chapter 6

The System Development Framework — Managerial

It's easy to play any musical instrument:
all you have to do is touch the right key at the right time
and the instrument will play itself.
—Johann Sebastian Bach

The other teams could make trouble for us if they win.
—Yogi Berra

I. Integrating Technical and Managerial Activities

As Figure 6.1 illustrates, activities composing the management effort include Risk Management, development and tracking of metrics and technical performance measures, Configuration Management, etc. The technical effort comprises Requirements Development, Synthesis, and Trade Analyses. Because these activities are closely related, they must be coupled for efficient management of the development. The SDF can be used to effectively link these together.

II. Developing the Program Structure

How should the program be partitioned to enhance the efficiency of the development? How should information flow from one development team to another? Where does each development team get its requirements? Who should be responsible for identifying, characterizing, and controlling interfaces? The SDF can be used to address these key questions.

Logically, the first task in developing the program structure is to partition the system into its constituent elements. In the case of a precedented system, the first level decomposition is usually well understood. A low earth orbiting spacecraft bus, for example, is often partitioned into the following subsystems: structures and mechanisms, thermal control, electrical power,

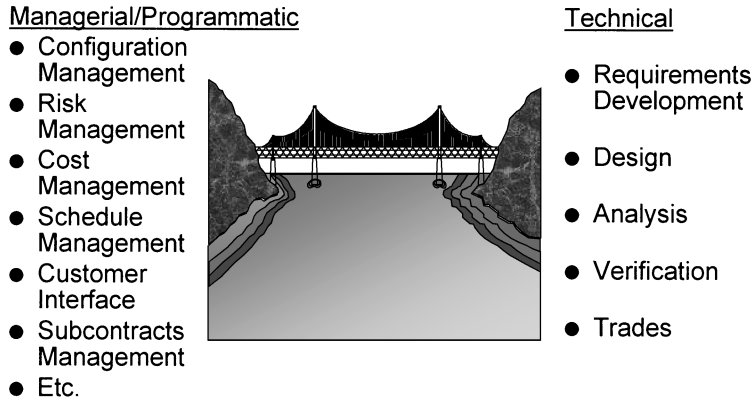


Figure 6.1 Technical and Managerial Activities[61]

attitude determination and control, propulsion, communications, and command and data handling. It is often the case that the development teams are organized in the same way. A significant benefit of doing this is that the interfaces and roles and responsibilities are well understood. In the case of an unprecedented system, the partitioning of the functional teams should be guided by the current system description to provide a first-cut hierarchical arrangement of functional teams.

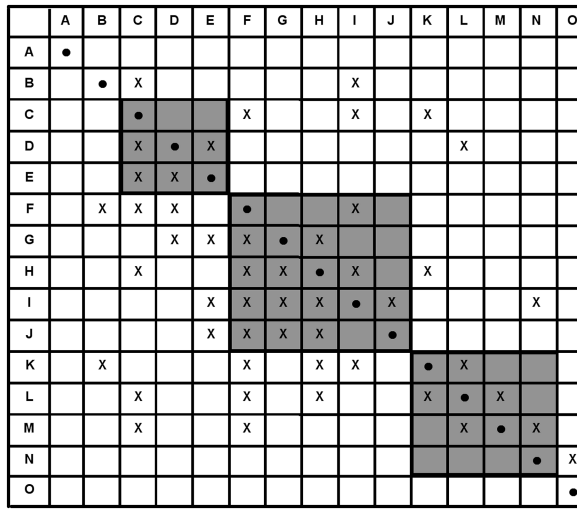
With regard to information flow, it is often the case that problems within a system occur at an interface.

Key Point

> Therefore, a key objective in partitioning the system into subelements is the minimization of the number of interfaces between partitioned elements.

The Design Structure Matrix (DSM) can be used to determine the best partitioning of the program elements as well as the functional teams. Figure 6.2 illustrates a typical DSM. Each system element is represented in the matrix down the first column and across the first row. The interfaces between elements are indicated by an "X" where the appropriate column and row intersect. The diagonal is obviously left blank. Partitioning is indicated by a box drawn around the group of elements that make up each

[61] Adapted from Adamsen's presentation charts presented at the 1996 INCOSE Symposium. Cf. Rochecouste, who develops a similar listing of activities. Systems Engineering Process/Activities: Requirements Analysis, Functional Analysis, Design Synthesis and Specifications, Subsystem Integration, System Test and Evaluation, etc. Technical Management Activities: Technical Planning, Requirements Management, Configuration Management, Design Reviewing, Technical Risk Management, Technical Performance Measurement, etc. Rochecouste, Hervé. *A Systems Engineering Capability in the Global Market Place*. P004.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A	•														
B		•	X						X						
C			•			X			X		X				
D			X	•	X							X			
E			X	X	•										
F		X	X	X		•			X						
G				X	X	X	•	X							
H			X			X	X	•	X		X				
I					X	X	X	X	•	X				X	
J					X	X	X	X		•					
K		X				X		X	X		•	X			
L			X			X		X			X	•	X		
M			X			X						X	•	X	
N														•	X
O															•

Figure 6.2 The Design Structure Matrix (DSM).

subsystem. A goal is to arrange the order of the elements such that the interfaces between them are minimal in number.[62]

Obviously in the case of precedented systems, there may be limited flexibility in terms of repartitioning established subsystems. However, whenever possible, interfaces between elements should be minimized. This applies to the creation of development teams on the program. If the system design has been efficiently partitioned so as to minimize interfaces, this will likely provide an efficient model for partitioning of functional teams as well. For this reason, it is advocated that the team structure should follow the structure of the partitioned system insofar as it makes sense to do so.

Now that the question regarding how to partition the program hierarchy has been addressed, the question concerning how these partitioned elements ought to interact is discussed. It is suggested that the logical flow of information between the partitioned elements should be provided by the control logic defined in the SDF.

Figure 6.3 illustrates the development of the program structure from the existing system design partitioned so as to minimize the interfaces between system elements. It also uses the "control logic" of the SDF to define the information flow paths within the total system hierarchy.

[62] A thorough treatment of the subject of Design Structure Matrix methodology is beyond the scope of this book. For more information, see Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith, and David A. Gebala, A Model-Based Method for Organizing Tasks in Product Development, *Research in Engineering Design*, 6:1-13, 1994, and Kent R. McCord and Steven D. Eppinger, Managing the Integration Problem in Concurrent Engineering, MIT Sloan School of Management, Working Paper Number 3594, August 1993. See also Rosaline K. Gulati and Steven D. Eppinger, The Coupling of Product Architecture and Organizational Structure Decisions, MIT Sloan School of Management, International Center for Research on The Management of Technology, Working Paper Number 151-96.

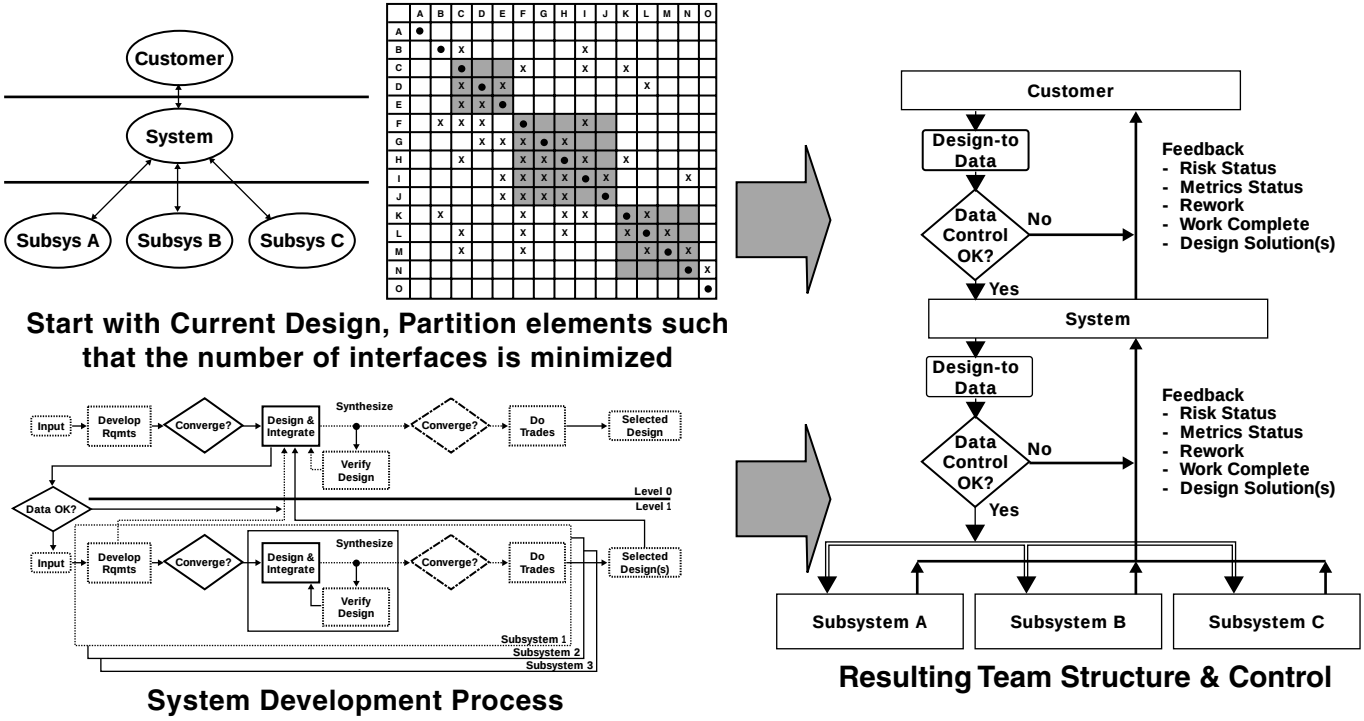


Figure 6.3 Program Structure Development.

There are several reasons for applying the control logic provided by the SDF to the program structure:[63]

- Defines activity flow and information flow between and within development teams
- Provides basis for requirements, design, and decision database structure
- Identifies interfaces between tiers of system hierarchy
- Defines team roles and responsibilities in context of total system hierarchy
- Defines activities more precisely enabling more precise progress measurement

III. Interaction in the Logical Domain

Figure 6.4 illustrates the horizontal and vertical interfaces between the various functional teams in the program hierarchy. These interfaces are derived directly from the SDF. All "design-to" data is passed to a lower level from the team at the level directly above. This is also true of feedback. All such data is communicated to the level directly above. Data travels horizontally through the team at the level above. It is the responsibility of that team to disseminate information to the appropriate teams under its jurisdiction. Remember, it is formal interfaces that are being addressed here. Of course, it is desirable for teams on the same horizontal level to communicate and interact.

Key Point

> All design data, however, must pass through and be coordinated by the team at the level above. This is important in order to maintain control of the information and to ensure that all information is communicated accurately and in a timely manner.

It is also important to maintain such control when changes occur. The appropriate reviewers must be consulted so that all impacts resulting from the change are properly assessed. Any modifications to the design made to accommodate such changes must be coordinated so as not to introduce new problems into the design.

Each lower level element functions semi-autonomously from its next level up element as long as it stays within its allocations (i.e., prescribed boundary conditions). Periodic status is provided to the level above in terms of technical performance measures, risk assessments and mitigation approaches, design issues and solutions, etc. If an allocation is violated, the

[63] Adamsen (1996), pp. 1093-1100.

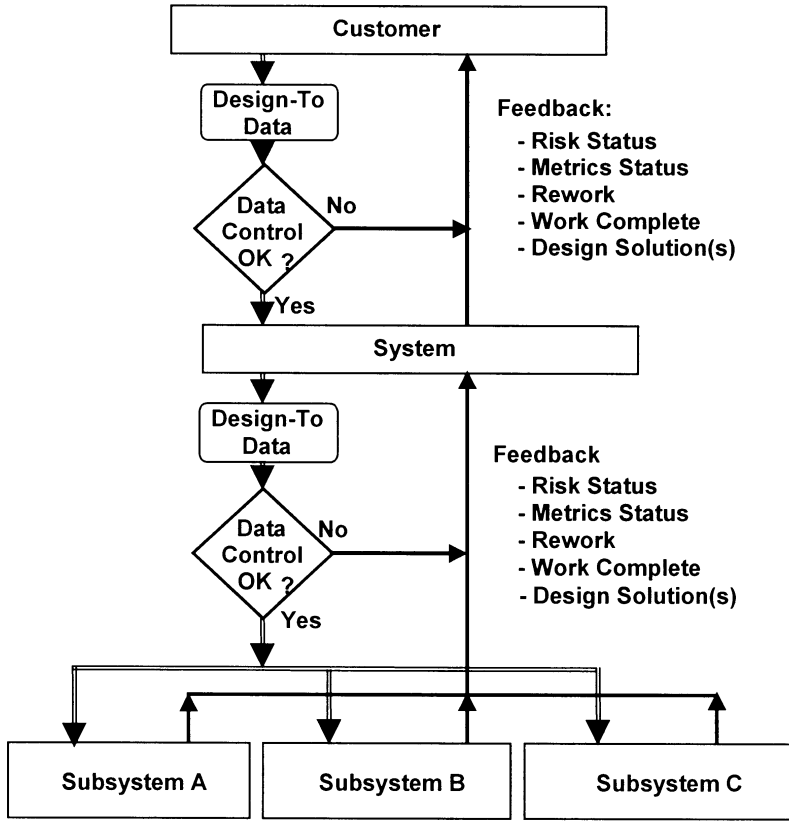


Figure 6.4 Program Team Interactions.[64]

next level up element becomes involved to either reallocate or modify the design at its level.

IV. Interaction in the Time Domain

Having described how the SDF drives the interactions of teams in the logical domain, the question of how it functions in the time domain is now addressed. Figure 6.5 depicts the SDF in the time domain. Along the timeline, the number of options converges to a single solution while the level of detail definition increases. As discussed in previous chapters, the time domain view describes how the generation of data evolves as a function of time. This includes prescribing the level of fidelity of each output required at a particular point on the program timeline for each tier of the hierarchy. As discussed previously in Chapter 3, in order to proceed along the program timeline with minimal risk, it is necessary to "incrementally solidify" key requirements by program milestone.

[64] *Ibid.*

Hierarchical Tier	Award	First	Second	Third	Fourth
Level 0 Rqmts Level 0 Design	>70% >50%	>90% >70%	Update >90%	Update Update	Update Update
Level 1 Rqmts Level 1 Design	>50% --	>70% >50%	>90% >70%	Update >90%	Update Update
Level 2 Rqmts Level 2 Design	-- --	>50% --	>70% >50%	>90% >70%	Update >90%
Level 3 Rqmts Level 3 Design	-- --	-- --	>50% --	>70% >50%	>90% >70%

Figure 6.5 Time Domain View.

Key Point

> It is highly desirable to determine which requirements are needed from the customer and when, in order to maintain progress with minimal risk. The 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 negotiated.

Figure 6.5 illustrates how requirements and design data are progressively generated. Four stages of increasing fidelity are illustrated: 50%, 70%, 90%, and update.[65] Requirements lead the development of the implementation by one stage of fidelity. First, the system level requirements are generated to a fidelity level of, say, 70%. Once the requirements set has reached this level (or a level of acceptable risk), the Synthesis activity can begin with an acceptable level of risk. As the Synthesis activity progresses toward an increasing level of fidelity, the requirements are also increasing in fidelity as they are validated against the developing design.

Also illustrated is the progression of the Development activity down the hierarchy as a function of time. As upper levels become increasingly stable, lower-level requirements and Synthesis activities are initiated at acceptable levels of risk. Thus, the key criterion for vertical (e.g., system to subsystem) and horizontal (e.g., movement from Requirements Development to Synthesis or passing a Major Milestone Review) progression is attaining to an acceptable level of risk. This suggests that the more accurately risk can be quantified, the better the program can be managed in terms of meeting cost and schedule goals.

[65] In this context, fidelity refers to the completeness of the requirements or the certainty of the requirements in terms of stability or risk. The quantification of the status is admittedly subjective and represents a relative measure and not an absolute one.

Key Point

> The above discussion indicates the necessity of solidifying the requirements to some level before design activities are initiated in order to minimize the risk of a nonconverging or slow-to-converge Synthesis effort. It further illustrates the necessity of solidifying the design before requirements for the next level down are passed along, thereby initiating the development activities at that next level.

This is one cause of overruns on development programs. Requirements are not stabilized prior to initialization of the synthesis activity. Or, the design is not stabilized prior to initializing the next-level-down development activities. This implies that a structured approach to system development is needed.

V. A Note on Complexity

Systems comprise various subsystems which also comprise several sub-subsystems and so on. It is intuitively apparent that complexity grows as more subsystems and tiers are added to the system. Furthermore, growth in complexity appears to be non-linear in that it grows so quickly on many development programs.

It might be helpful to quantify the growth of complexity in order to understand its impact on a program. Figure 6.6 depicts four tiers in the hierarchy, with three subsystems below each system above. It is clear that growth in complexity is an exponential function. It has been assumed that each subsystem comprises only three subsystems. This is quite conservative as many systems comprise seven subsystems or more. For example, a typical spacecraft can have seven or eight subsystems, while an aircraft can have more still.

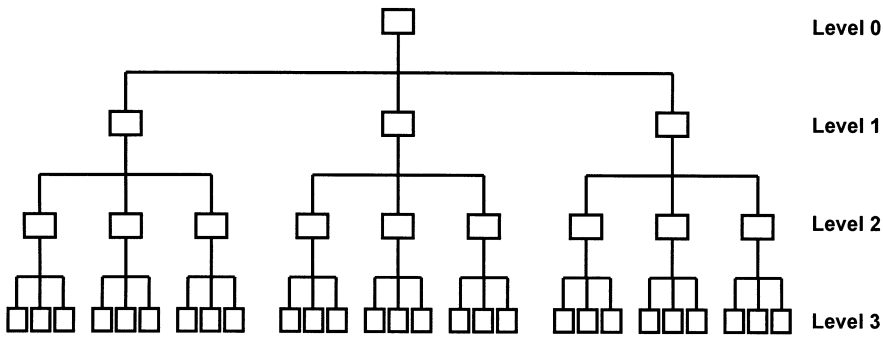


Figure 6.6 Exponential Growth in Complexity.

To put this in mathematical terms, if "S" represents the number of subsystems per system and "n" the n^{th} tier in the hierarchy, then the following relationship emerges:

$$Total\ System\ Elements = \sum_{n=0}^{m} S^n \qquad 9910(6.1)$$

The Total System Elements (TSE) refers to the total number of hardware and software pieces of the system. Therefore, as the TSE grows, the overall complexity also grows as does the energy needed to develop each element. As has been suggested previously, in this context energy refers to manpower and all other resources needed to perform the development of all the system elements.

Figure 6.7 is a graphical depiction of the above relationship. One curve is plotted for the assumed number of subsystems below each system. Figure 6.6 is a picture of the case where three subsystems are assumed for each system. The total number of elements included in Figure 6.6 is 40. This is represented in Figure 6.7 by the curve labeled "3" in the legend. It is apparent from the graph that complexity increases rapidly as subsystems per tier are added and as tiers are added to the system hierarchy.

VI. Major Milestone Reviews

A central motivation for conducting a technical review is to gain consensus with all stakeholders that the requirements and design, at the level of the review, are of sufficiently low risk or acceptable risk, so as to continue toward the next milestone. There are relatively few complex system development efforts that are accomplished on schedule and within the proposed cost. One reason may be that the purpose of each Major Milestone Review is not clearly defined. Both the customer and the contractor must know when critical requirements are needed in order for the development activity to proceed with an acceptable amount of risk. As discussed previously, if requirements or implementation are unstable at upper levels, the risk induced can be significant. Therefore, the following is offered as suggested objectives for each review.[66]

First Major Milestone Review — This review is conducted to solidify the system requirements in the form of a configuration-controlled system-level specification and the Interface Control Document (ICD) that defines *external* interfaces before significant resources are invested in developing the system-level design. This review focuses on understanding the total context in which the system must function over its full life cycle. It assesses the risk profile of the program in order to determine whether or not to proceed toward the Second Major Milestone Review.

[66] See Appendix C for SDF-derived major milestone review criteria.

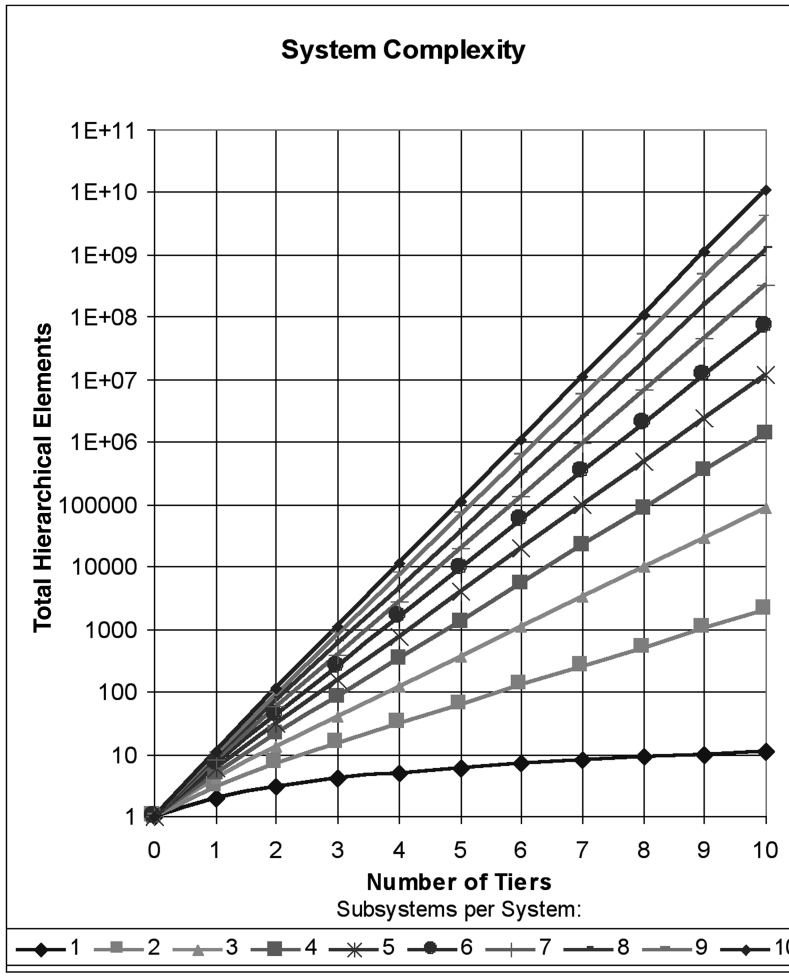


Figure 6.7 Complexity Growth.

Second Major Milestone Review — This review is conducted to establish the baseline system-level architecture by configuration controlling the system block diagram, ICDs that characterize and control system-level *internal* interfaces, and the Operations Concept. Successful completion of this review results in the release of the configuration-controlled specifications for the next level system elements. It assesses the risk profile of the program in order to determine whether or not to proceed toward the Third Major Milestone Review.

Third Major Milestone Review — This review is conducted to establish the subsystem baseline designs by configuration controlling the subsystem block diagrams and ICDs that characterize and control subsystem-level internal interfaces. Successful completion of this review results in the release

of the configuration-controlled specifications for the next level system elements. It assesses the risk profile of the program in order to determine whether or not to proceed toward the Fourth Major Milestone Review.

Fourth Major Milestone Review — In general, this review is conducted to establish the component-level baseline designs and any necessary next-level-down specifications by configuration controlling the component block diagrams and any necessary ICDs that characterize and control component-level internal interfaces. Successful completion of this review results in the release of any remaining and necessary configuration-controlled specifications for the next level system elements. It assesses the risk profile of the program in order to determine whether or not to proceed toward the build phase of development. Note that in a very large system with many hierarchical tiers it may be necessary to add more major milestone reviews. In such a case, the basic flow and content of the reviews remains the same as indicated in the preceding.

VII. What About Metrics?

Certainly, a central concern of any program manager is how to determine the true status of the development. Is the right amount of progress being made? What is the projected cost to complete the development of the system? When will the development phase be completed? What are the risks and their potential impacts?

Meaningful metrics in the area of system engineering have been difficult to define. One reason for this difficulty may be the lack of a well-defined system engineering process that can be consistently applied across a broad range of programs. It is virtually impossible to implement a process that is inadequately defined and it is likewise difficult to measure progress against such a process.

The SDF is defined in sufficient detail to enable the implementation of meaningful metrics. Each major activity (i.e., Requirements Development, Synthesis, and Trades) and each sub103103activity (e.g., requirements analysis, functional analysis, design, allocation, analysis, integration, verification, etc.) is allocated cost, schedule, manpower, computer, and other resources. Actual consumption of these resources is tracked against the allocated resource plan to measure progress. These data are cataloged in a database. As more programs are performed, the metrics are refined in terms of the necessary allocation of resources for each activity. The database becomes the benchmark against which future programs are measured. In order to ensure that apples are compared to apples and not to oranges, this approach necessitates that each program implement the same SDF. Such a database would be useful for estimating costs for new proposals and for internal estimates of completion.