Research in Engineering Design (1998)10:189–209
© 1998 Springer-Verlag London Limited

Research in
Engineering
Design

Axiomatic Design Theory for Systems

Nam P. Suh

The Ralph E. & Eloise F. Cross Professor, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract. *A general theory for system design is presented based on axiomatic design. The theory is applicable to many different kinds of systems, including machines, large systems, software systems, organizations, and systems consisting of a combination of hardware and software. Systems are represented by means of a system architecture, which takes the form of the* {FR}/{DP}/{PV} *hierarchies, a 'junction-module' diagram, and the 'flow diagram'. The 'flow diagram' for system architecture concisely represents the system design, the relationship among modules, and the control sequence in operating systems. The flow diagram of the system architecture can be used for many different tasks: design, construction, operation, modification, and maintenance of the system. It should also be useful for distributed design and operation of systems, diagnosis of system failures, and for archival documentation.*

Keywords: Architecture; Axioms; Independence; Information; Design; Systems; Theory

1. Introduction

What is a system? A system may be defined as

> an assemblage of sub-systems, hardware and software components, and people designed to perform a set of tasks to satisfy specified functional requirements and constraints.

Engineers build systems. Machines, airplanes, software systems, and automobile assembly plants are systems – albeit systems of different kinds – and each has sub-systems and components. Systems often consist of hardware, software and people. Such human-made systems must be designed, fabricated and operated to achieve their intended functions. Each of these systems performs many functions. Some, like the automobile, perform a large number of different but dedicated functions. Others, such as a job-shops, perform a variety of different functions during their lifetimes and as such may be categorized as large flexible systems [1].

The design of effective systems is the ultimate goal of many fields, including engineering, business, and government. Yet system design has lacked a formal theoretical framework and thus, has been done heuristically or empirically [2]. Heuristic approaches emphasize qualitative guidelines – exemplified by use of the phrases 'Murphy's laws', 'make it simple' and 'ask five why's'. After systems are designed, they are sometimes modeled and simulated. In many cases, they have to be constructed and tested. All these very expensive and unpredictable processes are done to debug and improve the design after heuristic design solutions are implemented in hardware and software. Such an approach to systems design entails both technical and business risks because of the uncertainties associated with the performance and the quality of a system that is created by means of empirical decisions.

Some people use dimensional analysis, decision theory and others to check or optimize the system that has already been designed. There are three issues with this approach. First, they do not provide tools for coming up with a rational system design beginning from the definition of the design goals. Secondly, some of these methods simply confirm the result, if systems are correctly designed. For example, any physical system that is properly configured should satisfy the π-theorem – a correctly designed physical system should always satisfy the π-theorem. Thirdly, they are not general principles for system design since they cannot be applied to non-physical systems such as software and organizations.

Systems with many functional requirements (*FR*s), physical components and many lines of computer codes can be complex, in the sense that the probability of satisfying the highest *FR*s decreases with increase in the number of *FR*s and design

Correspondence and offprint requests to: N. P. Suh, The Ralph E. and Eloise F. Cross Professor, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

parameters (*DP*s). One of the goals of axiomatic approach to design is to reduce this complexity by being able to make right design decisions at all levels. From the axiomatic design point of view, the design of systems is not fundamentally different from the design of simple mechanical products and software. All of them – systems, machines and software – must satisfy functional requirements, constraints, the Independence Axiom and the Information Axiom. However, the specific relationship among functional requirements, design parameters, and process variables are different depending on what the system must do. Also, the governing physical laws may also be different depending on the *FR*s the system must satisfy, the nature of the *DP*s chosen, and the Process Variables (*PV*s) employed to achieve the design objectives.

Notwithstanding its obvious importance, no effective way of representing system architecture has been advanced. The situation is equivalent to trying to build an electronic instrument without a circuit diagram. Just as an electrical system is described by an electric circuit diagram, a system – either physical or software or a combination thereof – must be represented by a diagram that shows how the 'modules' must be put together and how the systems should be operated. In this paper, the concept of 'module' is defined, and the 'system architecture' is represented in several forms: {*FR*}/{*DP*}/{*PV*} hierarchies, 'junction-module' diagrams, and the 'flow diagram'. The System Control Command (SCC) provides the operational and procedural instruction for operation of the system. Some of these ideas were presented elsewhere [3].

The design of the system must be documented to capture all its relevant information. The document must show the complete structure of the system and the relationships among the decomposed *FR*s, *DP*s and *PV*s. The relevant information about the system includes the specific functional requirements for which the system is designed, the rationale for choosing specific design parameters and process variables, and the causality relationship between the *FR*s, *DP*s and *PV*s. Also, the system architecture must be clearly known to construct the system, distribute the design responsibilities, track the effect of engineering change orders, create a maintenance procedure, organize and coordinate the complex tasks of executing the project management.

Maintenance of complex systems is expensive – the annual cost being roughly 10% of the system cost. Documentation of the system is needed to provide the means of determining the causes of system failures or detecting impending failures. It should provide ready instruction for identifying the components to be checked when the system does not perform certain functions. Effective documentation is also useful for other purposes: tracking design history, improving system performance through modification, and creating a knowledge-base related to the system.

The purpose of this paper is to present a theory for system design and address some of the key issues in designing systems. The axiomatic design theory for systems has been applied to many different systems such as manufacturing systems [4], software [4–6], products [4] and organizations [4]. They are not given in this paper because of the limited space.

For those who are not familiar with axiomatic design theory, the basic elements of the theory, upon which the system design theory presented in this paper is based, are presented in Appendix A.

2. Issues Related to System Design

There are several key issues in system design that will be addressed in this paper:

1. How should a complex system be designed? Some argue that it can only be designed from the bottom up. In this paper, a top-down approach based on axiomatic design is used. It is shown that a system can be designed within the basic framework of axiomatic design theory and methodology. The final design of the system can be constructed from the modules associated with the lowest-level leaves of the *FR* and *DP* hierarchies and design matrices.
2. How should the complex relationships between various components of a system be coordinated and managed? In this paper, it is shown that this can be done using the Flow Diagram of the system architecture. This is the basis for dealing with distributed project management, Engineering Change Orders (ECOs), and the maintenance of the system.
3. How can the stability and controllability of a system be guaranteed? A system must be stable and controllable. If the system is not stable and randomly changes, it will not be reliable and controllable. If the system is not controllable, it cannot satisfy functional requirements at all times. In this paper, it is argued qualitatively that when a system satisfies the Independence and the Information Axioms, the system is stable and controllable within the specified tolerance.

4. What is the role of human operators in a system? People are an integral part of many systems. Human-machine systems is an important consideration in these systems, since the performance of human beings is a determinant factor in the outcome of the system.

3. Classification of Systems

Notwithstanding the fact that systems can be designed using the same set of basic principles, it is nevertheless useful to classify systems. A proper classification of systems will shift the thinking and focus from conventional identifications to a functionally based identification of systems.

The conventional way of characterizing systems is based on the physical size of the system. However, when making design decisions, physical size is of less significance than the number of functional requirements that the system must satisfy at the highest level and the number of levels of decomposition required to arrive at a complete design solution. Therefore, a better classification scheme is to define the systems according to the number and the nature of *FR*s the system must satisfy.

Functional classification of systems can be done using a number of different features or characteristics: large systems from small systems, static systems from dynamic systems, fixed systems from flexible systems, passive systems from active systems, and automated systems from manual systems. Some systems cannot be classified in a simple way, e.g. large, flexible, dynamic systems. Some systems are open systems – systems whose constituents change throughout their lifetimes – in contrast to closed systems that are made up of the same components at all times. Examples of open systems are factories, universities and many machine tools; closed systems include inertial guidance systems and television sets.

The axiomatic design theory can be applied to all these systems with slight variations.

4. Axiomatic Design Theory for Fixed Systems

Consider the design of a closed fixed system which is defined as a system that has to satisfy a fixed set of functional requirements at all times and whose components do not change as a function of time. Many machines and robots may be classified as closed fixed systems, since they are designed to satisfy a fixed set of functions at all times in contrast to a flexible system whose functional requirements change as a function of time. Closed fixed systems can be complex or simple depending on the difficulty or ease in achieving the functional requirements.

4.1. The First Step in Designing a System: Define FRs of the System

The first step in designing a system is to determine the Customer Needs (*CN*s) or attributes in the customer domain that the system must satisfy. Then, the *FR*s and Constraints (*C*s) of the system in the functional domain are determined to satisfy the customer needs. The *FR*s must be determined in a *solution neutral environment* – defining *FR*s without thinking about the solution – so as to come up with creative ideas. *FR*s must satisfy the customer needs with fidelity. It is important to remember that the *FR*s are *defined* as the minimum set of independent requirements that the design must satisfy.

4.2. Mapping Between the Domains: A Step in Creating System Architecture

The next step in axiomatic design is to map these *FR*s of the functional domain into the physical domain – conceiving a design embodiment and identifying the *DP*s. *DP*s must be so chosen that there is no conflict with the constraints. In the case of products, *DP*s may be physical parameters or parts or assemblies, whereas in the case of software, *DP*s may be software modules or program codes. Once the *DP*s are chosen, designers must go to the process domain and identify the Process Variables (*PV*s) based on the creation of a new process or the use of an existing process. When existing machines are to be used, the *PV*s are given, and therefore, *PV*s act as constraints since we are not free to create new processes and choose new *PV*s. In the case of organizations, the *PV*s are typically resources – both human and financial. In the case of software, *PV*s may by subroutines or machine codes or compilers.

During the mapping process, the design must satisfy the Independence Axiom, which requires that the functional independence be satisfied through the development of an uncoupled or decoupled design. In an ideal design, the number of *FR*s and *DP*s is equal, a consequence of the Independence Axiom and the Information Axiom. In choosing *DP*s, we must also be mindful of the constraints and the information content. The Independence Axiom does not require that the *DP*s must be independent nor that each *DP* must correspond to a separate physical piece. For example, a beverage can may have 12 *FR*s and 12 *DP*s, but has only three physical pieces.

The Information Axiom states that the design that has the least information content is the best design. Their information contents can be measured to compare the relative merit of equally acceptable designs that satisfy functional independence. The Information Axiom provides a guide in selecting *DP*s, in addition to providing the selection criteria for best design among those designs that satisfy the Independence Axiom. The Information Axiom is a powerful tool in identifying the best design when there is more than one functional requirement. Through the use of the Independence Axiom and the Information Axiom, one can easily optimize a multi-*FR* design solution.

The mapping process between the domains can be expressed mathematically in terms of the characteristic vectors that define the design goals and design solutions. At a given level of design hierarchy, the set of functional requirements that define the specific design goals constitutes a vector $\{FRs\}$ in the functional domain. Similarly, the set of design parameters in the physical domain that are the 'How's' for the *FR*s also constitutes a vector $\{DPs\}$. The relationship between these two vectors can be written as

$$\{FRs\} = [A]\,\{DPs\} \tag{1}$$

where $[A]$ is a matrix defined as the design matrix that characterizes the product design. Equation (1) may be written in terms of its elements as $FR_i = A_{ij}\ DP_j$. Equation (1) is a *design equation* for product design. The design matrix is of the following form for a square matrix (i.e. the number of *FR*s is equal to the number of *DP*s):

$$[A] = \begin{bmatrix} A11 & A12 & A13 \\ A21 & A22 & A23 \\ A31 & A32 & A33 \end{bmatrix} \tag{2}$$

When the change in *FR*s (i.e. ΔFR) are related to the changes in *DP*s (i.e. ΔDP), the elements of the design matrix is given by

$$Aij = \partial FRi/\partial DPj$$

For a linear design, *Aij* are constants, whereas for nonlinear design, *Aij* are functions of *DP*s. There are two special cases of the design matrix: the diagonal matrix where all *Aij*'s except those $i=j$ are equal to zero, and the triangular matrix where either upper or lower triangular elements are equal to zero. For the design of processes involving mapping from the physical domain to the process domain, the design equation may be written as

$$\{DPs\} = [B]\,\{PVs\} \tag{3}$$

$[B]$ is the design matrix that defines the characteristics of the process design and is similar in form to $[A]$.

4.3. The Independence of System Functions

To satisfy the Independence Axiom, the design matrix must be either diagonal or triangular. When the design matrix $[A]$ is diagonal, each of the *FR*s can be satisfied independently by means of one *DP*. Such a design is called an *uncoupled* design. When the matrix is triangular, the independence of FRs can be guaranteed if and only if the *DP*s are changed in the proper sequence. Such a design is called a *decoupled* design. All other designs violate the Independence Axiom and they are called coupled designs. Therefore, when several functional requirements must be satisfied, designers must develop designs that have a diagonal or triangular design matrix.

There are a large number of corollaries and theorems that can be derived from these axioms. They are listed in Appendix B.

4.4. Information Content for Systems: The Best Design

Among all the designs that satisfy the Independence Axiom, the design that has the least information content is the best design, according to the Information Axiom. The information in axiomatic design is defined in terms of the logarithmic probability of satisfying the functional requirements (see Appendix A). Information content is a relative concept since it is given by the intersection between the tolerance specified by the designer (called the design range in a plot of probability distribution versus the functional requirement) and the tolerance the system can satisfy (called the system range). Thus, the information associated with a given functional requirement is obtained by computing the probability (or uncertainty) of achieving the functional requirement, which is done by determining the common range – the intersection of the design range and the system range.

How do we measure the information content of a system that has many decomposed layers in its hierarchy? In the case of a system design with many layers of *FR*s and with many *FR*s at all levels of the design hierarchy, the information content of the system is simply the information needed to satisfy the highest functional requirements, $\{FRi\}$. To compute the information content of the system, we need to determine the probability distribution of the systems range and the common range to compute the information content (see Appendix A and Suh [7]). Hence, the information content of the system is given by

$$I_{\text{system}} = \sum I_{\text{highest level FRi}} = -\sum \log\,(A_c)_{\text{highest level FRi}} \tag{4}$$

where $(A_c)_{\text{highest level FRi}}$ is the area of the common range associated with each one of the highest level *FRi*.

The probability of satisfying the highest level *FR*s is related to the probability of satifying the lowest levels *FR*s (i.e. leaves), since the lowest level *FR*s yield the highest *FR*s when they are combined according to the instruction given by design matrices. Therefore, the probability of satisfying the highest level *FR*s is given by the product of all probabilities associated with all the lowest level *FR*s in the system architecture. Then, the information content of the total systems is the sum of the information contents associated with all lowest-level *FR*s (i.e. leaves), which may be expressed as

$$I_{\text{system}} = \sum \log(p_{\text{leaf}}) = -\sum \log(A_c)_{\text{leaf}} \tag{5}$$

where $(A_c)_{\text{leaf}}$ is the area of the common range associated with each leaf. Equating Eqs (4) and (5), we obtain

$$\sum \log\,(A_c)_{\text{leaf}} = \sum \log\,(A_c)_{\text{highest level FRi}} \tag{6}$$

Equation (6) is valid only for uncoupled and decoupled designs if the integration of the lowest level modules does not introduce a new element of uncertainty. In the case of a coupled design, it is expected that in most cases,

$$\sum \log\,(A_c)_{\text{leaf}} < \sum \log\,(A_c)_{\text{highest level FRi}} \tag{7}$$

since any change in any other *FR* in the same set of *FR*s at a given level will affect the A_c.

When the integration of the modules introduces a new element of the uncertainty, Eqs (5), (6) and (7) must be modified to account for this additional probability associated with the assembly of modules. Equation (5) should be modified as

$$I_{\text{system}} = \sum \log(p_{\text{leaf}}) + I_a = -\sum \log\,(A_c)_{\text{leaf}} + I_a \tag{8}$$

where I_a is the information associated with assembly of modules. Then, Eq. (7) becomes

$$\sum \log\,(A_c)_{\text{leaf}} + I_a < \sum \log\,(A_c)_{\text{highest level FRi}} \tag{9}$$

The ultimate goal is to minimize the additional information required to make the system function as designed by making all p_{leaf} equal to one, i.e. to minimize the information content as per the Information Axiom. To achieve this goal, the design must satisfy the Independence Axiom. When the design satisfies the functional independence, the bias can be eliminated and the variance of the system range may be made small so that the system range lies inside the design range, reducing the information content to zero. A design that can accommodate large variations in design parameters and process variables and yet satisfy the functional requirements is called a robust design. The Information Axiom provides a theoretical foundation for robust design.

Based on the Information Axiom, we can show that there are four different ways of reducing the bias and the variance of a design to develop a robust design, provided that the design satisfies the Independence Axiom [4,8]. If robust design is practiced at each level of the hierarchy and with each *FR*, then the information content will be minimal, and thus, the design and operation of the system will be done efficiently and reliably.

4.5. Decomposition of {*FR*s}, {*DP*s} and {*PV*s}

The *FR*s, *DP*s and *PV*s must be decomposed until the design can be implemented without further decomposition to create their hierarchies. These hierarchies of *FR*s, *DP*s, and *PV*s and the corresponding matrices represent the system design. The decomposition of these vectors cannot be done by remaining in a single domain, but can only be done through zigzagging between the domains. The decompositon must be done until the design task is completed.

The systems design is represented by three different representation of the system architecture: the $\{FRs\}/\{DPs\}/\{PVs\}$ hierarchy, the module-junction diagram, and the flow diagram. The module-junction diagram and the flow diagram are explained in the next section.

5. Definition of Modules

The concept of modules is important in system design. It must be defined carefully based on basic principles. Otherwise, confusion can arise. For example, a module is not a piece of hardware, although in some cases, by coincidence, it may correspond to a hardware piece. In axiomatic design, module is defined in terms of the (*FR*/*DP*) or the (*DP*/*PV*) relationship.

A module is defined as the row of the design matrix that yields an *FR* when it is provided with the input of its corresponding *DP*. For example, consider the following design equation:

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \tag{10}$$

*M*1 is the module that corresponds to the elements of the design matrix and appropriate *DP*s that yield *FR*1, when it is multiplied by (or related to) *DP*1. Similarly, *FR*2 is obtained when *DP*2 is provided as an input to the module *M*2. *M1* and *M*2 are given by

$$\begin{aligned} FR1 &= a\,DP1 = M1 \,^*\, DP1 \\ FR2 &= b\,DP1 \;+\; c\,DP2 = M2 \,^*\, DP2 \end{aligned} \tag{11}$$

where *M*2 = *b* (*DP*1/*DP*2) + *c*. It should be noted that the design equations given in Eq. (11) are state equations. The differential form of Eq. (11) is

$$\begin{aligned} \Delta FR1 &= a\,\Delta DP1 = M1 \,^*\, \Delta DP1 \\ \Delta FR2 &= b\,\Delta DP1 \;+\; c\,\Delta DP2 = M2 \,^*\, \Delta DP2 \end{aligned} \tag{12}$$

where *a* is the partial derivative of *FR*1 with respect to *DP*1, *b* and *c* are the partial derivatives of *FR*2 with respect to *DP*1 and *DP*2, respectively, and *M*2 = *b* (*ΔDP*1/*ΔDP*2) + *c*.

The above definition of module *M*2 given by Eq. (11) has an advantage in that it simplifies the representaton of the system architecture as shown later in this paper [4].

6. System Architecture

The system architecture of a design is represented in several different forms: the *FR*, *DP* and *PV* hierarchies with the corresponding design equations and matrices, the module-junction diagram, and the flow diagram. The latter two means of representing a system were first introduced for software design [9], which is extended to represent the system architecture. The flow diagram is a concise and powerful tool that provides a road map for implementation of the system design.

6.1. Design Equations for a System Design

Suppose that we have completed a system design such that the *FR* and the *DP* hierarchies are as shown in Fig. 1. At each level of the hierarchies, the design process generates physical embodiment (e.g. drawings, physical mock-ups) or software modules. The physical embodiment is assumed to be done but not shown here. When the design is completed, the result of the design process is the final product, in addition to the complete hierarchies of functional requirements and design parameters.

The design shown in the figure required four layers of decomposition to satisfy the highest level *FR*s. Suppose that the design matrices for the design shown in Fig. 1 are as follows:

$$\begin{gathered} \begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} A11 & 0 \\ 0 & A22 \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \\ \begin{Bmatrix} FR11 \\ FR12 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP11 \\ DP12 \end{Bmatrix} \\ \begin{Bmatrix} FR21 \\ FR22 \\ FR23 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ 0 & X & X \end{bmatrix} \begin{Bmatrix} DP21 \\ DP22 \\ DP23 \end{Bmatrix} \\ \begin{Bmatrix} FR121 \\ FR122 \\ FR123 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & 0 & X \end{bmatrix} \begin{Bmatrix} DP121 \\ DP122 \\ DP123 \end{Bmatrix} \\ \begin{Bmatrix} FR1231 \\ FR1232 \end{Bmatrix} = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \begin{Bmatrix} DP1231 \\ DP1232 \end{Bmatrix} \end{gathered} \tag{13}$$

Some of the non-zero elements of design in Eq. (13) are indicated by *X*'s for simplicity, except for the first and last equations.

The first layer is an uncoupled design, but the second, third and fourth layers are decoupled designs. There is no coupling in this design, and thus it represents an acceptable design. *FR*21, *FR*22 and *FR*23 do not have to be decomposed, since they are satisfied by the *DP*s chosen for them, and therefore they are the terminal *FR*s, i.e. leaves. Similarly, *FR*11, *FR*121, *FR*122, *FR*1231 and *FR*1232 are terminal *FR*s. The decoupled design must be controlled following the sequence dictated by the design matrices. *FR*1 and *FR*2 can be simply summed up to obtain the highest-level *FR* since they are uncoupled with respect to each other. (In practice, the higher-level *FR* may automatically be satisfied when *FR*1 and *FR*2 are satisfied.) If the design matrix is a triangular matrix, the lower-level *FR*s must be satisfied in the sequence given by the design matrix to satisfy the higher-level *FR*. Each design matrix represents a junction and shows how the lower-level modules must be assembled to yield the desired *FR*.

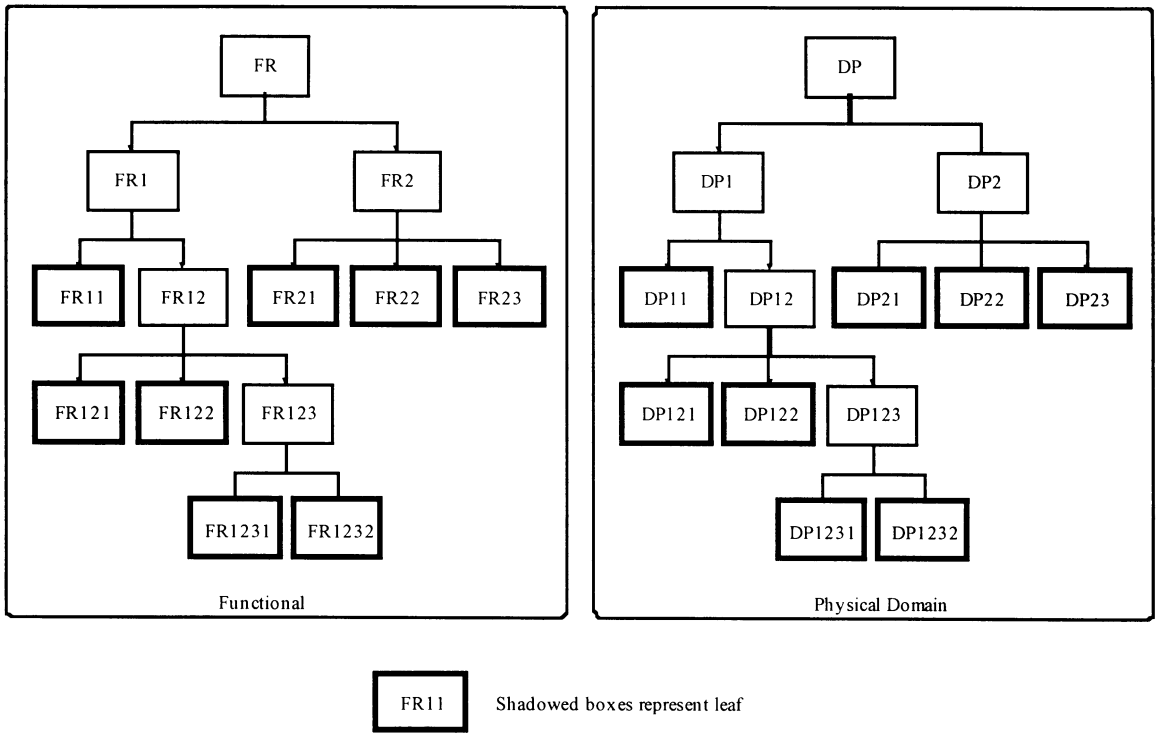

Fig. 1. The *FR* and *DP* hierarchies. The leaf is shown by boxes with thick lines. The physical embodiment of the design that should be shown at each level of decomposition is not shown here.

Each *FR* of the lowest layer of each branch (i.e. each *FR* that does not need further decomposition) is defined as the 'leaf'. The 'leaves' of the system shown in Fig. 1 are *FR*11, *FR*21, *FR*22, *FR*23, *FR*121, *FR*122, *FR*1231 and *FR*1232. The higher-level *FR*s are satisfied by combining the 'leaves' according to the information contained in the design matrix. For example, the leaves *FR*1231 and *FR*1232 yield *FR*123 if they are combined following the sequence given by the design equation, since it is decoupled design.

6.2. Modules

In the previous section, a module was defined as the row of the design matrix that yields an *FR* when it is provided with the input of its corresponding *DP*. For example, *M*123 is a module that corresponds to the elements of the design matrix and appropriate *DP*s that yield *FR*123 when it is multiplied by (or related to) *DP*123.

Combining *M*1231 and *M*1232 as per the design matrix can represent *M*123. However, *M*123 does not appear in the design explicitly, since it is not the leaf or the terminal design. Only those *DP*s corresponding to leaves need to be brought in to execute the system architecture. Lower-level modules are combined to obtain higher level modules that satisfy the higher level *FR*s. The question at this point is; 'How should the lower-level modules be combined to obtain the higher-level modules?' The design matrix at each level and at each junction provides guidance as to how the lower-level modules can be combined to obtain the higher-level modules.

At each level, each *FR* is controlled by its corresponding *DP* through a module. The module is given by the elements of each row of the design matrix. The *DP* can be a simple input. For example, *M*1231, which is also a leaf, is the module that enables the determination of *FR*1231 by 'multiplying' *M*1231 with *DP*1231 (Note: in many situations, *DP*1231 is simply an input to *M*1231.) To obtain *FR*1232, however, we have to multiply the elements of the second row of the design matrix with their respective *DP*s (since it is a decoupled design) as

$$\begin{aligned} FR1231 &= a\,DP1231 = M1231 * DP1231 \\ FR1232 &= b\,DP1231 + c\,DP1232 = M1232 * DP1232 \end{aligned} \tag{14}$$

where $M1232 = b\,(DP1231/DP1232) + c$.

To obtain the higher-level *FR*, *FR*123, the design matrix states that *FR*1231 should be determined first and then *FR*1232 because the design of *FR*123 is a decoupled design. Therefore, the lower-level modules *M*1231 and *M*1232 can be combined to obtain *FR*123 in accordances with Eq. (14).

6.3. Design Matrix and Module-Junction Diagrams

To represent the properties of junctions at each level of decomposition, a 'Module-Junction Structure Diagram' is defined [9]. The following symbols in circles denote the control of the system flow chart: **S** in the circle is for simple summation of *FR*s for an uncoupled design; **C** in the circle is for sequential control of *DP*s as suggested by the design matrix for a decoupled design; and **F** in the circle is for coupled designs, indicating that it requires feedback and violates the Independence Axiom.

In an uncoupled design, since the child *FR*s are independent of each other, their parent *FR* is satisfied by combining all the outputs of its child modules in any random sequence. The summation junction, **S**, represents this fact. When the design is decoupled, their parent *FR* is determined by combining the child modules in a given sequence indicated by the design matrix, i.e. the output of the left-hand side module is controlled first and then the right-hand side module is executed next. This is represented by the control junction, **C**. For a coupled design, which violates the Independence Axiom, the junction **F** requires that the output of the right-hand side module be fed back to the left-hand module, requiring a number of iterations until the solution converges. Many feedback junctions may never converge. When there are feedback junctions, the program will quickly become unmanageable. Figure 2 shows these control symbols and associated modules.

The module-junction structure diagram is shown in Fig. 3 for design given by Eq. (8). It should be noted that the higher-level modules are obtained by combining lower-level modules. For example, by combining modules *M*121, *M*122 and *M*123 with a control junction (**C**), we obtain the next higher-level module *M*12. Then, *M*11 and *M*12 are combined to

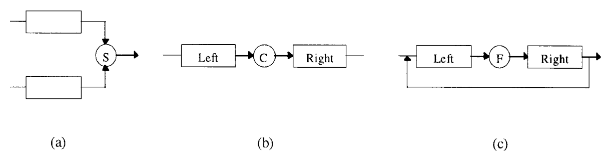

Fig. 2. Junction properties of the module-junction structure diagram. (a) Summing junction (uncoupled case), (b) control junction (decoupled case), (c) feedback junction (coupled case) [9].

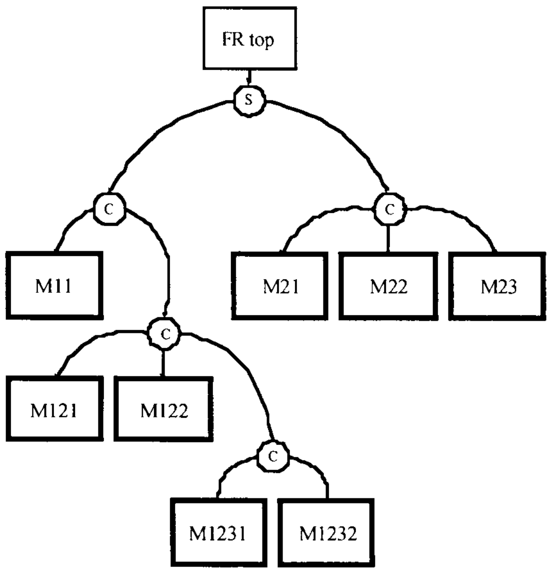

Fig. 3. The module-junction diagram for the design described by Eq. (8).

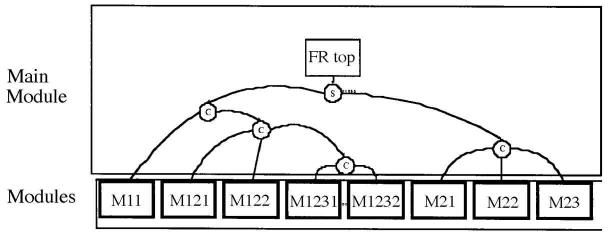

Fig. 4. This figure shows the modules and the main module for the design represented by Eq. (8). These modules, which are 'leaves', must be multiplied by their corresponding *DP*s to obtain the corresponding *FR*s.

obtain *M*1 using a control junction (**C**) again. It should be noted that the junctions define how the modules should be combined at each level of decomposition.

A main module is defined as a module which contains all the junctions of all levels (Fig. 4). A system has one main module and *n* modules corresponding to *n FR* leaves. Starting from the leaves, modules can be combined to obtain higher-level *FR*s and ultimately the highest level *FR*s in accordance with the module-junction structure diagram shown in Fig. 4. These modules corresponding to leaves must be multiplied by their corresponding *DP*s.

6.4. Flow Diagram and System Architecture

Having defined the modules for the system, the representation of the system architecture in the form of a flow diagram will be shown.

Consider a system design given by the following design equations:

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} X & O \\ O & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix}$$

$$\begin{Bmatrix} FR11 \\ FR12 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP11 \\ DP12 \end{Bmatrix}$$

$$\begin{Bmatrix} FR21 \\ FR22 \\ FR23 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP21 \\ DP22 \\ DP23 \end{Bmatrix}$$

$$\begin{Bmatrix} FR111 \\ FR112 \end{Bmatrix} = \begin{bmatrix} X & O \\ O & X \end{bmatrix} \begin{Bmatrix} DP111 \\ DP112 \end{Bmatrix} \tag{15}$$

$$\begin{Bmatrix} FR121 \\ FR122 \\ FR123 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & O & X \end{bmatrix} \begin{Bmatrix} DP121 \\ DP122 \\ DP123 \end{Bmatrix}$$

$$\begin{Bmatrix} FR211 \\ FR212 \\ FR213 \\ FR214 \end{Bmatrix} = \begin{bmatrix} X & X & O & O \\ O & X & X & O \\ O & O & X & O \\ O & O & O & X \end{bmatrix} \begin{Bmatrix} DP211 \\ DP212 \\ DP213 \\ DP214 \end{Bmatrix}$$

$$\begin{Bmatrix} FR1231 \\ FR1232 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP1231 \\ DP1232 \end{Bmatrix}$$

$$\begin{Bmatrix} FR12321 \\ FR12322 \\ FR12323 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP12321 \\ DP12322 \\ DP12323 \end{Bmatrix}$$

At the highest level of decomposition, it has two *FR*s: *FR*1 and *FR*2. The design at this level is an uncoupled design. Therefore, it can be represented, using the convention given by Fig. 2, as shown in Fig. 5.

*FR*1 has to be decomposed further to *FR*11 and *FR*12 with corresponding *DP*s and modules *M*s. Therefore, the boxes shown in Fig. 5 can be further decomposed as shown in Fig. 6. This process can continue until the complete flow diagram of the system architecture is obtained.

Figure 7 shows the final flow diagram for the system design represented by Eq. (15). It consists of modules that correspond to the leaves. It shows how the system represented by Eq. (15) should be structured in terms of its modules and operational sequence. It is constructed by following the *FR* and *DP* hierarchies, design matrices, and the module-junction diagram.

The flow chart for the system given by Eq. (15) shown in Fig. 7 consists of hardware and software. In the figure, the boxes with solid lines indicate

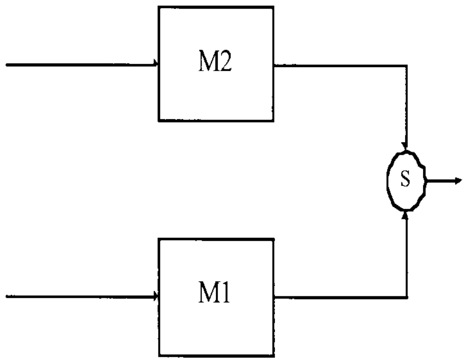

Fig. 5. This figure is the higest-level flow diagram for the system architecture given by Eq. (15).

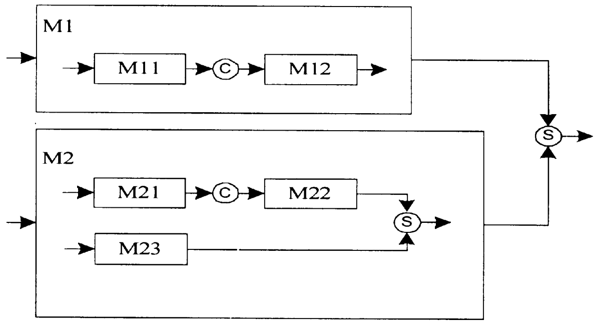

Fig. 6. This figure shows the flow chart of the system architecture given by Eq. (15) at the second level decomposition.

hardware modules. *M*122 is highlighted using a different box to indicate that this module is a software module and different from other hardware modules. This distinction can be useful to software programmers as they develop their software codes. It is important to clarify where software should be inserted in assembling the system, since in many companies, the software programmers are given a finished machine and are given the task of developing the software that can make the machine run without any system architecture. Often when the software is first developed, it is typically full of bugs because the software programmers are given the unenviable task of figuring out how the machine is expected to function without the help of the system architecture. Often they get the blame when the machine malfunctions, although the problem was caused by poor system engineering.

The design given by the flow diagram shown in Fig. 7 is an acceptable system design since the design satisfies the Independence Axiom at all levels of the design hierarchy. During the operation of the system, the lowest *DP*s are the input variables to the system that ultimately control the final output of the system.

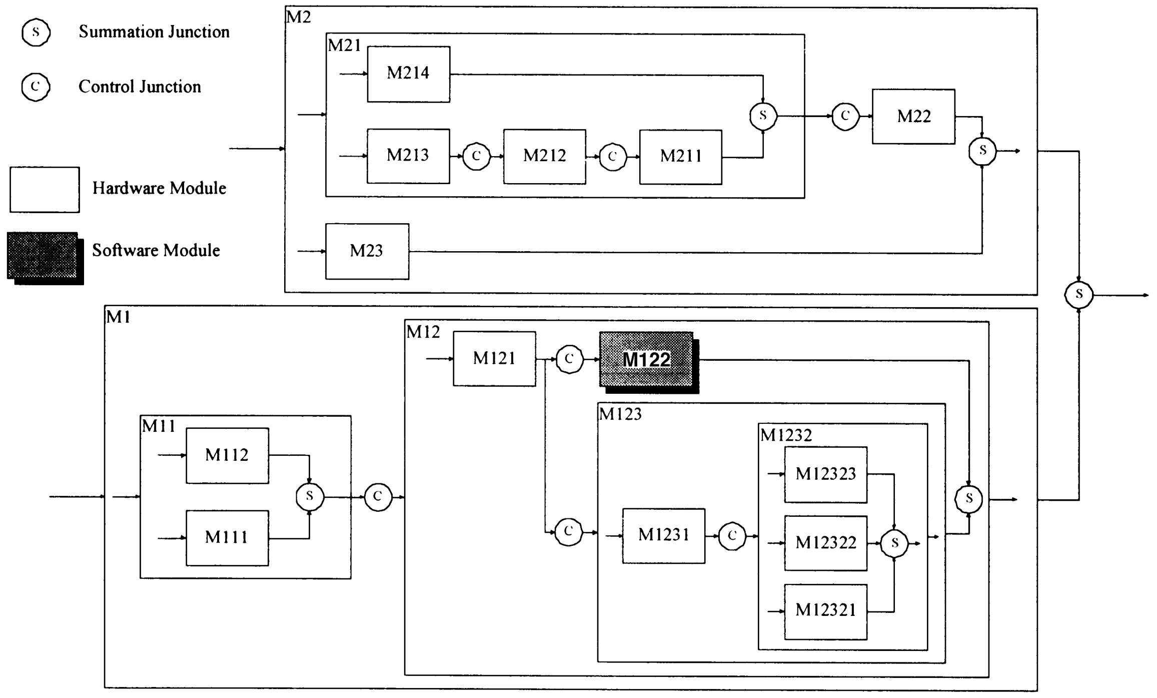

Fig. 7. The flow chart of the design of the system given by Eqs (15). This flow chart represents the system architecture of the system. This system is made up of only summation and control junctions and thus, satisfies the Independence Axiom.

At MIT, we have developed computer software that automatically generates the flow diagram of the system architecture when the hierarchies in the *FR* and *DP* domains or in the *DP* and *PV* domains are established as a result of axiomatic design of systems [5].

6.5. Modeling of Systems

One of the questions in system design is; 'How do you model a system for the purpose of simulation, optimization and identification?'

To determine the *DP*s quantitatively or to find the design window in a nonlinear design, where the elements of the design matrix are functions of *DP*s for the system given by Fig. 7 and by Eq. (15), we need to model the system. We must replace X's in Eq. (15) with mathematical expressions, based on the laws of nature, to whenever it is possible to do so. When the design satisfies the Independence Axiom, and thus the matrix is either diagonal or triangular, the values of *DP*s can be determined explicitly from the given values of *FR*s and their tolerances. In the case of linear design, optimization is not needed for a design that satisfies the Independence Axiom. In the case of nonlinear decoupled design, the best operating point for the design can be sought where the system approaches an uncoupled design as closely as possible.

6.6. System Control Command (SCC)

Once the system is designed, it can be operated using the flow diagram of the system architecture. The entire system represented by Fig. 7 may be a software system. In this case, the flow diagram shows the sequence of computation/operation of the software. In the case of a system consisting of hardware and software, the flow diagram also shows how the system must be sequenced together and what the operational sequence should be.

The goal of the systems implementation is to achieve the highest-level *FR*s. The implementation must always begin from the inner-most modules that represent the lowest-level leaves in each major branch (e.g. *M*1232*i* and *M*21*j*) and then move to the next higher level modules following the sequence indicated by the system architecture – from the inner most boxes to the outer most boxes. Therefore, the rule for execution of the system architecture, which is called the System Control Command (SCC), may be stated as follows:

1. From the input data for the lowest level *DP*s (i.e. *DP* leaves) and the corresponding elements of the design matrix, construct the lowest level modules of all branches. Multiply these modules with appropriate *DP*s (or make these modules operational by providing *DP*s as inputs) to obtain low-level *FR*s corresponding to 'leaves'.
2. Combining these low-level *FR*s based on the system architecture, obtain the next level higher *FR* and then proceed with the operation at the next higher-level modules until the highest-level module is obtained.
3. Execute this procedure simultaneously at all branches where such an action can be achieved independently so as to minimize the operational time.

Therefore, the system control module for the system architecture shown in Fig. 7 may be programmed to perform the following:

1. Establish *M*12321, *M*12322 and *M*12323 based on modeling of the given design. Based on these modules and the given input data on *DP*12321, *DP*12322 and *DP*12323, determine the next level module *M*1232.
2. Similarly, execute the operation indicated by *M*11 and *M*21.
3. Then, execute the operation indicated by *M*123 and *M*2.
4. Next, execute the operation indicated by *M*12.
5. Next, execute the operation indicated by *M*1.
6. Combine *M*1 and *M*2 as indicated by the system architecture to obtain the final desired output.

6.7. Application of the Flow Diagram of the System Architecture

The system architecture represented by the flow diagram can be used in many different applications and for many different purposes. The following are some of these applications which will be further elaborated in other publications:

1. *Diagnosis* – The impending failure of a complex system can be diagnosed based on the system architecture. The flow diagram will reduce the service cost of maintaining machines and systems.
2. *Engineering change orders* – during the development process, it is inevitable that changes will need to be made to the decisions made in the past [10]. The system architecture and the flow diagram can be used to identify all the related modules that must also be changed when a single change in the system architecture is proposed.
3. *Job assignment and management* – the flow diagram can be used to organize development projects and to assign tasks to various participants, since it defines the inter-relationships between various modules. This will aid in project execution and management.
4. *Distributed systems* – when a large development project is executed, the lower-level tasks are performed by many located at distance places, sometimes involving sub-contractors and sub-sub-contractors. The flow diagram of the systems architecture will provide the road map by which the tasks can be coordinated.
5. *Software design through assembly of modules* – ultimately, software will be designed by assembling existing software modules in a database. The systems architecture will provide a guideline for software development.

7. Design and System Architecture of Large Flexible Systems

Large flexible systems are defined as systems whose functional requirements are time variant [1]. The design process for these systems is similar to the design of fixed systems, except that the system may have to satisfy different subsets of functional requirements with the set of *FR*s changing as a function of time. When the set of *FR*s changes, the corresponding *DP* set must be selected anew to satisfy the Independence Axiom. Although some of the *FR*s in the subsets of {*FR*s} may be common, the set of *DP*s must be considered as a unit because of the need to satisfy the Independence Axiom. Several theorems pertaining to the design of these large flexible systems are given (see Theorems 19 through 24, Appendix B).

In the system architecture for a large flexible system, new {*DP*s} must be chosen each time that a new set of {*FR*s} is to be satisfied by the large flexible system. This is the case, even though some of the *FR*s in the new set {*FR*s} might have appeared in the preceding set of {*FR*s}, so as to maintain the independence of functional requirements for the new set of {*FR*s}.

When the functional requirements can be satisfied by the chosen *DP*s without further decomposition, then the creation of a system architecture for a large flexible system is straightforward. For example, in the case of the large flexible system whose functional requirements are given as

$$\begin{array}{lll} @\ t = 0, & \text{the subsets are} & \\ & \{FR\text{s}\}_0 = \{FR1, FR3, FR7, FR8\} & \\ @\ t = T_1, & \{FR\text{s}\}_1 = \{FR3, FR5, FR6, FR9\} & (16) \\ @\ t = T_2, & \{FR\text{s}\}_2 = \{FR3, FR4, FR8, FR9\} & \end{array}$$

From the database, we may find the following solution:

$$@\ t = 0,\quad \begin{Bmatrix} FR1 \\ FR3 \\ FR7 \\ FR8 \end{Bmatrix} = \begin{bmatrix} X & O & O & O \\ X & X & O & O \\ O & X & X & O \\ O & O & O & X \end{bmatrix} \begin{Bmatrix} DP1^b \\ DP3^f \\ DP7^e \\ DP8^g \end{Bmatrix} \qquad (17)$$

This is the final solution since no *FR* needs to be decomposed further. It should be noted that $DP7^e$ and $DP8^g$ affect only *FR*7 and *FR*8, respectively. However, $DP3^f$ affects *FR3* and *FR*7, and $DP1^b$ affects *FR1* and *FR*3. Therefore, the set *FR1*, *FR*3 and *FR*7 constitutes a decoupled design, and *FR*8 can be treated as an uncoupled design with respect to this set.

The system must reconfigure at $t = T_1$, because the system must satisfy a new set of functional requirements. Although, *FR*3 is also a part of this set, $DP3^f$ may not necessarily be an appropriate choice if the selection of $DP3^f$ violates the Independence Axiom. Therefore, whenever a new set of *FR*s must be satisfied, it is better to start the design process from scratch rather than to try to modify an existing design solution. This is stated as Theorem 5.

When the decomposition of the *FR*s shown in Eq. (17) is necessary to complete the system design at a given instant, the designers have to zigzag between the domains and develop the design hierarchy each time the *FR* set changes. Based on the design solution, the system must be reconfigured when a new set of *FR*s must be satisfied at a later time.

The design process becomes much more complicated when decomposition is necessary, since a particular choice of the highest level may not yield satisfactory lower-level *FR*s and *DP*s for a variety of reasons. For instance, it may not be possible to find a suitable set of lower-level *DP*s that can satisfy the independence of the corresponding *FR*s at that level. In some cases, the information content required to make the system function properly may be too large. In these situations, the higher-level design should be reviewed to select new high-level *DP*s until a satisfactory design solution is found through zigzagging and decomposition.

8. Other Issues of System Design

8.1. Reusability

In constructing systems, the reusability of the existing *FR/DP* and *DP/PV* relationships is an important issue to utilize the known solutions without having to reconfigure the system each time. This need exists both in the hardware and the software world.

In the case of hardware, the solution is more obvious. For example, if *FR* is ‘create a linear motion’, we can identify several known design solutions as *DP*s, e.g. linear motor, pneumatic piston, ball screw, etc. These possible *DP*s constitute a hardware database.

In the case of software, the reusability is in the form of a knowledge database [7]. Given an *FR*, we have to find a *DP* that can satisfy the *FR* from a database (which may be recalled from a library of databases or knowledge-bases) such as

$$FR\ \$\ f[DPa, DPb, \ldots\ldots.. DPy] \qquad (18)$$

Equation (18) states that *FR* can be satisfied (indicated by $) by any one of the *DP*s listed. Such a database can be created for a large number of frequently used *FR*s, making it possible to reuse the database. It constitutes a software library.

When there is more than one *FR*, we have to select a *DP* set that satisfies the independence of the entire set of *FR*s by finding a right set of *DP*s. For frequently used sets of multiple *FR* design problems, a database or a library of software modules may be created, which relates {*FR*s} to {*DP*s} as follows:

$$\{FR\text{s}\} = f[\{DP\text{s}\}^a, \{DP\text{s}\}^b, \ldots\ldots.. \{DP\text{s}\}^z] \qquad (19)$$

The knowledge base or the software module represented by Eqs (18) and (19) is reusable when a design solution for the identical of *FR*s set is needed.

8.2. Controllability and Stability of Systems

How good is a system that has been created using design axioms and the associated design methodologies? Is it the best solution? How do we know that it is a correct solution? To answer these questions, explanation is necessary to put them into their proper context.

In natural sciences, the validity of axioms (*a la* Newton’s laws, thermodynamic laws, Euclid’s geometry) are checked by comparing theoretical predictions made based on the axioms with the physical measurements made of natural phenomena. However, since design axioms are not part of natural science, their validity must be checked by different means. This can be done in several different ways. For example, we may determine how good designs satisfy the Independence Axiom and how much better the functional requirements are satisfied by an uncoupled or decoupled design in comparison to coupled designs. We may also show how a new creative design can be generated quickly without much trial-and-error when these axioms are used, and how a robust design can be created based on the axioms. We may also show how the design that requires the lowest information content is better than the designs with high information contents. Of course, the ultimate check is to look for exceptions or counterexamples to the design axioms.

Design axioms were found to improve all designs without exceptions or counter-examples. Everything researchers and engineers all over the world have done with these axioms has yielded powerful or improved designs in many different areas. Poorly designed machines were improved significantly when the design axioms were applied. Also, new inventions have been made and many diverse problems have been solved simply and elegantly using the axioms. To date, the author has not seen counterexamples and exceptions to the design axioms. When counter-examples or exceptions are proposed, the author always found flaws in the arguments. The designs that satisfy the Independence Axiom perform and fulfill functional requrements better than those which violate the axiom. Moreover, the Information Axiom does lead to robust and more reliable designs.

Nevertheless, the basic question is; ‘Why do these axioms improve or generate better designs?’ The answer is that the axioms guarantee the controllability and stability of the design – two requirements of a system [11]. This is the link between axioms in natural science such as the laws of thermodynamics and the design axioms.

Controllability means that the *FR*s can be satisfied within the specified tolerances. In axiomatic design, this is done by first selecting proper *DP*s and then by changing *DP*s that can satisfy the *FR*s. The optimality of the solution is checked by eliminating the bias and reducing the variance to satisfy the Information Axiom, which can be done with the Independence Axiom is satisfied by the design [4]. Controllability means that the desired operating point can be achieved and that *FR*s can be satisfied in a desired range of the design space.

Moreover, uncoupled or decoupled designs – the designs that satisfy the Independence Axiom – are stable. Stability means that the design performs consistently and reliably even when subjected to external disturbances and noise. This is achieved by selecting a set of {*DP*s} that can be used to change {*FR*s} over the range specified in a stable manner. A stable system is one that does not change its state unless a finite amount of external energy is applied to the system. In other words, changing the state of a stable system requires a finite amount of energy input. A design that satisfies the Independence Axiom is inherently stable, because the designer has chosen a *DP* to contorl a given *FR* within the specified tolerance without affecting other *FR*s. The designer would not have chosen a *DP* that is unstable or that makes an *FR* unstable, since such a design cannot satisfy the functional requirements within the specified tolerance. Other potential disturbances such as those due to noise and the variations of other *FR*s are checked so as not to affect the stability of a given *FR* because of its independence from other *FR*s.

Unstable systems are difficult if not impossible to control, since an *FR* of an unstable system can change its state on its own without any external inputs through changes in *DP*s. The design that violates the Independence Axiom (i.e. coupled design) has a high probability of not satisfying the functional require-ments within the specified tolerance, since in such a coupled design the *FR*s cannot be ‘controlled’ independently from other *FR*s. Furthermore, a coupled design is unstable in the sense that unintended *FR*s will change arbitrarily when changes are made to control a specific *FR*.

If a design is coupled, functional requirements can only be satisfied when there exists a unique solution, which requires that all *DP*s change in unison along a pre-prescribed path. This is difficult if not impossible to achieve when there are many *FR*s and many layers of design hierarchies in a system. Furthermore, if a fully coupled design is acceptable, it is equivalent to having only one independent *FR*, since when a *FR* of the set changes, all other *FR*s change in concert. It should be noted again that the functional requirements are defined as the minimum set of independent requirements.

A stable system is at an ‘equilibrium’ state at all times in the sense that the *FR*s remain at a prescribed state unless changes in the *FR*s are introduced by changing corresponding *DP*s. Hence, the design axioms are consistent with thermodynamic laws, but are different from them because while the design axioms also deal with controllability, thermodynamic laws do not.

By demanding that human designed artifacts be controllable and stable, the Independence and the

Information Axioms capitalize on the natural tendency and ability of human beings to control stable systems. Furthermore, the Information Axiom demands that an artifact that satisfies the Independence Axiom must be such that it is simple to make and use. Since the information content is expressed in terms of probability, the Information Axiom defines the degree of random variations of *FR*s or the deviation from the ideal system.

Why is a stable system a better design? A stable design that satisfies the design axioms has the following desirable characteristics:

1. Predictable behavior in a finite time scale.
2. Clear input/output relationships.
3. Compatibility with human capabilities.
4. Ability to tolerate variations among the *FR*s, *DP*s and modules due to clear hierarchical features.

8.3. System Design as a Chaotic Process

In dynamics, the notion of *chaos* has had a major impact in describing behavior of many natural systems. 'Chaos' is a phenomenon in which 'a deterministic system exhibits a periodic behavior that depends sensitively on the initial conditions, thereby rendering long term predictions impossible' [12]. Design has some of these chaotic characteristics.

Although the formulation of design equations appears substantially different from the set of differential equations that govern dynamic systems, the final outcome of the design also depends on initial mapping conditions. The final outcome of design activity is unpredictable and it depends on the initial mapping from *FR*s to *DP*s. Furthermore, the mapping at any level can yield entirely different results in the physical domain, even though the designs satisfy the same set of *FR*s at the highest-level of the design hierarchy. In this sense, design may also be called a chaotic process, notwithstanding the fact that axiomatic design provides a rigorous decision-making framework that can lead to rational design that is simple.

Two or more systems that perform an exactly identical set of functions may be entirely different in the physical domain because of the 'chaotic' nature of the design process. What distinguishes the physically different designs of these systems is not the physical embodiment itself, but the information required to fulfill the functions in the functional domain. This is the essence of the Information Axiom – even in two designs that identically satisfy the highest level {*FR*s}, one of the two systems may be superior to the other because it has less information content. If all the system designs satisfy the same set of *FR*s, constraints, and the Independence Axiom, they are 'equivalent' – equally good designs – if the information content is the same, although physically they may be entirely different.

The following definition of 'equivalent' and 'identical' design may be adopted:

Definition S1. Two designs are defined to be *equivalent* if they perform the same set of the highest level *FR*s within the bounds established by the same set of constraints, even though the mapping and decomposition process might have yielded designs that have substantially different lower level *FR*s and all *DP*s for each of these designs.

Definition S2. Designs that fulfill the same set of the highest level *FR*s and satisfy the Independence Axiom with zero information content are defined to be *identical* if their lower level *FR*s and all *DP*s are also the same.

Based on the foregoing reasoning and definitioins, the following theorems can be stated:

Theorem S1. The decomposition process does not affect the overall performance of the design if the highest level *FR*s and *C*s are satisfied and if the information content is zero, irrespective of the specific decomposition process.

Theorem S2. Two 'equivalent' designs can have a substantially different cost structure, although they perform the same set of functions and they may even have the same information content.

8.4. On Human-Machine Interface

Human-machine interface has been studied in relation to automobiles, nuclear reactors, aeronautics, astronautics, and other fields [13]. Previous studies, which have dealt with interaction between human operators and specific machines, have modeled the interactions, sometimes analytically and sometimes based on the results of simulations. Based on these models they have attempted to create ideal human/machine interfaces.

From the axiomatic design point of view, human/machine interactions can occur at all levels of a design hierarchy. In the case of modern aircraft, the human/machine interface is at the highest level – between the pilot and the aircraft cockpit. The rest of the aircraft system is made up of hardware and software.

In the case of large factories, the human-machine interactions occur at all levels – operators running machines who may interface with Automatic Guided Vehicles (AGVs), which in turn interface with many other hardware or software modules. In such a system, the human/machine interface can be imbedded at all levels of the hierarchy. Therefore, in these systems, the system functions depend upon hardware, machine-to-machine level interactions, and human-machine interactions at many different levels.

Large systems involving many human-machine interfaces have both positive and negative factors. The positive aspects are related to the fact that human intelligence is a powerful module that can easily relate the *DP*s to *FR*s without the need to model the system and any dependence on software. Human operators can be trained to generate operational knowledge from experience. Human operators are also flexible and can create their individualized databases. Sometimes, they provide both intelligence and motive power.

On the other hand, a major shortcoming of the human/machine interface is that the human operator is not precise and sometimes, may not be dependable. The probability of human operators making errors is greater than well-proven hardware/software systems. Therefore, the system must be able to detect and compensate for errors of human operators when the consequence of system failure cannot be tolerated.

The system architecture of organizations can have many modules that are performed by human operators at multiple levels. In the extreme case, the entire system can be made of modules, all of which are performed by human operators, e.g. government agencies or consulting companies. Many management organizations are systems with only human modules. One of the virtues as well as difficulties of these 'modules' is that human beings have their own intelligence and can adjust the characteristics of the modules, the *FR*s and *DP*s, which may wreck havoc with the overall system operation unless guidelines and rules are firmly established.

9. Case Studies

The system design theory presented in this paper has been applied to manufacturing systems [4], software systems [5,6], products [16], complex equipment design [14,15] and organizational design [4]. These are substantial studies that cannot be incorporated in this paper because of limited space. These will be presented in detail in the forthcoming book [4].

System architecture is created during these design processes. It can be in the form of the *FR/DP/PV* hierarchies or flow diagrams. However, it may also be used to be sure that there is no coupling in an existing design and to create a complete documentation for its hardware and software design.

10. Conclusions

This paper presents a theoretical basis for design of systems. Systems should be classified as large or small according to the number of functional requirements. Small, fixed systems are those that have a limited number of functional requirements that do not change as a function of time. Large systems are those with a large number of functional requirements at the highest level. Large systems are further sub-divided into large, fixed systems and large, flexible systems. Large, flexible systems are those whose functional requirements change as a function of time.

Large, flexible systems require constant reconfigurations of the system by choosing new sets of {*DP*s} during the operation. Even when a large number of *FR*s in the *FR* set do not change and only a few of the *FR*s change, the {*DP*s} sets must change to satisfy the functional independence. Therefore, the design and operation of large, flexible systems requires a large database (or knowledge base), from which appropriate *DP*s can be selected to satisfy a given set of *FR*s at a given instant. The Independence Axiom provides the criterion for acceptability of chosen *DP*s each time the {*FR*s} sets change in a large, flexible system.

The flow diagram for system architecture is presented. The flow diagram presents how a system is configured in terms of *FR*s, *DP*s and design matrices (*DM*). This is done using 'modules' derived from the row of design matrices and design parameters. The system architecture and the associated flow diagram are made up of modules that characterize the system. The flow diagram of a system captures the essence of the system, including the control of the system for its operation. The flow diagram and the system architecture may be used as the road map in design, development, service, and failure diagnosis of systems. In addition, it provides a useful documentation of the system design. By examining the flow diagram, we can point out weak design links by being able to spot the coupled parts of the systems design.

Acknowledgements

The author would like to thank Dr. Sung-Hee Do, Jason Hintersteiner, Tae-Sik Lee, Professor Gyung-Jin Park and Derrick Tate for reviewing this paper and for their comments.

References

1. Suh, N. P. Design and operation of large systems. Journal of Manufacturing Systems 1995;14(3):203–213.
2. Rechtin, E. Systems Architecting: Creating and Building Complex Systems. Prentice-Hall, New York; 1991.
3. Suh, N. P. Design of systems. Annals of CIRP 1997;46(1):75.
4. Suh, N. P. Axiomatic Design: Advances and Applications, Oxford University Press, Oxford, 1998 (in preparation).
5. Do, S.-H., Suh, N. P. Axiomatic Design Software, unpublished copyrighted software, 1998.
6. Park, G.-J., Do, S.-H., Suh, N. P., Design and Extension of Software Using the Axiomatic Design Framework for the Design Software for TV Glass Bulbs. To be submitted, 1999.
7. Suh, N. P. The Principles of Design, Oxford University Press, Oxford, 1990.
8. Suh, N. P. Designing-in of quality through axiomatic design. IEEE Transactions on Reliability 1995;44(2):256–264.
9. Kim, S. J., Suh, N. P., Kim, S.-K. Design of software systems based on axiomatic design. Annals of the CIRP 1991;40(1):165–170. (Also in Robotics & Computer-Integrated Manufacturing 1992;3:149–162).
10. Nordlund, M. An information framework for engineering design based on axiomatic design. Doctoral Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 1996.
11. Suh, N. P. Is there any relationship between design science and natural science? Ho-Am Prize Memorial Lecture, Seoul National University. Published by the Ho-Am Prize Committee, Seoul, Korea, 1997.
12. Strogatz, S. H. Nonlinear Dynamics and Chaos, Addison-Wesley, Reading, MA, 1994.
13. Sheridan, T. Mann-Machine Systems: Information, Control, and Decision Models of Human Behavior, MIT Press, Cambridge, MA, 1974.
14. Hintersteiner, J., Ph.D. Thesis, Department of Mechanical Engineering, MIT. (In progress).
15. Lee, T.-S., Masters Thesis, Department of Mechanical Engineering, MIT 1998.
16. Suh, N. P. Axiomatic design of mechanical systems. Special 50th Anniversary Combined Issue of the Journal of Mechanical Design and the Journal of Vibration and Acoustics, Transactions of the ASME 1995;17:1–10.

Appendix A: Introduction to Axiomatic Design [7]

The Concept of Domains

Design involves an interplay between ‘what we want to achieve’ and ‘how we choose to satisfy the need (i.e. the what)’. The world of design is made up of four *domains*: the *customer domain*, the *functional domain*, the *physical domain* and the *process domain*. The domain structure is illustrated schematically in Fig. A1. The domain on the left relative to the domain on the right represents ‘what we want to achieve’, whereas the domain on the right represents the design solution for ‘how we propose to satisfy the requirements specified in the left domain’.

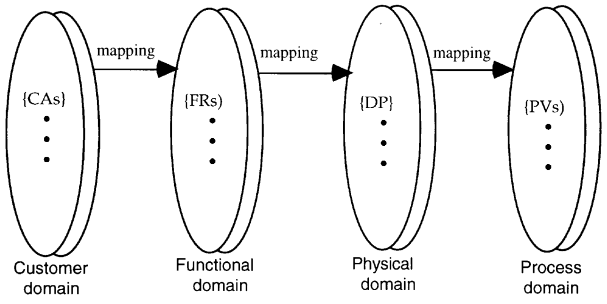

Fig. A1. Four domains of the design world. {x} are characteristic vectors of each domain.

The customer domain is characterized by customer needs or the attributes the customer is looking for in a product or process or systems or materials. In the functional domain, the customer needs are specified in terms of Functional Requirements (*FR*s) and constraints (*C*s). To satisfy the specifed *FR*s, we conceive design parameters, *DP*s, in the physical domain. Finally, to produce the product specified in terms of *DP*s, in the process domain we develop a process that is characterized by process variables, *PV*s. Many seemingly different design tasks in many different fields can be described in terms of the four design domains, including products, organizations, systems, materials and software.

All designs fit into these four domains. Therefore, all design activities, be they product design or software design, can be generalized in terms of the same principles. Because of this logical structure of the design world, the generalized design principles can be applied to all design applications and we may consider all the design issues that arise in the four domains systematically and if necessary, concurrently.

Definitions

It is important to remember the definition of a few key words used in axiomatic design, as axioms are valid only within the bounds established by the definitions of these key terms. The definitions are as follows:

- *Axiom*: a self-evident truth or fundamental truth for which there are no counter examples or exceptions. axioms may not be derived from other laws of nature or principles.
- *Corollary*: an inference derived from axioms or propositions that follow from axioms or other propositions that have been proven.
- *Functional Requirement*: a minimum set of independent requirements that completely characterizes the functional needs of the product (or software, organizations, systems, etc.) in the functional domain. By definition, each *FR* is independent of every other one at the time the *FR*s are established.
- *Constraint*: constraints are bounds on acceptable solutions. There are two kinds of constraints: input constraints and system constraints. Input constraints are imposed as part of design specifications. System constraints are constraints imposed by the system in which the design solution must function.
- *Design Parameter*: design parameters are the key physical (or other equivalent terms in the case of software design, etc.) variables in the physical domain that characterize the design satisfying the specified *FR*s.
- *Process Variable*: process variables are the key variables (or other equivalent term in the case of software design, etc.) in the process domain that characterize the process which can generate the specified *DP*s.

The First Axiom: The Independence Axiom

During the mapping process going from the functional domain to the physical domain, we must make correct design decisions using the Independence Axiom. When several designs that satisfy the Independence Axiom are available, the Information Axiom can be used to select the best design.

The basic postulate of the axiomatic approach to design is that there are fundamental axioms that govern the design process. Two axioms were identifed by examining the common elements that are always present in good designs, be it a product, process, or systems design. They were also identified by examining actions taken during the design stage that resulted in dramatic improvements. The history of how the design axioms were created is given by Suh [7].

The first axiom is called the Independence Axiom. It states that the independence of Functional Requirements (*FR*s) must always be maintained, where *FR*s are defined as the minimum number of independent functional requirements that characterize the design goals. The second axiom is called the Information Axiom, and it states that among those designs that satisfy the Independence Axiom, the design that has the smallest information content is the best design. Because the information content is defined in terms of probability, the second axiom also states that the design that has the highest probability of success is the best design. Based on these design axioms, we can derive theorems and corollaries. The axioms are formally stated as

- *Axiom 1*: *The Independence Axiom*
  Maintain the independence of the Functional Requirements (*FR*s)
- *Axiom 2*: *The Information Axiom*
  Minimize the information content of the design.

As stated earlier, the functional requirements are defined as the minimum set of independent requirements that the design must satisfy. A set of functional requirements {*FR*s} is the description of design goals. The Independence Axiom states that when there are two or more functional requirements, the design solution must be such that each one of the functional requirements can be satisfied without affecting the other functional requirement. That means we have to choose a correct set of *DP*s (design parameters) to be able to satisfy the functional requirements and maintain their independence.

The Independence Axiom is often misunderstood. Many people confuse functional independence with physical independence. The Independence Axiom requires that the functions of the design be independent from each other, not the physical parts. For example, a beverage can has about 12 *FR*s, and yet it is made up of only three physical pieces. The *DP*s are imbedded in the can shape. The second axiom would suggest that physical integration is desirable to reduce the information content, if the functional independence can be maintained.

After the *FR*s are established, the next step in the design process is the conceptualization process, which occurs during the mapping process going from the functional domain to the physical domain.

To go from ‘what’ to ‘how’ requires mapping which itself involves creative conceptual work. Once the overall design concept is generated by mapping, we must identify the Design Parameters (*DP*s) and complete the mapping process. During this process, we must think of all the different ways of fulfilling each of the *FR*s by identifying plausible *DP*s.

Sometimes it is convenient to think about a specific *DP* to satisfy a specific *FR*, repeating the process until the design is completed.

The mapping process between the domains can be expressed mathematically in terms of the characteristic vectors that define the design goals and design solutions. At a given level of design hierarchy, the set of functional requirements that define the specific deseign goals constitutes a vector {*FR*s} in the functional domain. Similarly, the set of design parameters in the physical domain that are the 'how's' for the *FR*s also constitutes a vector {*DP*s}. The relationship between these two vectors can be written as

$$\{FRs\} = [A]\,\{DPs\} \qquad (A1)$$

where [*A*] is a matrix defined as the design matrix that characterizes the product design. Equation (A1) may be written in terms of its elements as $FR_i = A_{ij}\ DP_j$. Equation (A1) is a design equation for the design of a product.

For a linear design, A_{ij} are constants, whereas for nonlinear design, A_{ij} are functions of *DP*s. There are two special cases of the design matrix: the diagonal matrix where all A_{ij}'s except those $i = j$ are equal to zero, and the triangular matrix where either upper or lower triangular elements are equal to zero.

For the design of processes involving mapping from the physical domain to the process domain, the design equation may be written as

$$\{DPs\} = [B]\,\{PVs\} \qquad (A2)$$

[*B*] is the design matrix that defines the characteristics of the process design and is similar in form to [*A*].

To satisfy the Independence Axiom, the design matrix must be either diagonal or triangular. When the design matrix [*A*] is diagonal, each of the *FR*s can be satisfied independently by means of one *DP*. Such a design is called an *uncoupled* design. When the matrix is triangular, the independence of *FR*s can be guaranteed if and only if the *DP*s are changed in a proper sequence. Such a design is called a *decoupled* design. Therefore, when several functional requirements must be satisfied, we must develop designs that will enable us to create a diagonal or triangular design matrix.

The design matrices [*A*] and [*B*] can be made of matrix elements which are constants or functions. If the matrix is made of constants, it represents a linear design. If it is a relational matrix where its elements are functions of *DP*s, the design matrix may represent a nonlinear design.

The design goals are often subject to constraints, *C*s. Constraints provide the bounds on the acceptable design solutions and differ from the *FR*s in that they do not have to be independent. Some constraints are specified by the designer. Many constraints are imposed by the environment within which the design must function. These are system constraints.

All functional requirements are not at the same level. There are hierarchies in *FR*, *DP* and *PV* domains, which are created through decomposition. However, contrary to the conventional wisdom on decomposition, they cannot be decomposed by remaining in one domain. One must zigzag between the domains to be able to decompose the *FR*s, *DP*s and *PV*s. Through this zigzagging we create hierarchies for *FR*s, *DP*s and *PV*s in each design domain. Through this decomposition process, we establish a set of hierarchies of *FR*, *DP* and *PV*, which is a representation of the architecture of the design.

According to the Independence Axiom, a superior design between an uncoupled and coupled design is the uncoupled one, and therefore there must always exist an uncoupled design that has a smaller information content than a coupled design.

In the preceding three examples, the design matrix was formulated in terms of X and 0. In some cases, it may be sufficient to complete the design with simply X's and 0's. In many cases, we may take further steps to optimize the design. After the conceptual design is done in terms of X and 0, we need to model the design more precisely to optimize the design. Through modeling, we can replace X's with constants in the case of a linear design or functions that involve *DP*s.

The Second Axiom: The Information Axiom

There can be many designs which are equally acceptable from the functional point of view. However, one of these designs may be superior to others in terms of probability of success in achieving the design goals as expressed by the functional requirements. The Information Axiom states that the one with the highest probability of success is the best design. Specifically, the Information Axiom may be stated as:

- *Axiom 2*: *The Information Axiom*
  Minimize the information content.

Information content I is defined in terms of the probability of satisfying a given *FR*. If the probability of success of satisfying a given *FR* is *p*, the information *I* associated with the probability is defended as

$$I = -\log_2 p \qquad (A3)$$

The information is given in units of bits. The logarithmic function is chosen so that the information content will be additive when there are many functional requirements that must be satisfied at the same time.

In the general case of *n FR*, for an uncoupled design, *I* may be expressed as

$$I = \sum_{i=1}^{n} [\log l/ p_i] \qquad (A4)$$

where *pi* is the probability of *DPi* satisfying *FRi*, and log is either the logarithm based on 2 (with the unit of bits) or the natural logarthim (with the unit of nats). Since there are *n FR*s, the total information content is the sum of all these probabilities. The Information Axiom states that the design that has the smallest *I* is the best design, since it requires the least amount of information to achieve the design goals. When all probabilities are equal to one, the information content is zero, and conversely, the information required is infinite when one or more probabilities are equal to zero. That is, if probability is low, we must supply more information to satisfy the functional requirements.

In the real world, the probability of success is governed by the intersection of the tolerance defined by the designer to satisfy the *FR*s and the tolerance (or the ability) of the system to produce the part within the specified tolerance.

The probability of success can be computed by specifying the *Design Range* (*dr*) for the *FR* and by determining the *System Range* (*sr*) that the proposed design can provide to satisfy the *FR*. Figure A2 illustrates these two ranges graphically. The vertical axis (the ordinate) is for the probability density and the horizontal axis (the abscissa) is for either *FR* or *DP*, depending on the mapping domains involved. When the mapping is between the functional domain and the physical domain as in product design, the abscissa is for *FR*, whereas for the mapping between the physical domain and the process domain as in process design, the abscissa is for *DP*. In Fig. A2, the system range is plotted as a probability density versus the specified *FR*. The overlap between the design range and system range is called the *common range* (*cr*), and this is the only region where the functional requirements are satisfied. Consequently, the area under the common range divided by the area under the system range is equal to the design's probability of success of achieving the specified goal. Then, the information content may be expressed as [7]

$$I = \log\,(A_{sr}/A_{cr}) \qquad (A5)$$

where A_{sr} denotes the area under the system range and A_{cr} is the area of the common range. Furthermore, since A_{sr} = 1.0 in most cases and there are *n FRs* to satisfy, the information content may be expressed as

$$I = \Sigma\,\log\,(1/A_{cr})_i \qquad (A6)$$

Often, design decisions must be made when there are many *FR*s that must be satisfied at the same time. The Information Axiom provides a powerful criterion for making such decisions without the use of arbitrary weighting factors used in other decision-making theories.

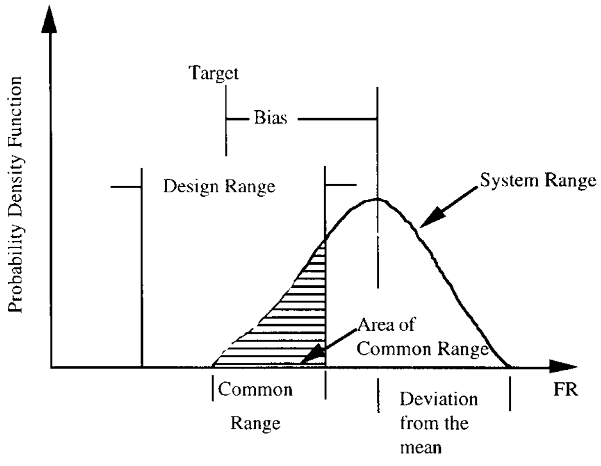

Fig. A2. Design range, system range and common range in a plot of the probability density function (pdf) of a functional requirement. The deviation from the mean is equal to the square root of the variance.

Appendix B: Corollaries and Theorems

Corollaries (from Strogatz [12])

Corollary 1 (Decoupling of Coupled Designs) Decouple or separate parts or aspects of a solution if *FR*s are coupled or become interdependent in the designs proposed.

Corollary 2 (Minimization of *FR*s) Minimize the number of *FR*s and constraints.

Corollary 3 (Integration of Physical Parts) Integrate design features in a single physical part if *FR*s can be independently satisfied in the proposed solution.

Corollary 4 (Use of Standardization) Use standardized or interchangeable parts if the use of these parts is consistent with *FR*s and constraints.

Corollary 5 (Use of Symmetry) Use symmetrical shapes and/or components if they are consistent with the *FR*s and constraints.

Corollary 6 (Largest Tolerance) Specify the largest allowable tolerance in stating *FR*s.

Corollary 7 (Uncoupled Design with Less Information) Seek an uncoupled design that requires less information than coupled designs in satisfying a set of *FR*s.

Corollary 8 (Effective Reangularity of a Scalar) The effective reangularity *R* for a scalar coupling 'matrix' or element is unity.

Theorems of General Design (Mostly from Suh [7])

Theorem 1 (Coupling Due to Insufficient Number of *DP*s) When the number of *DP*s is less than the number of *FR*s, either a coupled design results, or the *FR*s cannot be satisfied.

Theorem 2 (Decoupling of Coupled Design) When a design is coupled due to the greater number of *FR*s than *DP*s (i.e. $m. > n$), it may be decoupled by the addition of new *DP*s so as to make the number of *FR*s and *DP*s equal to each other and a triangular matrix.

Theorem 3 (Redundant Design) When there are more *DP*s than *FR*s, the design is either a redundant design or a coupled design.

Theorem 4 (Ideal Design) In an ideal design, the number of *DP*s is equal to the number of *FR*s.

Theorem 5 (Need for New Design) When a given set of *FR*s is changed by the addition of a new *FR*, or substitution of one of the *FR*s with a new one, or by selection of a completely different set of *FR*s, the design solution given by the original *DP*s cannot satisfy the new set of *FR*s. Consequently, a new design solution must be sought.

Theorem 6 (Path Independence of Uncoupled Design) The information content of an uncoupled design is independent of the sequence by which the *DP*s are changed to satisfy the given set of *FRS*.

Theorem 7 (Path Dependency of Coupled and Decoupled Design) The information contents of coupled and decoupled designs depend on the sequence by which the *DP*s are changed to satisfy the given set of *FR*s.

Theorem 8 (Independence and Tolerance) A design is an uncoupled design when the designer-specified tolerance is greater than

$$\left(\sum_{\substack{j \neq i \\ i=1}}^{n} (\partial FR_i / \partial DP_j)\ \Delta DP_j \right)$$

in which case the nondiagonal elements of the design matrix can be neglected from design consideration.

Theorem 9 (Design for Manufacturablity) For a product to be manufacturable within the specified tolerances, the design matrix for the product, [**A**] (which relates the **FR** vector for the product to the **DP** vector of the product) times the design matrix for the manufacturing process, [**B**] (which relates the **DP** vector to the **PV** vector of the manufacturing process) must yield either a diagonal or triangular matrix. Consequently, when any one of these design matrices, i.e. either [**A**] or [**B**], represents a coupled design, the product cannot be manufactured readily within the specified tolerances, because small variations in the manufacturing process result in large variations in *FR*s and *DP*s. When they are triangular matrices, both of them must be either upper triangular or lower triangular for the manufacturing process to satisfy the independence for functional requirements.

Theorem 10 (Modularity of Independence Measures) Suppose that a design matrix [**DM**] can be partitioned into square submatrices that are nonzero only along the main diagonal. Then the reangularity and semangularity for [**DM**] are equal to the product of their corresponding measures for each of the nonzero submatrices.

Theorem 10a (*R* and *S* for Decoupled Design) When the semangularity and reangularity are the same, the design is a coupled design.

Theorem 11 (Invariance) Reangularity and semangularity for a design matrix [**DM**] are invariant under alternative orderings of the *FR* and *DP* variables, as long as orderings preserve the association of each *FR* with its corresponding *DP*.

Theorem 12 (Sum of Information) The sum of information for a set of events is also information, provided that proper conditional probabilities are used when the events are not statistically independent.

Theorem 13 (Information Content of the Total System) If each *DP* is probabilistically independent of the other *DP*s, the information content of the total system is the sum of the information of all individual events associated with the set of *FR*s that must be satisfied.

Theorem 14 (Information Content of Coupled versus Uncoupled Designs) When the state of *FR*s is changed from one state to another in the functional domain, the information required for the change is greater for a coupled process than for an uncoupled process.

Theorem 15 (Design-Manufacturing Interface) When the manufacturing system compromises the independence of the *FR*s of the product, either the design of the product must be modified, or a new manufacturing process must be designed and/or used to maintain the independence of the *FR*s of the products.

Theorem 16 (Equality of Information Content) All information contents that are relevant to the design task are equally important regardless of their physical origin, and no weighting factor should be applied to them.

Theorem 17 (Design in the Absence of Complete Information) Design can proceed even in the absence of complete information only in the case of decoupled design if the missing information is related to the off-diagonal elements.

Theorem 18 (Importance of High Level Decisions) The quality of design depends on the selection of *FR*s and the mapping from domains to domains. Wrong selection of *FR*s made at the highest levels of design domains cannot be rectifed through the lower level design decisions.

Theorems for Design of Large Flexible Systems (from Suh [1])

Theorem 19 (The Best Design for Large Flexible Systems) The best design among the proposed designs for a large system that satisfy *n FR*s and the Independence Axiom can be chosen if the complete set of the subsets of {*FR*s} that the large system must satisfy over its life is known *a priori*.

Theorem 20 (The Need for Better Design for Large Flexible Systems) When the complete set of the subsets of {*FR*s} that a given large system must satisfy over its life is not known *a priori*, there is no guarantee that a specific design will always have the minimum information content for all possible subsets and thus, there is no guarantee that the same design is the best at *all times*, even if there are designs that satisfy the *FR*s and the Independence Axiom.

Theorem 21 (Improving the Probability of Success) The probability of choosing the best design for a large system increases as the known subsets of {*FR*s} that the system must satisfy approach the complete set that the system is likely to encounter during its life.

Theorem 22 (Infinite Adaptability versus Completeness) The large system with an infinite adaptability (or flexibility) may not represent the best design when the large system is used in a situation where the complete set of the subsets of {*FR*s} that the system must satisfy is known *a priori*.

Theorem 23 (Complexity of Large Flexible Systems) A large system is not necessarily complex if it has a high probability of satisfying the {*FR*s} specified for the system.

Theorem 24 (Quality of Design) The quality of design of a large flexible system is determined by the quality of the database, the proper selection of *FR*s, and the mapping process.

Theorems for Design and Operation of Large Organizations (Mostly from Suh [1])

Theorem 25 (Efficient Business Organization) In designing large organizations with finite resources, the most efficient organizational design is the one that specifically allows reconfiguration by changing the organizational structure, and by having a flexible personnel policy when a new set of *FR*s must be satisfied.

Theorem 26 (Large system with several sub-units) When a large system (e.g. organization) consists of several sub-units, each unit must satisfy independent subsets of {*FR*s} so as to eliminate the possibility of creating a resource intensive system or coupled design for the entire system.

Theorem 27 (Homogeneity of organizational structue) The organizational structure at a given level of the hierarchy must be either all functional or product-oriented to prevent the duplication of the effort and coupling.