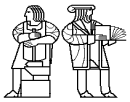


Massachusetts Institute of Technology
Engineering Systems Division

Working Paper Series

ESD-WP-2003-01.25-*ESD Internal Symposium*

THE ANATOMY OF LARGE SCALE SYSTEMS

Joel Moses

MAY 29-30, 2002

The Anatomy of Large Scale Systems

Joel Moses

April, 2002

"The goal of this course is for you to learn how to create complex systems while maintaining robustness, efficiency, extensibility and flexibility."

> Introduction to MIT's first Computer Science subject, 6.001, by Professor Eric Grimson

Introduction

Many theoretical analyses of systems emphasize their behavior. In this paper we shall emphasize the role of organizational structure in influencing certain aspects of the behavior of systems, rather than the full behavior of the systems. There are several historical examples where structure was analyzed early on in order to gain a better understanding of systems. In medicine, for example, anatomy was studied well before we had a deep understanding of the role and behavior of subsystems or infrastructures of the body, such as the liver and blood flow. Different generic structures or architectures provide different advantages and disadvantages in coping with changes in the overall environment in which an evolving system is expected to operate during its lifetime. We shall discuss some of these advantages and disadvantages for various generic structures or architectures. One difficulty in discussing systems issues is the lack of a relatively precise language and concepts for dealing with such issues. We propose that abstract algebra has at least some of the needed concepts.

Our emphasis here is not on the parts of the system that deal with meeting the basic requirements of the system, nor on issues such as cost and efficiency, but rather the additions to the system's requirements that are needed to meet the 'ilities,' such as robustness and flexibility. We agree with John Doyle[2] that much of the complexity of engineering systems is due to such requirements that are often not imposed at the beginning of a new technology, when people are impressed that something works at all, or even later when they are concerned about reducing the cost and increasing the performance of the system. Rather it is a concern over extending the useful lifetime of a system or a family of systems that usually adds greatly to the scale and complexity of a system or the enterprises that design, manufacture and maintain it.

We emphasize a particular ility, namely flexibility. A large scale complex engineering system can be expected to evolve by undergoing many changes in its lifetime. We assume that the initial design meets the basic requirements of the system, but does it make it easy to change the system later on? Flexibility is the ability of a system to be modified relatively easily in certain classes of situations. Flexibility is related to other system aspects and properties, such as complexity and architecture. We believe that one of the key concerns people have about complex systems is that it appears difficult to make even small changes in them beyond a certain point. That is, complex systems appear to be inflexible. In our view all systems possess some measure of complexity, but how much complexity and how inflexible the system becomes after a number of relatively small changes is a function of its initial structure or architecture and the types of changes made later on. There are a few generic architectures that have been around for millennia. These include the family or team architecture, the network architecture and two hierarchical architectures – the tree structure and the layered structure. Well-designed architectures of particular systems will tend to rely on a mix of these generic architectures. Each of the generic architectures has advantages and disadvantages relative to the others that we shall mention.

We begin by noting that the hierarchical architectures tend to be related to philosophical and cultural values, so that different societies may opt to emphasize one of these architectures over another. Before we discuss the various generic architectures and their relationship to flexibility and complexity, we wish to mention mathematical concepts, such as the notion of extensions, which can help us understand the power of certain architectures. Several branches of mathematics have been employed in order to understand engineering systems, such as analysis, probability theory, combinatorics and logic. Abstract algebra, from which much of modern mathematics derives its power, is rarely employed in engineering, and we believe that its concepts have much to add to our understanding of engineering systems.

Holistic thinking and Philosophy

One of the issues often associated with thinking about large scale engineering systems is holistic thinking. This is usually contrasted with reductionistic thinking. These two modes of thought are related to problem solving strategies as well as organizational forms, such as the hierarchies noted above. What we shall point out here is that the contrast between these two modes of thought is very old, and is at least in part associated with differences in philosophy and culture.

The two great philosophers of antiquity are Plato and Aristotle. Aristotle was Plato's student, but other than the fact that both were rational Greek thinkers they seemed to have had little in common. Plato belonged to the aristocracy in Athens, and Aristotle's father was a doctor, thus placing him in the middle class. Plato believed intensely in cooperation, and Aristotle seemed to trust competition more, although probably not as much as we now do. Aristotle was a reductionist. A reductionistic approach to problem solving (breaking problems into smaller and smaller subproblems and integrating the

results) usually leads to a tree structure of subproblems. Note therefore Aristotle's use of tree structures to classify the plants and animals in the Greek islands, and his use of tree structures to describe the political structures in the Greek city-states. Plato, on the other hand, was holistic. He and Socrates were interested in issues affecting the entire society, such as justice. Plato's utopian society in *The Republic* was based on three layers. Layered structures are, we claim, holistic since each layer represents a horizontal cut across the whole society or across significant parts of an engineering system. Layered systems promote cooperation across the layers, whereas tree structured systems are consistent with competition between different parts of the tree.

Aristotle is best described as a natural scientist, and scientists, as the line goes, know more and more about less and less. Plato is best described as a mathematician, and in particular an algebraist. Algebra, the key to mathematical abstraction, is holistic in nature since its equations, for example, hold throughout a space. What is unfortunate about these two philosophers and their followers for the past 2400 years is that they did not recognize that both approaches are useful, depending on the circumstances. Since reductionistic thought is well understood in our culture, our emphasis below is on holistic thought.

Holistic Thinking and Mathematics

Science and engineering rely on mathematics, in particular the mathematical branch of analysis and the related branch of probability. Both of these branches usually assume continuous variables. In addition to understanding continuous systems, the design of engineering systems normally relies on understanding discrete parts, their interconnection structure and their interaction. The relevant mathematics for studying discrete elements includes logic and combinatorics. Logic and the process of decomposition are closely related since logic tends to break problems into cases and carefully examines the validity of each case. Combinatorics also considers cases, but emphasizes counting the number of elements in each case.

The mathematics most closely related to holistic thinking about systems of discrete elements is, in our opinion, abstract algebra. We grant that many people use the term 'holism' in a broader and looser fashion than we do. Nevertheless, we believe that a relatively precise approach to holistic thinking will turn out to be useful.

Much of the power of modern mathematics in fields such as algebraic geometry and algebraic topology is derived from the concepts and methods of abstract algebra. Linear algebra is heavily used in applications, but it lacks the descriptive power of abstract algebra. Group theory is sometimes used in physics, but it too does not have the full power of abstract algebra. The fundamental concept in abstract algebra is the notion of an abstraction, and in particular the notion of an extension (the process of creating an extension is sometimes called 'lifting'). We tend to think of a module composed of parts as an abstraction of the underlying components. An extension creates a set of related

modules. Whereas a decomposition makes a vertical cut of the space of alternative designs, an extension makes a horizontal cut of the space

The power of abstractions and extensions is two-fold. Consider a personal computer containing a microprocessor. If one is careful to maintain the standards of inputs and outputs to the microprocessor, thereby maintaining the integrity of the abstraction, then one can replace a broken one with another with little difficulty, thus achieving a degree of robustness. If one is even more careful in designing the computer, then one might be able to replace the microprocessor with a faster one of the same general family. Much more important is the fact that with a microprocessor one can program the computer system at a higher level of abstraction, the extension space, than without it. That is, an extension simplifies the design domain, and permits humans to solve new problems more readily than without it.

Abstractions, Modules, Extensions and Layers

The power of an abstraction is usually derived from creating a simplified domain where a set of details is hidden. A module is an abstraction in this sense if one can use the module without having to know its internal details. Yet the notion of a module does not fully encompass what is meant by an abstraction in abstract algebra. That is because many abstractions, in particular extensions, apply to sets of objects, rather than a single object, however complex. Consider the set of integers or even more simply just the binary numbers 0 and 1. One abstraction of the binary numbers is the set of polynomials, in x say, with binary coefficients. Let us call the set P(x). Consider an abstraction of P(x), the set of polynomials in y, say, whose coefficients are in P(x). Let us call it P(x,y). Note that in P(x) we can refer to either binary number as a coefficient. Likewise in P(x,y) we can refer to any polynomial in P(x) as a coefficient. This large number of references in P(x,y) to lower layers gives one an advantage that we relate to flexibility. For us flexibility often arises from the ability to switch from one path of interconnections to another. While mathematics does not clearly indicate connections, such as between the set of polynomials and the set of their coefficients, most of the physical world requires them. We show such connections in Figure 1.

Whereas the set of polynomials even with just binary digit coefficients is infinite, the set of operations on such polynomials that are of interest is quite small. Thus another hierarchy related to that in Figure 1 is the hierarchy of operations, such as addition and multiplication on the integers, on polynomials in one variable with integer coefficients, and on polynomials in two variables. We show the interconnections between such operations in Figure 2.

Taking a set of related objects or modules and creating a higher layer of abstractions is called, as we have previously noted, an extension. The ability to do so is called extensibility. Whereas an abstraction that forms a module is essentially a many-to-one operation, an extension usually is a many-to-many operation. The notions of base and extensions are relative, depending on the circumstances.

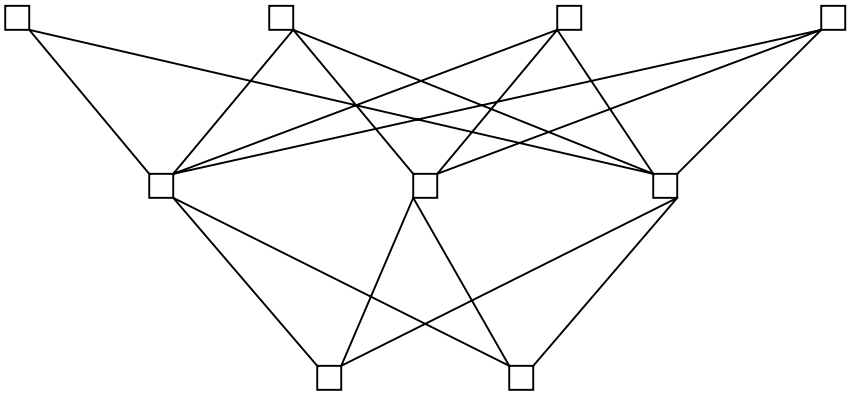


Figure 1
Interconnection pattern between polynomials in P(y,x), P(x), and the binary numbers 0 and 1

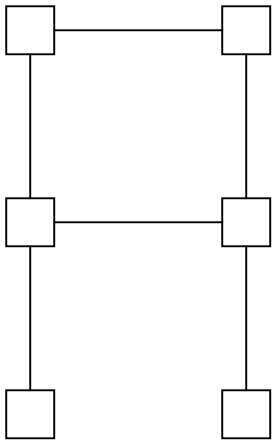


Figure 2
Interconnections between procedures for addition and multiplication of binary numbers, polynomials in one variable, and polynomials in two variables

Computer software lends itself nicely to abstractions, in particular to extensions. The relationship between a high level language and assembly or machine level language is a many-to-many type of extension. Statements in the high level language can refer to several machine level operations and different statements can refer to different subsets of the set of machine language operations. A set of related procedures can form a software package with standard interfaces and assumptions about data. A database system can be considered a package in this sense. Software levels that rely on a database package can be said to have a many-to-many relationship with the procedures that compose the database package itself.

Platforms

If we continue our analogy to algebraic extensions in the hardware world, we are looking for two sets, one set of related elements that can form a base, and another set of related elements that can act as an extension of it. Platforms have characteristics that make them into a base set of elements. For example, assume we have a hardware platform with some possibility for variation in its length, for example. This set of similar platforms is akin to the set of integers in our example above. Now consider a family of related products, such as cars, watches or Walkmans, which contains the set of platforms as a base. Let us call them the platform extensions. In some cases, the platform extensions will be connected to just one element in the base platform set. In other cases there may be several extensions of the same platform. Overall, the interconnection pattern between the extension set and the base set is many-to-many. The advantage of such an approach is the flexibility that is available in the overall design space. I am reminded of the question asked in my music appreciation class. “How come Mozart was able to write so many symphonies when he was so restricted by the Sonata form?” The answer was “He was able to write so many symphonies because he was restricted by the Sonata form.” One disadvantage of the platform approach, as indicated by the question, is that it has restrictions on the freedom of the designer. But restricted freedom at one level can be turned into ease of design at a higher level.

Another trade-off is between platform-based design and efficiency. There is indeed some loss of efficiency in relying on the standards required by platforms. Opponents of the approach often make this argument. Improved practice with platform standards can reduce the loss in efficiency. In some cases, however, one may have to violate the standards and make direct interconnections into a platform’s or a module’s components in order to gain efficiency. This is called an abstraction violation in computer science. Abstraction violations should be used with great care since they may reduce the flexibility of the resulting system and increase its complexity. The places where it is most advantageous to make an abstraction violation can be determined by analysis or simulation.

Infrastructures

Some of the largest scale engineering systems are infrastructures, such as the electric power system or the telephone system. Several of these infrastructures may be viewed as layered systems, transferring physical or informational content from producers to consumers via a distribution network. In the electric power case there are multiple plants, such as nuclear power plants or coal plants, which produce electric power. This electric power is transmitted over a network and distributed to the end users, be they homes, office buildings or factories. The telephone system is interesting in that the set of producers of calls is nearly the same set of consumers of telephone calls. An exception would be automatic calling machines that call us during dinner-time. These machines might not answer calls as well.

Due to the large geographic extent of many physical infrastructures it is not efficient for the set of producers to be directly connected to the set of consumers. At a national or regional level one tends to find network structures as the distribution architecture. Locally one tends to see tree structures, due to the high cost of alternative architectures. Thus, the distribution system tends not to be very robust locally, whereas regionally and nationally much attention is often paid to robustness and flexibility.

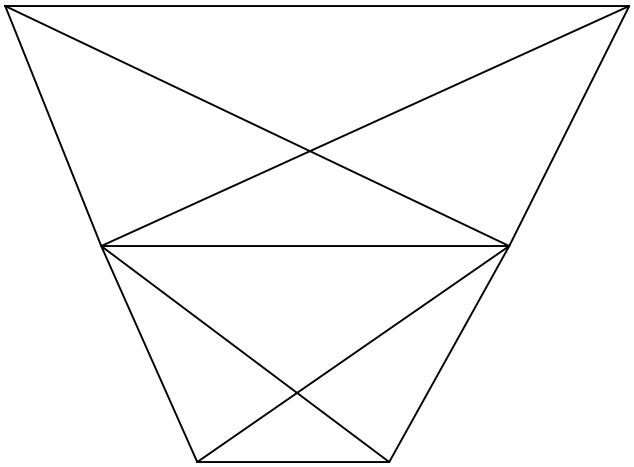


Figure 3

An infrastructure with several producers at the lower level, an architecture for transmission and distribution at the middle level, and consumers at the upper level

Internal Networks

Many engineering systems rely on internal networks for communicating information as well as various forms of energy and matter. As in the infrastructure examples above, these internal networks are usually in the middle layers of the system. For example, there is an electrical network in a car that is powered by a 12-volt battery. Recently our colleagues in the Ford-MIT Alliance worked on standards for a similar network using a 36-volt battery. Such batteries will be needed in future cars since many more subsystems in these cars will require electricity. Because a change of this kind was not anticipated in previous car designs, several modifications and new standards were needed to accommodate the new batteries. Thus, while a network approach and its attendant standards provides flexibility by permitting new attachments to the network that meet the standards, significant changes in intermediate standards, such as occurred in the 36-volt battery case, can have wide-spread ramifications.

Middleware in Software

Recall that with an abstraction one might be able to replace either the base or the extension. Large scale software systems will usually have several components making up a base or bases. Often it is desired to introduce yet another component into the base or one of the extension layers. This integration step can be difficult unless the new components use standards that are also used by the existing components. In recent years a technology has been developed to make the creation and utilization of such standards relatively easy. It is called middleware. IBM's Chairman Gerstner has recently called middleware the fastest growing segment of the software business[4]. One example of a software standard that facilitates this approach is CORBA, the Common Object Request Broker Architecture. As can be surmised, object-oriented programming is usually relied upon in software that uses middleware standards. Large scale enterprise software can often be divided into three layers with the bottom layer containing data bases, the middle layer containing business-related processes, and the top layer containing report generators and graphical user interfaces. Middleware is often used to integrate new business processes, but can be used in aspects of all three layers[1].

Ironically the first example of the use of middleware was probably in Project Athena. Project Athena was designed for hundreds of computers that relied on a base of UNIX software and at least two underlying hardware platforms. Security components, such as Kerberos, had to be designed by the Athena staff for this base. The irony comes from the fact that operating systems tend to integrate many components into themselves, thus making the distinction between a base and an extension almost non-existent. Nevertheless, some people (e.g., the NSF Middleware Initiative) use the term middleware for the layer of software between the network and applications. This layer includes services for authentication and security. Just as Athena used multiple hardware platforms, this software promotes standards and interoperability.

Coherence

One of the goals of Project Athena was coherence. By this we meant the ability of two or more processors on its internal network to understand each other's messages. Part of this goal was achieved in the web because computers can send messages to each other readily, and at least understand keywords in the text. But this is generally not enough to integrate two subsystems. Abstract algebra is concerned with determining the least extension needed that is simultaneously an extension of one base and a second base. Such a view may be helpful in integrating subsystems of engineering systems.

Complexity, Flexibility and Generic Structures

In the following we discuss briefly the concepts of complexity, flexibility and their relationship to generic organizational structures, such as networks, tree structures and layered structures.

Complexity

Joe Sussman in this symposium and Seth Lloyd both claim that there are dozens of definitions of complexity. Since our emphasis here is on the structure of systems, there are fewer alternative definitions. The key definition, in our opinion, for structural complexity is due to Kolmogorov. He defined complexity as the length of the shortest description of the system. One problem with this definition is that it is difficult to generate the shortest description. This definition is a relative one in practice, since one tends to rely on the shortest description that a person can generate, and this description clearly varies from individual to individual. Fortunately, given our interest in abstractions and flexibility and their relationship to generic structural forms, there are simpler approaches to structural complexity than Kolmogorov's that will suit our purposes. Suppose we start with a system S, what will be the resulting complexity if we add a few new interconnections between existing parts or add a few new parts with attendant interconnections and create a system S'? We do not claim to have a simple answer to the general question, except to note that if S' has an interconnection architecture that does not significantly vary from that of S, then the complexity of S' is not very different from that of S, except in unusual cases where, for example, the pattern becomes easier to describe. Conversely, if the new interconnections change the interconnection architecture so that it is no longer consistent with the original generic architecture, then the complexity will be increased much more. For example, in a tree structure, it is legal to add new child nodes to a given node, but not to add new parents or to connect to siblings. The latter kinds of interconnections in a tree structure will markedly increase the complexity of the resulting structure.

Flexibility

Flexibility has not been studied to the same extent as complexity, and has not developed as varied a set of definitions. For us a system is flexible if one can implement classes of changes in its specifications with relative ease. Note that we do not claim that even small changes in specifications always result in ease of implementation, just that the system is designed to handle certain classes of changes in an easy manner. How does this translate to system implementation? Again we do not yet know the general answer. However, if we consider the paths of interconnections in a system, a system with many alternate paths from the inputs to the outputs will usually be able to handle certain changes in specifications relatively well. Moreover, a system with many alternate paths can increase the number of paths a great deal with a small increase in the number of interconnections and additions of new nodes or modifications of existing ones. We shall therefore define the flexibility of a system as the number of paths in it, counting loops just once. In some architectures, notably the network structure and the layered hierarchy, the number of potential paths that keeps the generic structural form unchanged is very high. This, however, cannot be said of the tree structured hierarchy.

A large number of paths indicates that there are multiple choices that can be made inside the system. This would normally be the case when the system is 'programmable' in an intuitive sense. A system with multiple settings for a user driven switch will usually have multiple paths. Programming is one way in which the outside environment can cause changes in a system. Another way is to introduce new nodes, modify existing nodes, and create new interconnections between nodes. When the system is a layered hierarchy that is carefully designed based on prior experience with related, albeit simpler systems, the changes one is likely to wish to make during the lifetime of a new system may be implemented by small changes at the top level of the hierarchy. Such an approach will not increase the system's complexity much and will often provide additional paths to support future changes.

There are a number of papers in this symposium that deal with flexibility in one way or another. Hence it is useful to categorize the differences in the assumptions about flexibility that we believe are being made. Our assumption is that most changes are small, and that there are numerous such changes in a given period of time. The goal is to permit such changes to be made readily. We believe that people are most frustrated when apparently small changes to an evolving system are difficult to make. Others appear to assume that the changes to a system are relatively few in number in a given period of time and of relatively great magnitude and cost. There are systems, of course, for which all such assumptions can hold, and that require a variety of approaches to design.

Generic Organizational Structures

Below we discuss three major generic organizational structures – the network structure and two hierarchical structures, the tree structure and the layered structure. Large scale

systems will employ a variety of specific examples of structures. Thus it is useful to understand the advantages and disadvantages of each generic form. We do not discuss here the team structure in which all nodes are connected to each other, since such structures rarely model large scale systems due to the quadratic growth in the number of interconnections.

Networks

Networks are by nature quite flexible, given our definition. The telephone network, for example, permits one to call over one hundred million other phones. Instead of having a number of interconnections that is one half of the square of the number of phones, connecting each phone directly to each other one, the total number of interconnections is a small multiple of the number of phones. This reduces the overall cost and complexity of the system. The structure of the interconnections varies in the local loop, the regional centers, and the national centers. A national center for a long distance telephone carrier will control broadband connections between major cities. There will be multiple paths in this particular national network of connections. This has significant advantages. If one line is down or overloaded one may reroute calls through another path in the network. Moreover, the investment in a new interconnection between existing regional nodes will increase the total number of paths, and thus increase the flexibility and robustness of the overall system. Changes in the system's configuration, which permits new lines to be installed, have been fairly well automated and are indicative of the overall system's flexibility.

The Internet can be viewed as an extension of a physical network. It relies on an addressing scheme that is an abstraction of a telephone number. The World Wide Web relies on the truly gigantic number of virtual paths in the Internet in order to access data. Adding a new node to the Internet has become a routine operation because the interface standards are fairly well understood, and such additions increase the overall flexibility of the network. Note that adding a new node to the Internet increases the complexity of the interconnection structure, but even if add a number of new nodes, the complexity of the Internet will not grow a great deal since the topology of the system permits such new interconnections. What does increase the complexity is a significant change in topology, such as would result from the addition of subnets and routers.

In summary, networks have the property that they are quite flexible and their flexibility usually grows with the addition of new interconnections. Moreover, most new interconnections do not increase the overall network complexity a great deal. A key weakness of this structure, however, is that it is difficult to control. Thus regional and national centers may be needed to achieve a degree of control over a network. A related weakness is that the number of hops between any two nodes may be nontrivial, thus reducing the overall efficiency of the system.

Tree Structures

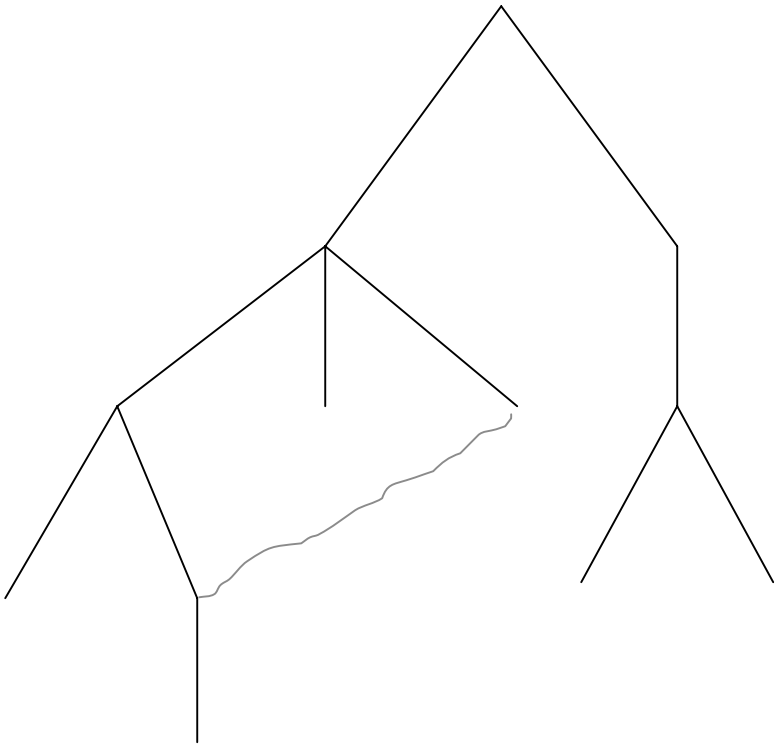
Tree structures are so common in our culture that people equate all hierarchies with tree structured organizations. Tree structures arise when solving problems using a reductionistic approach. Thus we break a problem into smaller and smaller subproblems, and integrate the solutions of these subproblems to arrive at a solution of the whole. Tree structures are common, partly because it is relatively easy to break problems up into subproblems in various ways. Tree structured human organizations are consistent with organizational cultures that value internal competition. Tree structures also have better control than networks and permit very large scale systems to function. Herb Simon probably had the deepest understanding of the advantages of tree structures, and his short chapter on complex systems is a classic[5,6].

Consider the number of paths in a tree structure beginning with the parent node. The number is exactly the number of final nodes in the tree, which is fewer than the total number of nodes in the tree. This is a very low measure of flexibility in contrast to the other generic architectures that we consider. In fact it is by far the lowest, and it is reasonable to say that tree structures are inflexible using our definition of flexibility. When one integrates the solutions to subproblems using the classic divide-and-conquer approach, the structure that results is usually not a tree any longer, but unless one has a design strategy that emphasizes flexibility and component reuse, then it is most likely that the resulting system will still be relatively inflexible. Moreover, if we consider human organizations that begin as tree structures, then as these organizations evolve, the additional interconnections between people in the firm (which are often across the tree and hence not 'legal' in the structure) will tend to result in greatly increased complexity of the connection structure with relatively little increase in flexibility due to increased number of paths (see Figure 3). This double whammy is, we claim, characteristic of tree structures and lead to the widespread belief that large hierarchical organizations (namely tree structured organizations in our culture) are inherently inflexible. As we shall soon see, the opposite is actually true.

One approach of dealing with the complexity that arises from the illegality of many changes in a tree structure is to ask the system to be redesigned largely from scratch with any modifications in its requirements. This is extremely expensive for most large scale systems, and is therefore not the approach of choice.

Layered Hierarchies

Based on our analysis, layered structures are holistic in character, and have a high degree of flexibility. As the layered systems evolve, either as human organizations or engineering systems, the flexibility usually tends to grow while keeping the increase in complexity in check. The algebraic extensions that we noted earlier are layered hierarchies, and the number of paths or potential for paths in them tends to grow geometrically with the number of layers. Moreover, when the layers are carefully designed the changes one is likely to make can usually be implemented as new nodes within existing layers (often the top layer), or new connections between existing layers.



A tree of depth 4 with 12 nodes with one non-standard interconnection

Human organizations that are layered usually have three layers. Examples are large law firms that have senior partners, junior partners and associates. Historically universities were based on a three-layered hierarchy of full professors, associate professors and assistant professors. This changed in the 19th century when departmental structures were overlaid on the layered one. As a former dean I can see the advantages and the disadvantages of the departmental structure. A key advantage comes from specialization, and a key disadvantage comes from competition between departments for resources and their resistance to cooperation. An advantage of many layered human organization is that cooperation tends to be quite common, and teams that operate quickly and effectively can be readily formed. A disadvantage is that a high degree of specialization may not be quite so desirable in such organizations. Jay Galbraith discusses related structures under the term 'flexible lateral organizations.'[2]

Engineering systems that are layered will usually be quite flexible and be able to handle classes of changes readily without growing too complex. This is shown in the number of paths in such systems, which grows geometrically with the number of layers and is far

higher than the corresponding number of paths in a tree structured system with a similar number of nodes. One cost of the increased flexibility is some loss of efficiency. This occurs when the system creates abstraction barriers between the layers. For example, a microprocessor interprets the bits coming into it from the layers above. The interpretation takes some time, and could have been eliminated with direct connection between the machine language instructions, now implemented in hardware, and the parts of the microprocessor that actually perform the instruction. If we had such direct connections, then it would have been difficult to change the implementation of the instruction processing at some future time. If the loss of efficiency is very great then simulation can show places where it might be advisable to break through the barrier and make direct interconnections. That is, we trade-off some flexibility for efficiency. Moore's Law has saved microprocessors from such fates, and specialized computer hardware has tended to be replaced with relatively general chips that possess abstraction barriers, but this is not the general case, of course.

Summary

The mathematical field of abstract algebra can provide a language for discussing systems taken as a whole. A structured or anatomical view of engineering systems when coupled with concepts and intuition from abstract algebra can give us a relatively deep understanding of certain systems issues, such as flexibility. We believe that people are frustrated when it is difficult to make even small changes to a large scale system. We claim that appropriate architectures can result in systems that can permit us to make many changes in the system without undue cost or undue growth in the resulting complexity of the system. We quantify this claim by providing a definition of flexibility that permits one to measure the flexibility inherent in a given architecture. Based on such definitions we claim that the classic hierarchy, a tree structured hierarchy, tends to be difficult to change while maintaining the integrity of a tree structure and keeping the overall complexity of the system under control. Layered hierarchies have significant advantages over tree structures in terms of flexibility, but have disadvantages, for example, in terms of efficiency. Such disadvantages usually make layered systems inappropriate for small scale systems. There is no ideal solution for all design issues, and thus a mixed strategy for design is required.

We recognize that many engineering systems will not undergo great change at any given time. Nevertheless, some systems or subsystems will change significantly at various points, and analyses such as ours can give the architects of such systems means of creating long-lived and flexible designs.

References

1. Chris Britton, *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*, Addison-Wesley, 2001
2. Csete and Doyle, Science, vol. 295, no. 5560, March 1, 2002, p.1664
3. Jay R. Galbraith, *Competing with Flexible Lateral Organizations*, 2nd edition, Addison Wesley, 1994
4. IBM Annual Report, 2001
5. Herbert A. Simon, *Sciences of the Artificial*, 3rd Ed., MIT Press, 1996
6. Herbert A. Simon, "Complex Systems," in Computational and Mathematical Organization Theory, Kluwer, 2001, pp.79-85