chapter 7

A Potpourri of SDF-Derived Principles

Whether we will philosophize or we won't philosophize, we must philosophize.
—Aristotle

If I wished to punish a province, I would have it governed by philosophers.
—Frederick II, the Great

The following principles have been derived during the course of study that lead to the development of the SDF described in the preceding chapters. These principles or heuristics have been collected under several broad categories, none of which are absolute.

I. General

1.1 A system architect is responsible to define the interfaces internal to his/her system, but he/she may not have control over those interfaces external to his/her system.

1.2 A sound architecture and a successfully managed program consider the total context within which the system must function over its full life cycle. It is not sufficient to develop an architecture for a deployed system that does not consider all contexts in which the system must function: manufacturing, integration and test, deployment, initialization, normal operations, maintenance operations, special modes and states, and disposal activities.

II. Risk

2.1 Acceptable risk is a key criterion in deciding to move from one activity to the next. The challenge is to quantify risk in an accurate and meaningful way.

©2000 CRC Press LLC

2.2 Allocation of resources is a key basis by which to measure and monitor risk. Risk in a system development activity can result from insufficient margin in technical, cost, and/or schedule allocations.
2.3 Program risk increases exponentially with requirements instability. Because upper-level requirements drive lower-level designs and requirements, and because the number of system elements increases exponentially down the program hierarchy, a few unstable top-level requirements can affect many lower-level system elements.
2.4 Risk is difficult to manage if it has not been identified.
2.5 In conceptual architecting, the level of detail needed is defined by the confidence level desired or the acceptable risk level that the concept is feasible.

III. Functional Analysis

3.1 A function cannot be decomposed without some reference to implementation. That which enables the decomposition of a function is knowledge and/or assumptions about its implementation.
3.2 A functional decomposition must be unique. Prescribed functionality describes "what" the system must do and, in order to be self-consistent, that functional description must be unique.
3.3 Function partitioning is not unique. There are many ways to partition functions.
3.4 The functional definition must include both functionality and associated performance in order to be useful in implementation. It is necessary but not sufficient for implementation to define only required functionality. Performance must also be prescribed in order for a function to be implemented in a meaningful way.
3.5 Within the same tier, the "what" (Requirements Development) drives the "how" (Synthesis activity). Form must follow function. Implementation, by definition, performs a function. It is not rational to try to determine "how" to perform a function that has not been identified.
3.6 With respect to interaction between hierarchical tiers, the "how" above drives the "what" below. This is a very important principle and has implications in areas such as specification development. When a customer or next-level-up design team defines functionality in a specification several tiers down, the probability of introducing problems increases because the intermediate decompositions may not be consistent with the prescribed requirements.

IV. Allocation

4.1 Margin unknown is margin lost. In order to optimally manage a system development, all system margin must be known to the architect having authority to allocate it. It is generally cost-effective to

©2000 CRC Press LLC

reallocate resources to handle issues. In order to do this effectively, the architect needs to know where the margin resides.

4.2 During an architecting effort, cost and schedule should be allocated and managed just as any other technical resource.

V. Process

5.1 Process understanding is no substitute for technical understanding. This is exemplified by the principle that a function cannot be decomposed apart from implementation. It is the technical understanding of the architecture implementation that facilitates the decomposition.

5.2 Before a process can be improved it must be described.

5.3 Tools should *support* the system development process, not *drive* it.

5.4 A central purpose of the SDF is to provide every stakeholder with a pathway for understanding the system and the state of its development.

5.5 There is a necessary order in which technical activities must be performed. Some notion of "what" the system must do must precede any effort to determine "how" to do it. Some notion of "how" the system will be implemented must precede any determination of "how well" it performs. Because trade analyses are based upon cost, schedule, and/or technical criteria, they must follow the Synthesis activity. A trade analysis cannot be performed without some definition of implementation. Trade analyses are based upon cost, schedule, and technical criteria. These cannot be determined without some relation to implementation.

5.6 Any technical activity can be categorized as either a "what," "how," "how well," "verify," or "select" activity. This is the primary organizing principle of the generalized SDF.

5.7 Describing the System Engineering Process in both the time domain (output evolution) and the logical domain (energy distribution) facilitates its application to many contents.

5.8 A system is "any entity within prescribed boundaries that performs work on an input in order to generate an output."

5.9 Given the above definition of "system," the same system development process can be applied at any level of design development.

5.10 The development process must include not only the deployed system but also all necessary supporting entities. A sound architecture involves the architecture of supporting system elements as well as the element that actually performs the mission functions directly.

5.11 Interface control is always the responsibility of the element in which the functional decomposition occurs. Interface identification and characterization must involve the elements at which they occur, but their control is the responsibility of the element above which the interface occurs.

©2000 CRC Press LLC

5.12 The technical process of architecting a system should drive the approach for managing the system development. Interfaces can (and should be) defined by the technical process. These help determine roles and responsibilities of teams, information flow and control, subcontract boundaries, etc.

VI. Iteration

6.1 Iteration generally occurs for one of three reasons: optimization, derivation, or correction. Optimization of a design (at some level of fidelity), by definition, results in a change to that design. Where change is necessitated, feedback and/or iteration occurs. This, of course, is distinct from optimization techniques that are used to develop a design based upon known parameters (e.g., linear, non-linear, and integer programming techniques). Therefore, when optimization is necessitated there is often feedback to the Requirements Development and/or Synthesis activities. Derivation refers to those situations where lower-level design must be done in order to provide the needed confidence level of the architecture at the level above. Correction covers that broad category of issues that arise as a result of errors.

6.2 Always try re-allocation before redesign. It is usually less expensive to reallocate resources than it is to redesign.

6.3 The cost of rework increases exponentially with time. It is relatively easy to modify a specification. It is more difficult to modify a design. It is still more difficult to modify several levels of design and decomposition. It is yet more difficult to modify hardware when it is in manufacturing, still harder during integration and test, still harder once deployed, and so on.

VII. Reviews

7.1 A central purpose of a Major Milestone Review is to stabilize the design at the commensurate level of the system hierarchy. At the first major milestone review, for example, the primary objective ought to be to stabilize the system level architecture and the lower-level requirements derived from it. This facilitates proceeding to the next level of design with an acceptable level of risk.

VIII. Metrics

8.1 A metric's "coefficient of elusivity" is proportional to the definition resolution of the process it is supposed to measure. The more accurately a process is defined and implemented, the more easily it can be accurately measured. This is a significant contributor to the difficulty of defining useful system engineering metrics. Lack of a sufficiently detailed and universal process makes universally applicable metrics difficult to define.

©2000 CRC Press LLC

IX. Twenty "Cs" to Consider

9.1 Control — Who's minding the store and how?
9.2 Context — How does this system fit into the next larger system?
9.3 Commonality — Can it be the same for all?
9.4 Consensus — Does the customer agree with our interpretation?
9.5 Creativity — Have all the creative solutions been considered?
9.6 Compromise — Are all system parameters properly balanced?
9.7 Change Control — Are all system impacts understood?
9.8 Configuration Management — Is everyone working to the current configuration?
9.9 Comprehension — Is the system understood in terms of what it must do and how it works?
9.10 Characterization — Has the required system functionality, performance, and rationale been defined and communicated?
9.11 Coherence — Does the system function as an integrated whole?
9.12 Consistency — Have all conflicts been resolved?
9.13 Completeness — Has the system been fully defined?
9.14 Clarity — Have all ambiguities been removed?
9.15 Communication — Is there good communication between all stakeholders?
9.16 Continuity — Will successors know why it was done this way?
9.17 Cost Effectiveness — Is the system over-designed?
9.18 Competitiveness — Can it be done better, faster, and/or cheaper?
9.19 Compliance — Does the system do what it is required to do?
9.20 Conscience — Have we done our best?

X. Suggestions for Implementation In Industry

Industry has been plagued with the "process *du jour* syndrome" for a number of years. As a result, there is significant reluctance on the part of many engineers to support yet another three-lettered process to fix all program problems. These attitudes often have merit. Many of these three-lettered processes are not worth the paper on which they are written.

A different tack to implementing this structured approach to complex system development in industry is suggested. First, implementation of the SDF is "tailored" to the specific application by identifying up front what the required inputs and outputs will be for each SDF activity. This can be accomplished with the worksheet, provided in Appendix A, which provides a framework for identifying outputs as well as release status and risk assessment. Second, Exit Criteria is developed for each Major Milestone Review. These criteria are derived directly from the outputs identified in Chapter 5. The required fidelity or "completeness" of each output is defined as a function of time along the program timeline. In addition, the purpose for each major review is clearly defined, as discussed in Chapter 6. Appendix C provides an example of Exit Criteria for a typical Major Milestone Review derived from the outputs of the SDF.

©2000 CRC Press LLC

For each Major Milestone Review, the program must produce the generalized output defined in Chapter 5. In so doing, the structured approach is followed by default. There are several reasons for moving the implementation strategy in this direction. While SDF training courses have been, in general, very well received by engineers in industry, there is still a significant element of the "Don't tell me how to do my job" attitude. This is despite the fact that even a cursory evaluation of Chapter 5 must acknowledge that it is only a framework that is constructed. A second reason for this approach is that the outputs required are generally necessary. Most would agree that the outputs identified are a necessary element of most any development program. How the outputs are generated is not prescribed, nor is the format in which they must be presented dictated. It is simply required that they be completed and presented at the Major Milestone Reviews.

©2000 CRC Press LLC