chapter 3

System Development Framework (SDF) Overview

If you don't know where you're going, you'll end up somewhere else.
—Yogi Berra

In Chapter 2, the basic building block of the SDF was developed, based upon the collection of activities that is commonly associated with the System Engineering Process. These activities were derived from the consensus that emerged through a survey of the literature. It was suggested that there was little consensus, however, in terms of the arrangement of those activities into a consistent and coherent process. In this chapter, a top-level framework in which each of the identified activities can be arranged is developed.

In many industries, fierce competition is forcing cost and schedule resources to decrease substantially. In some industries, new programs are moving from financially safe cost-plus contracts to more risky fixed price contracts.[25] This puts much more pressure on the contractor to contain costs so that overruns that detract from profit can be minimized or avoided altogether. In order to increase the probability of earning a reasonable profit, it is essential that the bid reflect an accurate assessment of the scope of effort involved. In order to develop an accurate assessment of scope, an accurate understanding of the tasks involved and the resources required to perform those tasks is critical. Therefore, the more accurately the process for developing the system is defined, the more accurately the scope of the necessary effort can be determined.

[25] Cost-plus contracts generally assure the contractor that all costs incurred during the performance of the contract will be reimbursed by the customer. This type of contract is usually implemented where there is significant risk in the development of a new system or technology. In this type of contract, the customer agrees to bear some or all of the financial risk. Fixed price contracts, on the other hand, require that the system or product be delivered for the price agreed to by the contractor and customer at the time of the contract award regardless of the actual costs incurred by the contractor.

©2000 CRC Press LLC

I. Two Views Needed for an Accurate Model

The system engineering process must be defined with respect to two scales of time — the macro-scale and the micro-scale. In terms of the macro-scale, program development over large increments of time or program phases spanning from initial studies to the operational phase of the program is considered. This view of the program is called the Time Domain view. On the micro-scale, the concern is with information flow and energy expenditure at any instant in time. This view has been called the Logical Domain view. These two domains are more fully discussed below, but first the rationale for this approach is considered.

A. Rationale

Why emphasize this distinction between the macro and micro time scales? A central goal of this book is to define a process that accurately reflects what ought to occur on a well-ordered system development program in the real world. And why is this important? There are several reasons:

- **Cost Containment** — Provide a basis for accurate modeling of the program to facilitate more accurate estimates to complete, and for more accurate change impact analyses. This will serve to maximize the probability that the program will be completed within its cost and schedule constraints.
- **Scope Assessment** — Provide a basis for accurately assessing the scope of a development program which, in turn, facilitates more accurate bids for new contracts with more solid bases of estimate.
- **Metrics Development** — Provide a basis for accurate measuring of the state of the program as the development progresses.
- **Resource Planning** — Provide a basis for determining when specific resources will be required.

So, the first reason for this distinction is to provide an accurate model of what actually occurs in the real world of system development programs. A second goal of this book is to define a system development framework that can be applied to a wide variety of contexts. It is suggested that when the Time and Logical Domains are not explicitly identified and characterized in distinction, much difficulty arises in terms of defining a coherent and consistent development process that can be applied in a wide variety of contexts.

B. An Illustration

Consider the activity commonly called Verification. Verification is an activity that often consumes a significant percentage of overall program resources for its execution. Most of those resources are generally expended after the

©2000 CRC Press LLC

design work has been completed. In fact, some system engineering processes define this activity as occurring after the last major design review. However, is that an accurate or even desirable depiction of when this critical activity should take place — at the end of the design phase?

In order to minimize risk to the program, Verification issues must be considered at the earliest stages of development. Where will the system be tested? What facilities will be available? What interfaces must be satisfied? How will the system be transported? What special equipment will be needed? If these issues are considered during the early stages of design it will likely be possible to minimize the cost and schedule needed to perform the Verification activity.

Given that Verification must be considered throughout the system development, how should this be depicted in the System Engineering Process? The same question could be asked about any of the activities identified in Chapter 2. What about Requirements Development — does that end with the first major review? What about functional analysis, design, allocation, design integration, all the various analyses that must be performed, trade studies, etc.? Do all these occur in a strict serial fashion? Once these have been performed at some level, are they completed for the duration of the development? What about other levels in the hierarchy — is Verification performed only at the system level? The obvious answer to each of these questions is "No." Most of the activities of the system engineering process occur in parallel within the same hierarchical level of the development.[26] They also occur across the development timeline within the design activities of subsystems and sub-subsystems and so forth. But how should this be depicted as a logical process?

It is suggested that, in order to accurately capture what actually occurs in the real world of complex system development, the system engineering process must be defined in two domains — the macro-scale, or Time Domain, and the micro-scale, or Logical Domain. In the former, the design evolution that should occur over large increments of time is planned (design phases, manufacturing, integration and test, etc.). In the latter, the logical flow of the technical activities (Requirements Development, Synthesis, and Trade Studies) that should be occurring at any instant in time is defined.

Now, returning to the example, how should each of the technical activities be represented in the overall development process? It has been pointed out that each activity is generally performed at some level of intensity and for each system element during the entire development phase. Requirements Development, Synthesis, and Trade Study activities are all performed at different levels of intensity across the program timeline. These activities are generally performed in parallel, not in series. They often

[26] Brooks speaks of the "classical sequential or waterfall model," p. 265. He then goes on to offer a critique, "The basic fallacy of the waterfall model is that it assumes a project goes through the process once, that the architecture is excellent and easy to use, the implementation design is sound, and the realization is fixable as testing proceeds," p. 266.

©2000 CRC Press LLC

overlap as one activity feeds the next with data that is not necessarily complete. It is also generally agreed that these activities are iterated over the course of the development. So, how should this be depicted? If it is agreed that each of these activities generally occurs on some element of the system at some level in the hierarchy at some level of intensity, then each activity must be depicted as occurring at each point on the program timeline. It is not realistic to depict each of the technical activities as occurring in strict serial fashion along the program timeline. Therefore, it is suggested that the concept of two domains be employed because this facilitates an accurate model of what actually occurs in the real world of complex system development.

II. Time and Logical Domain Views Provide a Full Program Description

As mentioned above, the macro-scale, or Time Domain, considers the System Development activity as viewed across the entire life cycle of the program. It is concerned with how inputs and outputs evolve over time as the system design matures. The micro-scale or Logical Domain, on the other hand, deals with what occurs within small increments of time. Imagine instantaneous "snapshots" along the macro-timeline, or time continuum, where there are an infinite number of "logical planes." Each "snapshot," or plane normal to the macro-timeline, represents an infinitesimally small slice of time. Each slice, or plane, reveals the instantaneous logical sequencing of activity occurring at that time. Figure 3.1 illustrates the Time Domain view of the program and the instantaneous "snapshots" that are provided by the Logical Domain view. It further depicts how these two views combine to fully define the program.

A. Time Domain Focus: Inputs and Outputs

The Time Domain view characterizes how the design evolves chronologically. The essential "value-added" element of this view is a clear definition of inputs and outputs as they are developed over time. The focus of this view is not the activities being performed, but rather the outputs generated by those activities. It is the outputs that change significantly from phase to phase. The definition of these outputs during the planning process becomes one of the primary bases by which the program scope is assessed.

Note here that the output of a previous activity becomes the input to the subsequent activity. As Figure 3.1 indicates, there are two distinct sets of data that are output: requirements and design. Notice that the requirement outputs lead the design outputs. The logic is that the requirements at any given level of the hierarchy drive the design being generated at the same level. This is discussed in more detail in Chapter 5.

©2000 CRC Press LLC

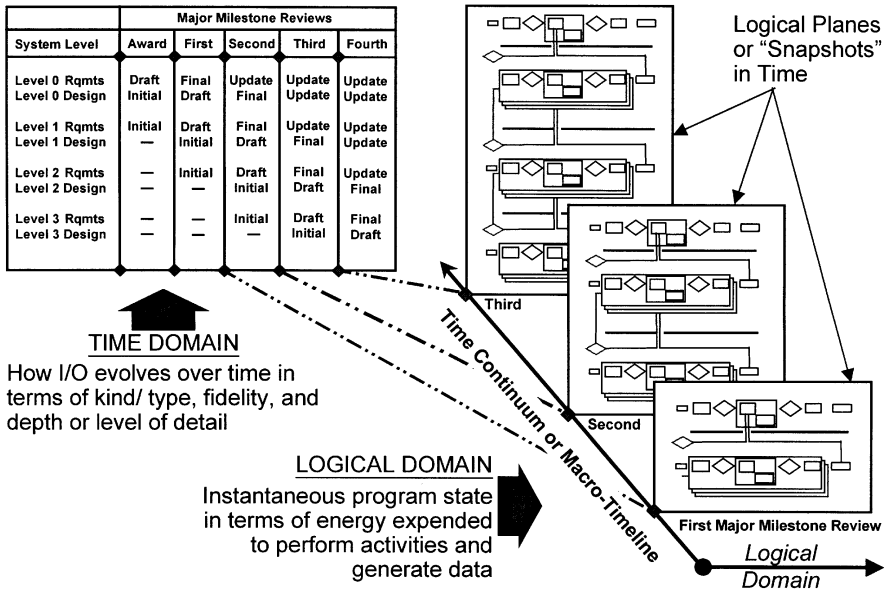

Figure 3.1 Time and Logical Domain Coupling.

B. Logical Domain Focus: Energy Expenditure

The Logical Domain view provides an instantaneous snapshot in time of the program state, revealing which activities are being performed at each hierarchical level, what the sequencing of those activities is, and how much energy is being applied in the performance of each activity.

Energy is expended in each activity (i.e., requirements development, synthesis, and trades) until the desired output at the necessary fidelity is generated. This "energy" refers to the manpower and other resources needed to generate the required output. It is the scope of the effort necessary. Thus the more accurate the estimates for outputs required and the inputs needed to generate the outputs, the more accurate the manpower and other resource estimates will be.

In any well-planned complex development program there are major milestones where a certain level of design definition is planned to be achieved. Figure 3.1 illustrates three such milestones. The first major milestone is concerned with defining the top-level system architecture. The second focuses on defining the architectures of the various subsystems. The third and subsequent major milestones generate the designs of the lower level elements. Each plane reveals how many levels of the hierarchy are involved and how many subsystems are at each level, what the logical connections are within and between each of the levels, and the energy level applied to each activity.

©2000 CRC Press LLC

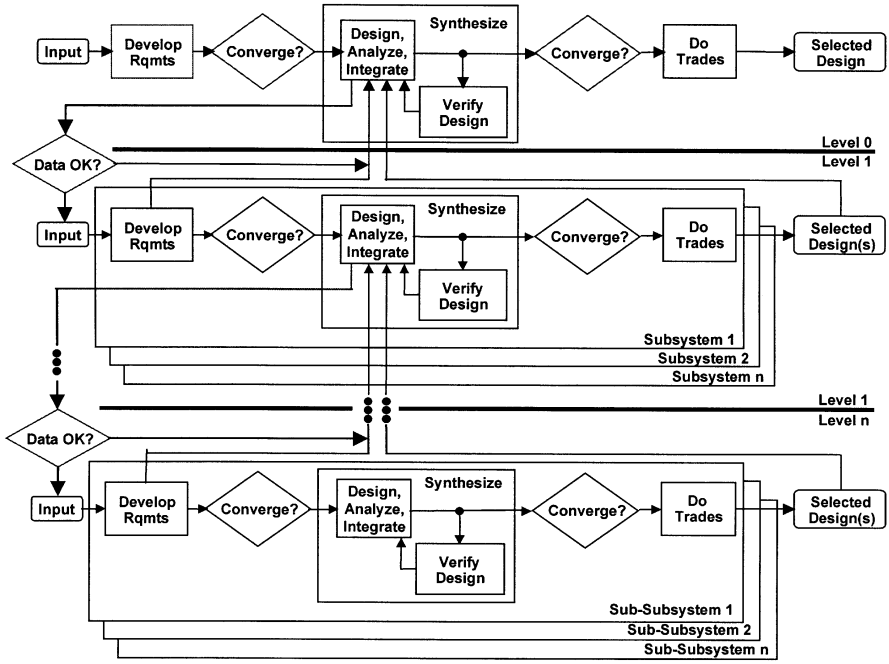

Figure 3.2 The SDF Logical View.

III. The SDF in the Logical Domain

Figure 3.2 enhances Figure 2.2 of Chapter 2 by including decision points for convergence, adding tiers to the program, and indicating connectivity between the tiers. It illustrates that there is control logic coupling the various modules of activity between and within the tiers of the system hierarchy. Several important points are implied by Figure 3.2.

A. Control Logic

One of the key elements of the SDF is the clear delineation of the data flow-down and feedback paths that connect same-tier and adjacent-tier activities. This "control logic" defines the paths and the gates through which information flows within the overall hierarchy. This is developed in detail in Chapter 5, where Figure 3.2 is decomposed to the next level.

B. Hierarchy

The Moses of the Bible implemented a hierarchy of leadership in his governing the nation of Israel (Exodus 18:13-27). Hierarchy is an essential element of organizing any complex system. The SDF building block, mentioned above, defines the basic activities and the logical sequencing of those activities for each tier in the program hierarchy.

©2000 CRC Press LLC

At any instant in time it is likely that each activity of the SDF is being performed at some level of intensity and at some level in the system hierarchy. The level of intensity applied to each activity is dependent upon a whole array of variables: stability of the input requirements, level of complexity of the system, whether the system is precedented or not, where on the program timeline the development effort is occurring, etc.

C. *Modularity*

The SDF is modular and therefore tailorable. Tailoring is accomplished by partitioning the program hierarchy appropriately, adding tiers as necessary, and by adding same-tier elements as needed.

D. *Closed Loop*

The SDF is closed loop, with information flowing down from the Design, Analyze, and Integrate activity to the next tier, and with data flowing back up from the lower tier into the Design and Integrate activity of the tier above. This is an important point because it is this organization that can preclude design teams from spending resources and increasing costs by designing in capabilities that are not required. Conversely, it can also be used to ensure design teams are responding to all design requirements, thus eliminating expensive redesign needed to include functionality that was missed.

E. *Traceability*

The SDF provides a structure for program requirements databases. This logical flow of activity ought to reflect the manner in which requirements flow up and down throughout the system hierarchy. Therefore, this same structure should be used in designing the requirements traceability system, which includes not only requirements flow, but also verification method and the stage in the system build-up at which the verification will be performed. It also provides the basis for change impact analyses, sensitivity analyses, cycle-time reduction analyses, etc.[27]

F. *Comprehensiveness*

It is important to note that each activity of which the SDF is composed considers not only the development of the deliverable product itself, but also all of the associated hardware, software, procedures, processes, etc. needed to produce, integrate, test, deploy, operate, support, and dispose of the system. The effort applied to the development of support elements is commensurate with program need.

[27] Cf. Adamsen (1995)

©2000 CRC Press LLC

G. Convergence

With regard to the flow of the process, the key criterion in moving from Requirements Development (RD) to Synthesis is convergence of the RD activity. Requirements convergence has to do with defining a set of requirements that are stable enough to proceed to the Synthesis activity with acceptable probability that an adequate solution can be generated. The requirements set will never reach absolute perfection. *The issue is whether or not the risk associated with proceeding to the next activity is acceptable.*

An example of non-convergence occurred on a space program in which the author was involved. The spacecraft was required to communicate with a relay satellite during a time when the two prescribed orbits precluded such communication. Because this was an impossible requirements set to satisfy, the requirements activity could not converge. The requirements set needed to be changed in order to enable the Requirements Development activity to converge upon a solution. Another example is the case in which a function cannot be performed without input from another function. In such a situation, the input and output requirements of the functions must be coordinated. If this is not possible, the Requirements activity can not converge.

The question of convergence relative to the Synthesis activity is similar to that of the Requirements activity. Examples of non-convergence might include the unavailability of certain technologies required to satisfy a particular requirements set. The requirements might be perfectly stable and consistent, but they are not implementable — at least not at a reasonable cost or schedule. A feedback path to the input source is provided in the process when convergence is not achieved.[28]

To summarize, convergence occurs when the output data has achieved a level of acceptable risk in terms of the probability of success that the subsequent activity will be able to reach convergence with that data.[29]

H. Risk

Risk management is an important tool by which to manage the development of any complex system. The primary difficulty arises in assessing it accurately and in a meaningful way. Nevertheless, a risk assessment of the output data ought to be an essential criterion in making the determination whether or not to proceed to the subsequent activity. The risk assessment should focus on determining if the requirement and design output are of such sufficient fidelity that the downstream activities can commence with an acceptable probability of success.

[28] In order to avoid clutter in Figure 3.2, the feedback paths to the input source are not shown. They are, however, shown explicitly in Figure 5.42 of Chapter 5.

[29] A key issue here is accurately quantifying the risk to the program if the subsequent activity proceeds with the data input to it.

©2000 CRC Press LLC

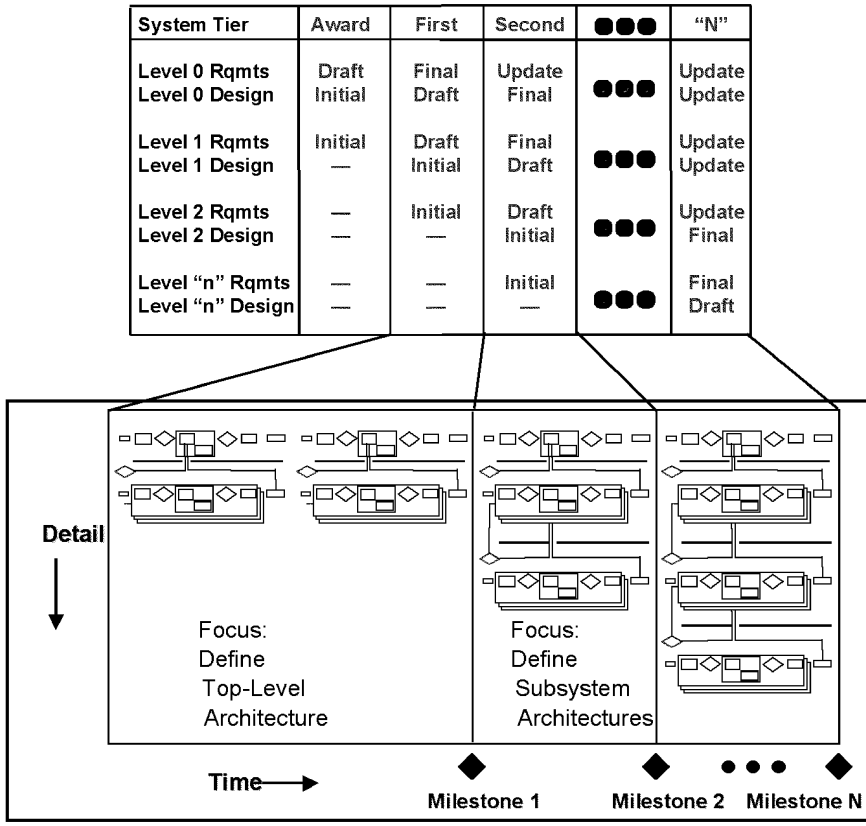

Figure 3.3 The SDF in the Time Domain.

IV. The SDF in the Time Domain

Having described how the SDF functions in the Logical Domain, the question of how it functions in the Time Domain is now addressed. Figure 3.3 depicts the SDF in the Time Domain. During the early stages of the development, the primary focus of activity centers on defining the top-level system architecture. The figure illustrates several top-level architecture candidates being evaluated during the first phase of activity. The conclusion of that phase of development culminates in the selection of the baseline architecture. After that point, the focus of activity moves to development of the major subsystems at the next level down in the hierarchy. This introduces the concept of "incremental solidification."

A. Incremental Solidification

In order to proceed along the program timeline with minimal risk, it is necessary to "incrementally solidify" key requirements by program milestone. It

©2000 CRC Press LLC

is highly desirable to determine which requirements are needed from the customer (or other entity) and when, in order to maintain progress with minimal risk. One goal is to include these critical need dates in the contract so that if there is delay, a cost and schedule scope change can be effectively negotiated. It is also apparent from the figure that an instability in upper-tier requirements or design results in instability at each dependent tier below it. Thus, unstable requirements or design at an upper level induce risk into lower dependent levels.

B. Risk Tolerance Defines Scope

Notice in Figure 3.3 that only one level of activity below the top-level has been shown during the first phase. This is intended to illustrate that, at early stages of development, lower levels of design and analysis are only performed in order to support the development of the top-level architecture. In other words, the degree of lower-level analysis necessary is determined by the amount of risk the program is willing to carry. If there is little uncertainty that the lower level design can perform as required, then there is little reason to perform detail design and analysis at that point on the timeline. Conversely, if there is great uncertainty that a key lower-level design will be able to perform as required, then significant lower level design and analysis may be necessary to lower the program risk to an acceptable level.

To illustrate this point, consider the conceptual design of a spacecraft constellation in which direct communication between satellites within the constellation is necessary. The method of communication is a key issue in the top-level conceptualization of the system. Suppose that for various reasons the communication method of choice is a laser cross-link. Because of the uncertainties involved in this technology, it may be necessary to perform a significant level of detailed analyses at relatively low levels in the system hierarchy in order to mitigate perceived risks with such a concept. How accurately must the laser be pointed? Will it be possible to point the laser to that degree of accuracy? How much onboard electrical power will be necessary to support the cross-link? It may not be possible to answer these questions without significant design and analysis during the early stages of development. The extent of design and analysis necessary will be a function of how much risk the program is willing to tolerate. The more risk adverse, the more design and analysis at lower levels will be required to mitigate perceived risk.

C. Time-Phased Outputs

Another feature of the Time Domain view is that it defines which particular outputs are required, at what fidelity, and when on the program timeline. *To reiterate, the basic activities of the SDF do not change over time. However, the outputs of those activities change dramatically over the course of the development.* The output of a structural analysis during conceptual studies will be quite

©2000 CRC Press LLC

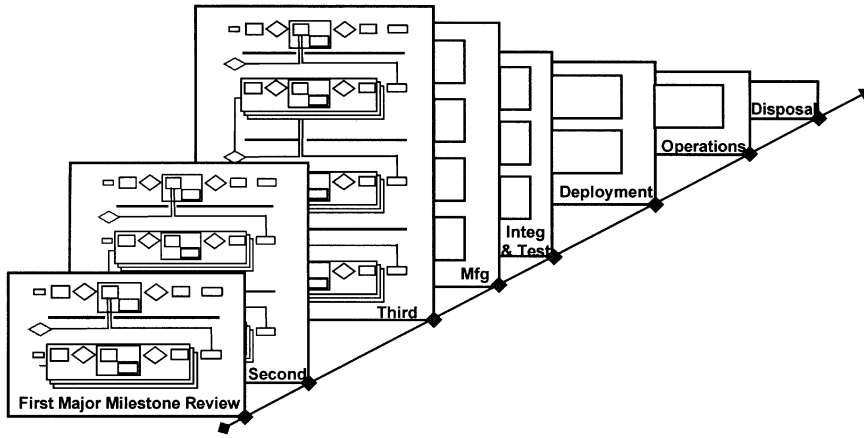

Figure 3.4 Full System Life Cycle.

different from those performed prior to a detail design review. Figure 3.3 indicates that there are two major types of outputs — requirements and design. The figure also illustrates that requirements should "lead" design because design activities respond to requirements. Over time, outputs become more detailed and are generated at increasingly lower levels of the hierarchy.

V. System Life Cycle

Figure 3.4 illustrates the full life cycle for a typical system, addressing how the teams responsible for particular system elements function over time. Supporting teams generally stay intact through Fabrication, Assembly, and Test phases of the program. As the program moves from development to production the composition of the teams may vary widely, moving from an engineering emphasis to production. As the program matures through production, deployment and operations lower-level teams may be subsumed under higher as appropriate.[30]

It is important to consider the full life cycle of the system at the earliest stages of the development because each mission phase imposes unique requirements on the system. In order to maximize the probability of success, these requirements must be considered from the start.

Figure 3.4 further illustrates that the design effort continues up to the final Major Milestone Review that concludes the design phase of the program. After that, the program moves from significant design effort (depicted in the first three "snapshots" with the technical activities shown as per Figure 3.2) to the subsequent phases of manufacturing, integration and test,

[30] Adamsen, Paul B., Jr., Controlling The Chaos: An Integrated Approach To Managing A System Development Program, "Systems Engineering Practices and Tools," *Proceedings Sixth Annual Symposium INCOSE*, Vol. 1, July 7-11, 1996, Boston, MA, pp. 1093-1100.

©2000 CRC Press LLC

deployment, operations, and disposal. For a production program, it is important to provide feedback to the design activity, capturing lessons learned from the deployed systems.

©2000 CRC Press LLC