accidents*, revised ed. Princeton, N.J.: Princeton University Press.

Ravasz. E., A. L. Somera, D. Mongru, Z. N. Oltvai, and A.-L. Barabasi. 2002. Hierarchical Organization of Modularity in Metabolic Networks. *Science* 297(30 August):1551-1555.

Rechtin, E. 1990 *Systems Architecting* Prentice Hall PTR

Reynolds, D., J. M. Carlson, et al. 2002. Design Degrees of Freedom and Mechanisms for Complexity. *Physical Review E* 66(1).

Simchi-Levi, D. P. Kaminsky, and E. Simchi-Levi. 2002. *Designing and Managing the Supply Chain: Concepts, Strategies and Case Studies*, 2nd ed. New York: McGraw-Hill.

Sloan, Alfred. 1996. *My Years with General Motors*. New York: Doubleday.

Smith, G., and D. A. Mindell. 2000. The Emergence of the Turbofan Engine. Pages 107-156 in *Atmospheric Flight in the Twentieth Century,* edited by Peter Galison and Alex Roland. Vol. 3. *ARCHIMEDES New Studies in the History and Philosophy of Science and Technology*. Dordrecht: Kluwer Academic Publishers.

Sterman, J. 2000. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Boston: Irwin/McGraw-Hill.

Steward, Donald. 1995. *Systems Analysis and Management: Structure, Strategy, and Design*. Fair Oaks, Calif.: Problematics.

Strogatz, S. 2001. Exploring Complex Networks. *Nature* 410(8 March):268-276.

Suh, N. P. 2001. *Axiomatic Design*. New York: Oxford University Press.

Cutcher-Gershenfeld, J., F. Field, R. Hall, R. Kirchain, D. Marks, K. Oye, and J. Sussman. 2004. Sustainability as an Organizing Principle for Large-Scale Systems. Paper presented at the MIT Engineering Systems Symposium, 29-31 March, at Cambridge, Mass. http://esd.mit.edu/symposium/monograph.

Ulrich, Karl T., and Steven D. Eppinger. 2000. *Product Design and Development*, 2nd ed. New York: Irwin/McGraw-Hill.

Utterback, J. M. 1994. *Mastering the Dynamics of Innovation: How Companies Can Seize Opportunities in the Face of Technological Change*. Boston: Harvard Business School Press.

von Meier, Alexandra. 2003. California Energy Commission: Power Delivery Systems Overview, June 4. http://ciee.ucop.edu/dretd/03_06_04_Power.pdf

Watts, D. J. 1999. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton, N.J.: Princeton University Press.

Watts, D. J., and S. Strogatz. 1998. Collective Dynamics of ”Small-World” Networks. *Nature* 393(4 June):440-442.

Weick, K., K. Sutcliffe, and D. Obstfeld. 1999. Organizing for High Reliability. *Res in Org Beh* 21:81-123.

Whitney, D. E. 1996. Why Mechanical Design Will Never be Like VLSI Design. *Research in Engineering Design* 8:125-138.

Whitney, D., Q. Dong, J. Judson, and G. Mascoli. 1999. Introducing Knowledge-based Engineering into a Tightly Coupled Product Development Environment. Paper presented at the ASME Design Engineering Technical Conferences, 12-16 September, in Las Vegas, Nev.