# chapter 5

# System Development Framework — Technical

*It is not the critic who counts, not the man who points out how the strong man stumbled, or where the doer of deeds could have done them better. The credit belongs to the man who is actually in the arena; whose face is marred by dust and sweat and blood; who strives valiantly; who errs and comes short again and again; who knows the great enthusiasms, the great devotions, and spends himself in a worthy cause; who, at the best, knows in the end the triumph of high achievement; and who, at worst, if he fails, at least fails while daring greatly, so that his place shall never be with those cold and timid souls who know neither victory nor defeat.*
*—Theodore Roosevelt*

In this chapter, a detailed decomposition of the System Development Framework (SDF) building block is developed, which was discussed in the preceding chapters. An example, loosely drawn from a conceptual spacecraft study done in the early 1990s, is included in order to clarify the application of the SDF. The primary focus of the example is functional analysis and decomposition. The spacecraft mission from which this example was drawn was the operation of an astronomical telescope in a low earth orbit. The example is simplified in order to keep attention focused on explaining the application of the SDF to a real development activity. In order to maintain clarity of the SDF discussion, the example will be applied at the end of each section. This spacecraft system will be referred to as Example Sat or ESAT for short. The task for the ESAT program is to develop the spacecraft bus — that portion of the satellite that supports the telescope in space.

As discussed in Chapter 1, the term "system" has been defined as "any entity within prescribed boundaries that performs work on an input in order to generate an output." Using this definition, it is asserted that at any level of the development hierarchy there exist one or more "systems." At the level below the top level, or system level, these are commonly called subsystems.

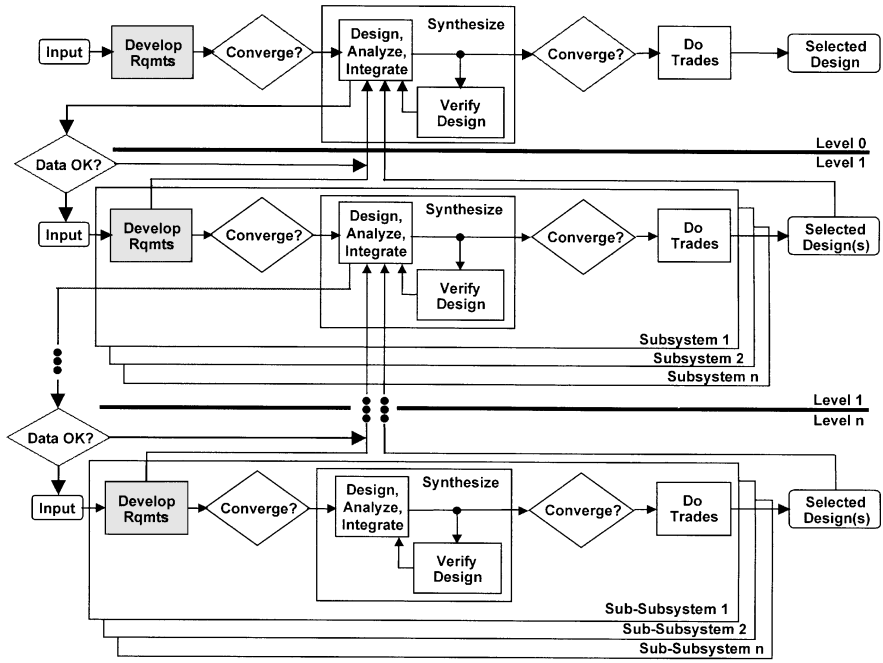



***Figure 5.1*** "Develop Requirements" in the System Hierarchy.

Each of these subsystem development activities is perpetuated as its own "system," applying the structured approach herein described as the SDF. Therefore, it is further asserted that while the specific inputs and outputs may change with hierarchical levels, the general activities delineated in this chapter do not change at each level. Thus this process is applicable to all levels of the hierarchy. In the discussion below, it is assumed that each of the activities described is applied to each development activity at each respective hierarchical level.

In order to keep the following discussion in the context of the total SDF, Figures 2.2 and 3.2 from the preceding chapters will be used in various sections to highlight the activity under discussion.

## *I. Develop Requirements — Determine "What" the System Must Do*

Figure 5.1 highlights the "Develop Requirements" activity at each level in the system hierarchy. This serves to illustrate that the same basic process is applied to each system element at each hierarchical level.

The Requirements Development set of activities addresses the question, "***What*** must the system do?" These activities include:

- Collect and analyze imposed requirements
- Derive requirements through context analysis, functional analysis, design, allocation, and decomposition
- Manage requirements derived during the development process
- Communicate requirements and requirements changes
- Determine and track how and where in the system build-up the requirements will be verified
- Achieve customer consensus regarding interpretation of the customer-imposed requirements
- Maintain traceability of requirements

It is important to maintain traceability of requirements throughout the development for several reasons:

- Cost Minimization — Avoid over-design; that is, adding cost by including functionality that is not necessary; or under-design, i.e., not providing functionality that is required by the customer
- Cycle Time Reduction — Facilitates a coordinated effort that "does it right the first time"
- Change Impact Analyses — Provides a logical and systematic approach to assessing the impacts of changes to the design
- Customer Requirement — Many customers require demonstrated traceability
- Consistency, Clarity, and Completeness Ensured — Early detection and correction of requirements issues

Figure 5.2 illustrates the decomposition of the Requirements Development (RD) activity, derived in Chapter 2, Figure 2.2. The RD activity is organized as a rework cycle as discussed in Chapter 4. It comprises two work generation activities: "Derive Context Requirements" and "Generate Functional Description," as well as two rework discovery activities: "Analyze Requirements" and "Analyze Functional Description."

As the RD activity progresses, it is continually or periodically assessed for convergence. Convergence occurs when the design team is able to reach a reasonable solution with the input data. There are several indicators that can be employed to assess if convergence is occurring: monitoring key Technical Performance Measures (e.g., technical budgets, remaining margin on key parameters), program risk assessments, various technical analyses, estimates to complete, and periodic design reviews and audits. If convergence is not occurring at an acceptable rate, changes may be required to enhance the quality of the work performed, and/or to accelerate the rework discovery activities.

RD rework discovered elsewhere in the process may feed back to the RD activity as "work to do" or as issues to be addressed with the customer who generated the input requirements. This is indicated by the "discovered

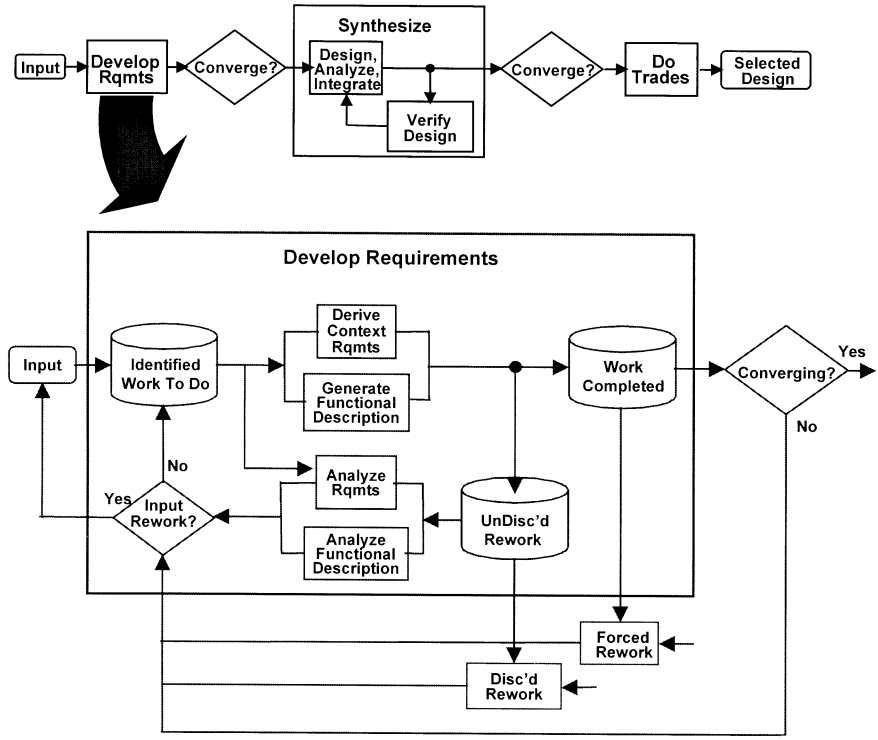



***Figure 5.2*** The "Develop Requirements" Activity Decomposed.

rework" box. Rework discovered in other areas may be the result of difficulty implementing the requirements as defined in the functional description. This may force work previously defined as complete to be redone. This is indicated by the "forced rework" box.

## *A. Inputs*

Requirements originate from many sources in varying forms, both explicit and implicit. These include technical, cost, and schedule concerns. All requirements must be considered to maximize success. The following is a nonexhaustive list of potential requirement sources that ought to be considered.

- Immediate customer
- The division
- Business development
- Subcontractors
- Procuring organization
- The corporation

- Heritage designs
- New technology
- User community
- The department
- Competitors

Some of the properties that compose a good requirement include:

- Clarity — unambiguous
- Consistency — no mutually exclusive or conflicting requirements
- Completeness — provides all necessary information
- Verifiability/Quantifiability — compliance demonstrable
- Traceability — necessity demonstrable
- Functionally oriented — maximizes design creativity/flexibility

The requirements management discipline has a nomenclature of its own. Some of the key terms are:

- Parent — A requirement from which other requirements have been derived
- Child — A requirement derived from a parent
- Orphan — A non-top-level requirement having no identified parent (otherwise described as a problem child)

Each of these are illustrated in Figure 5.3.

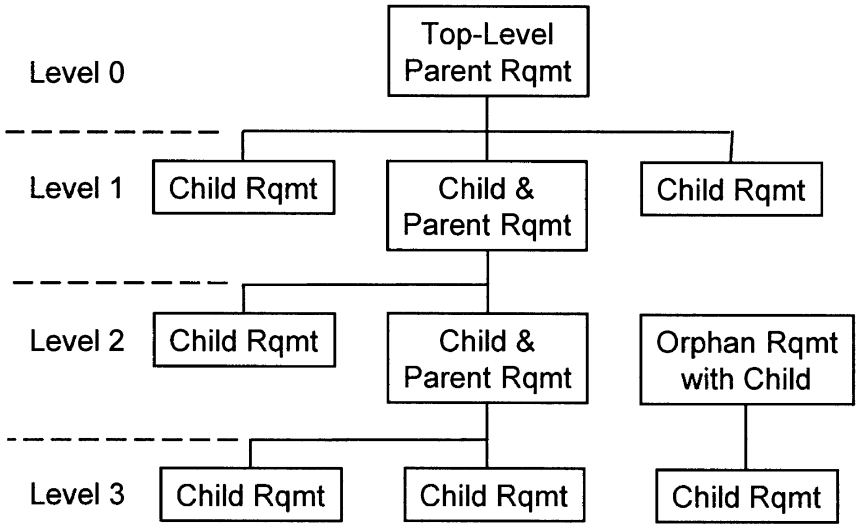



***Figure 5.3*** Requirements Relationships.

Other important terms in a requirements document include: shall, will, and may or should. In most contexts, employment of the term "shall" indicates that there is no flexibility in terms of the design providing that particular function and that function performing according to the specified level. Where there is some flexibility regarding the customer's expectations, other words are generally used such as "will," "may," or "should." Therefore, it is important for all the stakeholders to define these terms up front and to use them consistently so that any trade-offs can be performed according to the right priorities.

As discussed above, requirements originate from many different sources. Table 5.1 provides an abbreviated listing of some of the initial requirements defined by the customer at the beginning stages of the ESAT conceptual study. Some of these were recorded in presentation packages provided by the customer, others were mentioned in telephone or other informal conversations.[41] This is not unusual and it is important, even at this early stage in the program, to maintain traceability. Therefore, as shown below, the source of the requirement is recorded with each requirement.

## B. *Work Generation Activities*

Figure 5.4 highlights the Work Generation activities that are performed within the Develop Requirements activity.

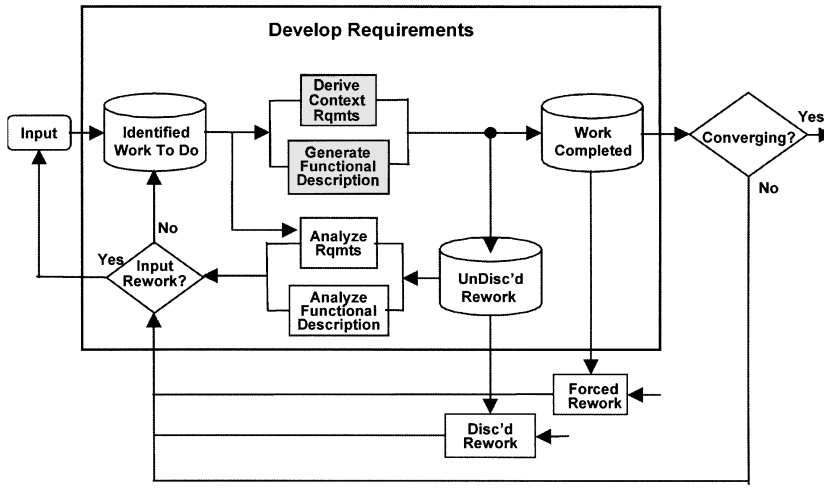



***Figure 5.4*** "Develop Requirements" Work Generation Activities.

### 1. *Derive Context Requirements*

The focus of this activity is to determine context in which the system must function over its complete life cycle. This is accomplished by:

[41] Most of these requirements are taken from the study; some are fabricated to facilitate the usefulness of this example.

*Table 5.1* ESAT Customer-Imposed Requirements Set

| No. | Title | Text | Source |
|---|---|---|---|
| **1.0** | **General Program** | | |
| 1.1 | Launch Date | The spacecraft shall support a launch date of October 1998 | Presentation Package |
| 1.2 | Operational Orbit | The operational orbit shall be 800 Km altitude, 28 degrees inclination | Presentation Package |
| 1.3 | Mission Life | The spacecraft bus shall provide full functionality for a minimum of 2 full years' operation on orbit after initialization | Presentation Package |
| **2.0** | **Space Segment** | | |
| 2.1 | Payload Instrument | | |
| 2.1.1 | Fine Star Sensor (FSS) Accuracy | The instrument shall provide FSS data to the spacecraft bus with an accuracy of 1/2 Hz. 0.33 arcsec (1 sigma) and a 20 arcmin field of view | Presentation Package |
| 2.1.2 | Instrument Mass | The instrument mass shall not exceed 1500 pounds mass | Presentation Package |
| 2.2 | Spacecraft Bus | | |
| 2.2.1 | Instrument Data Interface | The spacecraft bus shall provide a 4 to 300 kbps data interface | Presentation Package |
| 2.2.2 | Slew Rate | The spacecraft shall be able to slew 90 degrees within 45 minutes of initialization | Presentation Package |
| 2.2.3 | Contamination | The cleanliness level 500A shall be maintained during integration and test of the system | Presentation Package |
| 2.2.4 | Command and Telemetry Interface | The spacecraft shall provide MIL-STD-1553 and MIL-STD-1773 command and telemetry data bus interfaces | Presentation Package |
| 2.2.5 | On-Board Data Storage | A minimum of 100 megabytes of data storage shall be provided by the spacecraft | Telecon w/ Program Manager |
| 2.2.6 | Mechanical Interface | The spacecraft mechanical interface shall be a 4 point attachment on a 48 inch bolt circle | Presentation Package |
| 2.2.7 | Electrical Power | The spacecraft bus shall be capable of providing up to 300 watts of power at 28 +/- 7 v End Of Life | Telecon w/ Program Manager |

*Table 5.1 (continued)* ESAT Customer-Imposed Requirements Set

| No. | Title | Text | Source |
|---|---|---|---|
| 2.2.8 | Attitude Control | The spacecraft shall point the telescope to an accuracy of ±0.01o on all three axes | Presentation Package |
| **3.0** | **Launch Segment** | | |
| 3.1 | Spacecraft Bus Volume | The spacecraft bus volume shall not exceed 108 inches in diameter and 36 inches height above the separation plane | Presentation Package |
| 3.2 | Total Deliverable Mass | The maximum deliverable mass to the operational orbit shall be 3000 pounds | Presentation Package |
| 3.3 | Fairing Volume | The maximum fairing volume shall be 108 inch diameter, TBD inches height | Presentation Package |
| 3.4 | Minimum First Mode | The minimum first mode shall be 12 Hz | Presentation Package |
| **4.0** | **Ground Segment** | | |
| 4.1 | Antenna Configuration | The ground system antenna shall be a dichroic design, 5 meters in diameter | Telecon w/ Program Manager |

- Identifying all mission phases, modes, and states
- Identifying and characterizing all external interfaces, by mission phase
- Defining the environments to which the system will be subjected, by mission phase
- Identifying critical issues by mission phase (events, technologies, etc.)
- Developing the concept of operations

**Output**

- Specification(s)
- Operations Concept
- Context Diagram(s)
- Entity Relation Diagram(s)
- Event List(s)
- External ICDs

Figure 5.5 illustrates some of the key parameters that define the context within which the system must function over its life cycle. The various phases of the program are identified as columns ranging from "womb-to-tomb." Key parameters are identified as rows in the matrix. This provides an organized framework to begin deriving context requirements.

| Parameters | Mission Phases | | | | |
|---|---|---|---|---|---|
| | Integration & Test | Deploy | Initialization | Operations | Disposal |
| External Interfaces | Test Fixtures | Launcher Ground Sys | Launcher Ground Sys AKM | Ground Sys Relay Sats Other Sats | Ground System |
| Environment | Clean Room System Test | Air Ride Van Air Transport Launch site Facilities Fairing | Ascent Traj | Operational Orbit | Parking Orbit or Earth Re-Entry |
| System Modes | Test | Test Launch mode | On-Orbit test Maneuver Appendage Deploy | Nominal Standby Safe Maintenance On-Orbit test | De-Orbit |

*Figure 5.5* ESAT Mission/Context Definition

**Key Point**

It is necessary to consider all phases of the program early in the development process because each phase may impose unique requirements to the system. During manufacturing, assembly, and test activities or integration and test activities, for example, special interfaces may be required. This analysis should also expose incompatibilities among certain interfaces. The earlier in the design process that these things can be addressed, the higher the probability the system will be successful.

*2. Generate Functional Description*

As requirements are developed, the Functional Analysis activity seeks to arrange the functions into a coherent system. This can be done in several ways, one of which is computer simulation. A key goal is to ensure that there are no mutually exclusive or conflicting requirements.

This activity generates the specification of the system. A key concern here is proper protocols, and timing of input and output data. Outputs include:

- Identification of all functional requirements flowing out of imposed and derived requirements
- Development of the specification(s)
- Determination of performance requirements of each function and the relationships (interfaces, interdependencies, etc.) between functions

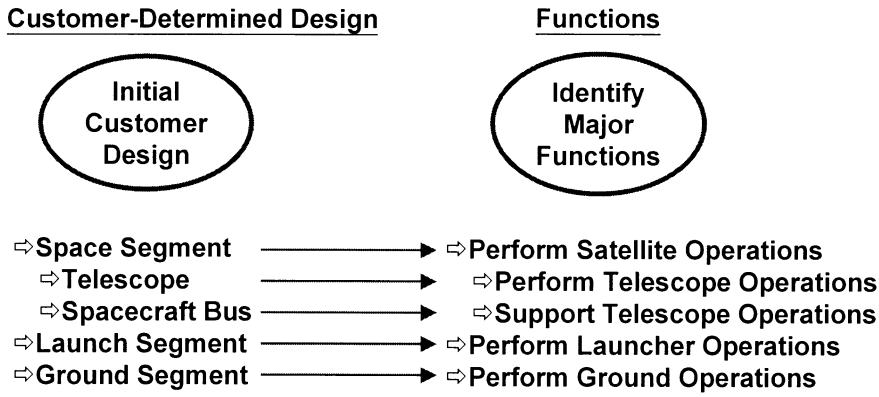



*Figure 5.6* Mapping Selected Implementation to Functions.

**Key Point**

> The way in which the customer defined his system-level requirements reveals his selection of a particular implementation of the system. He has determined that the telescope requires a dedicated spacecraft bus to support it; that a particular launch vehicle will be required; and that a particular ground station configuration will be used.

Figure 5.6 illustrates the mapping of the customer-defined system-level implementation of the program. Each major segment identified in the requirements corresponds to a function that must be performed by the system.

Figure 5.6 illustrates the primary functions that must be performed from a top-level system perspective during the launch and orbit acquisition phase of the mission. Notice that each function description begins with a verb. This is important because it emphasizes the essence of a function — it is something the system must do.

**Key Point**

> A functional description is not primarily concerned with defining "how" the system ought to be designed. Its purpose is to describe "what" the system must do. Such definition facilitates new ways of implementing systems — thinking "out of the box," which nourishes an environment conducive to design breakthroughs.

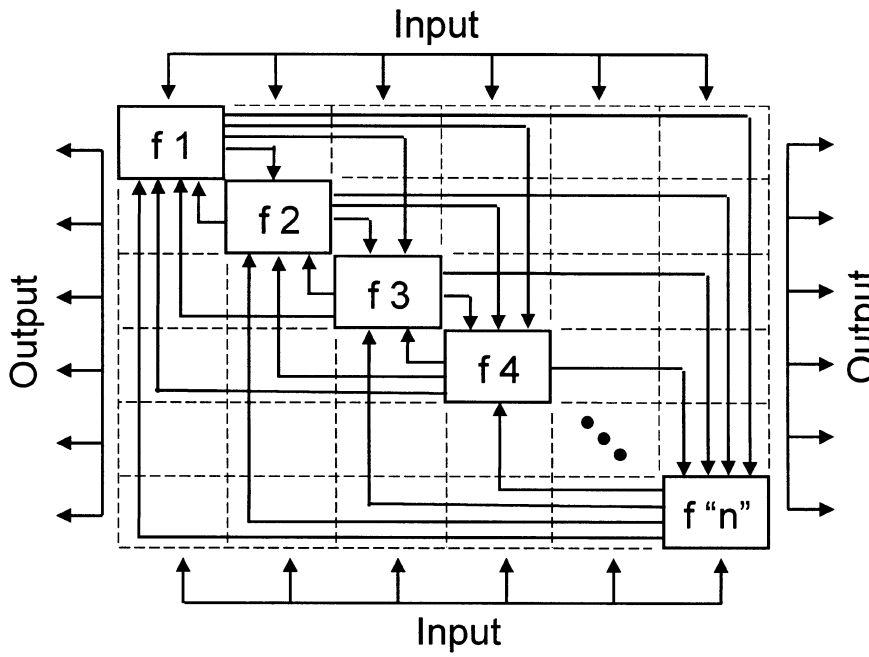



*Figure 5.7* The N² Diagram.

As the system is developed, the N² Diagram format will be loosely followed.[42] Figure 5.7 provides an illustration of the N² format, which is a convenient way of developing interfaces between system elements, whether functions or implementation. System elements are placed along the diagonal, inputs and outputs are indicated on the outer-sides of the chart, and interfaces are defined as shown.

During the Orbit Acquisition Phase, the ESAT system comprises three major segments: ground, space, and launch. Figure 5.8 illustrates the derivation of three major system functions from these major pieces of the system: Perform Satellite Operations, Perform Launcher Operations, and Perform Ground Operations. In keeping with the N² format, the functions are arranged diagonally with interfaces, inputs, and outputs described as shown.

Figure 5.9 illustrates the importance of defining system functions for each distinct phase. While both Figure 5.8 and Figure 5.9 describe the ESAT program at the same system level, they are quite different. Figure 5.9 does not have the same functions or interfaces as those shown in Figure 5.8. During the Launch Phase, the Launch Operations function is a major function with critical interfaces identified. Obviously, after the spacecraft achieves its operational orbit, the Launch Operations function is no longer required.

[42] Cf. the *Systems Engineering Management Guide*, January 1990, Section 6.3.2 for a discussion of the N² Diagram. As noted in that reference, it was developed by TRW and is described in "The N² Chart," R. Lano, Copyright 1977 TRW Inc.

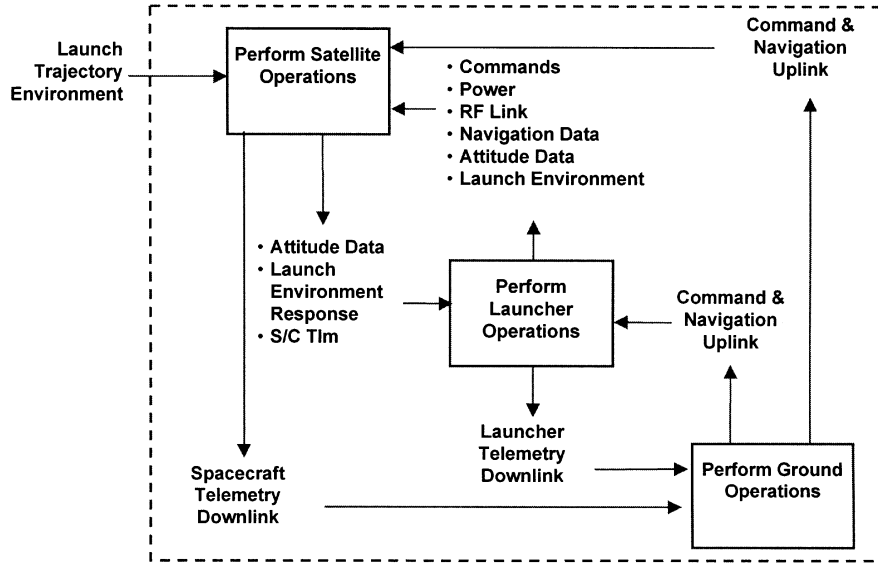



*Figure 5.8* ESAT System-level Functional Block Diagram — Orbit Acquisition Phase.

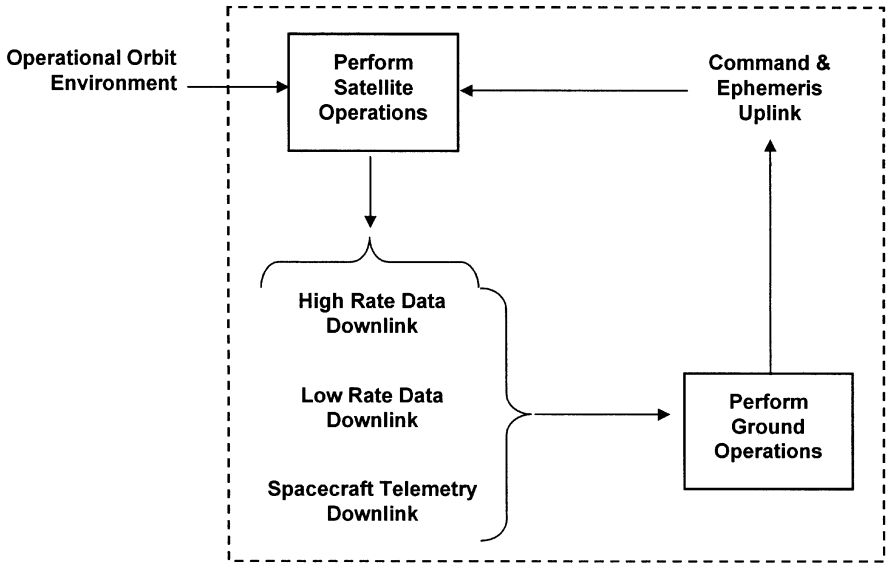



*Figure 5.9* Operations Phase.

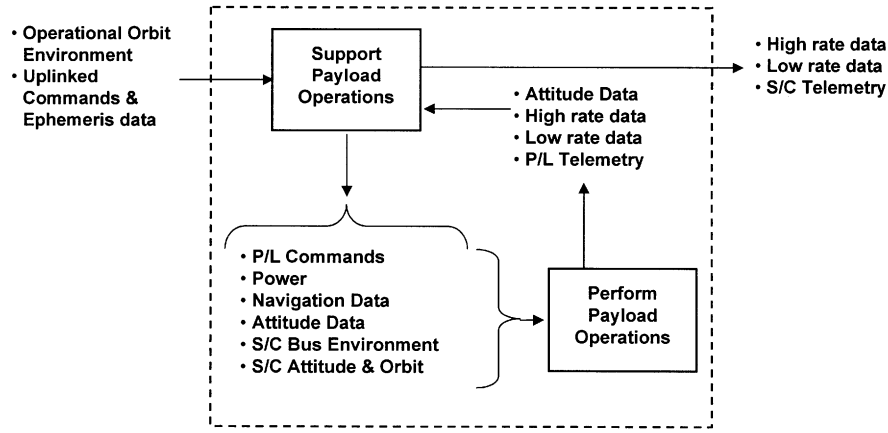



*Figure 5.10* "Perform Satellite Operations" Function Decomposition.

**Key Point**

> This is a simple example, but the point is important — the implementation required at each level of the hierarchy may be very different for each phase of the life of the system. Therefore, system functions must be defined for each mission phase.

As discussed in the preceding chapters, requirements come first, then functions are identified from those requirements, and finally implementations are developed that provide the necessary functionality at the required performance level. By the way the customer defined the initial requirements set, it is apparent that he or she has conceptualized a design in which a spacecraft bus will support the telescope. It is this knowledge about how the system is to be implemented that enables the decomposition of the "Perform Satellite Operations" function into two sub-functions: "Perform Payload Operations" and "Support Payload Operations." The latter function, of course, is implemented by the spacecraft bus, which is the focus of this example development activity. This decomposition is depicted in Figure 5.10.

Notice that the inputs (operational orbit environment, and uplinked commands and ephemeris data) and outputs (high rate data, low rate data, and S/C telemetry) defined for the "Perform Satellite Operations" function in Figure 5.9 are still present in its decomposition depicted in Figure 5.10. However, they are applied more specifically to the "Support Payload Operations" function in the decomposition.

Figure 5.10 also shows the development of the interfaces between the two functions identified in the decomposition. As defined in the requirements in Table 5.1, the payload requires commands, electrical power, navigation data, attitude data, a controlled physical environment, and a particular orbit

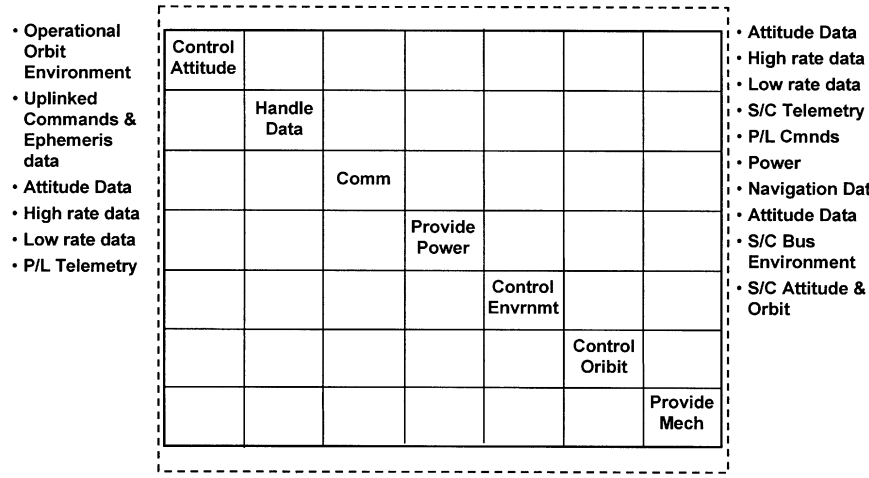



***Figure 5.11*** Support Payload Operations Decomposition.

and attitude within that orbit. In addition, the telescope must provide data to the spacecraft bus. Thus attitude data, high and low rate data, and telescope telemetry must be provided to the spacecraft bus.

As the decomposition of the "Support Payload Operations" function shown in Figure 5.11 indicates, the spacecraft bus must provide functions that perform the tasks required by the requirements set. Applying the same $N^2$ methodology again, each function is placed along the diagonal of the matrix. The matrix guides the design team to determine which, if any, interfaces are required between the various functions included in the decomposition.

Figure 5.12 illustrates the continuity between the various levels of decomposition. Also indicated is the implementation assumed that enabled each decomposition. The design assumption that enabled the decomposition of the "Perform Mission Operations" function was the concept of a separate spacecraft bus. This enabled a decomposition to two main functions at the next level down: "Support Payload Operations" and "Perform Payload Operations." The "Support Payload Operations" function, which is the spacecraft bus itself, was decomposed by assuming that the standard spacecraft bus subsystems would be implemented. These functions are identified along the diagonal, according to the $N^2$ format.

**Operations Concept** — The generation of the Operations Concept is initiated during the development of the functional description of the system. It typically describes how this system fits within the overall program in terms of specific capabilities and functions provided. It discusses management of the system during all operational modes for each mission phase and how the data is handled and generated by the system. This would include processing, storage, and distribution of the data. Other issues such as program organization, specific types of personnel and job functions necessary, equipment, and training are also addressed.

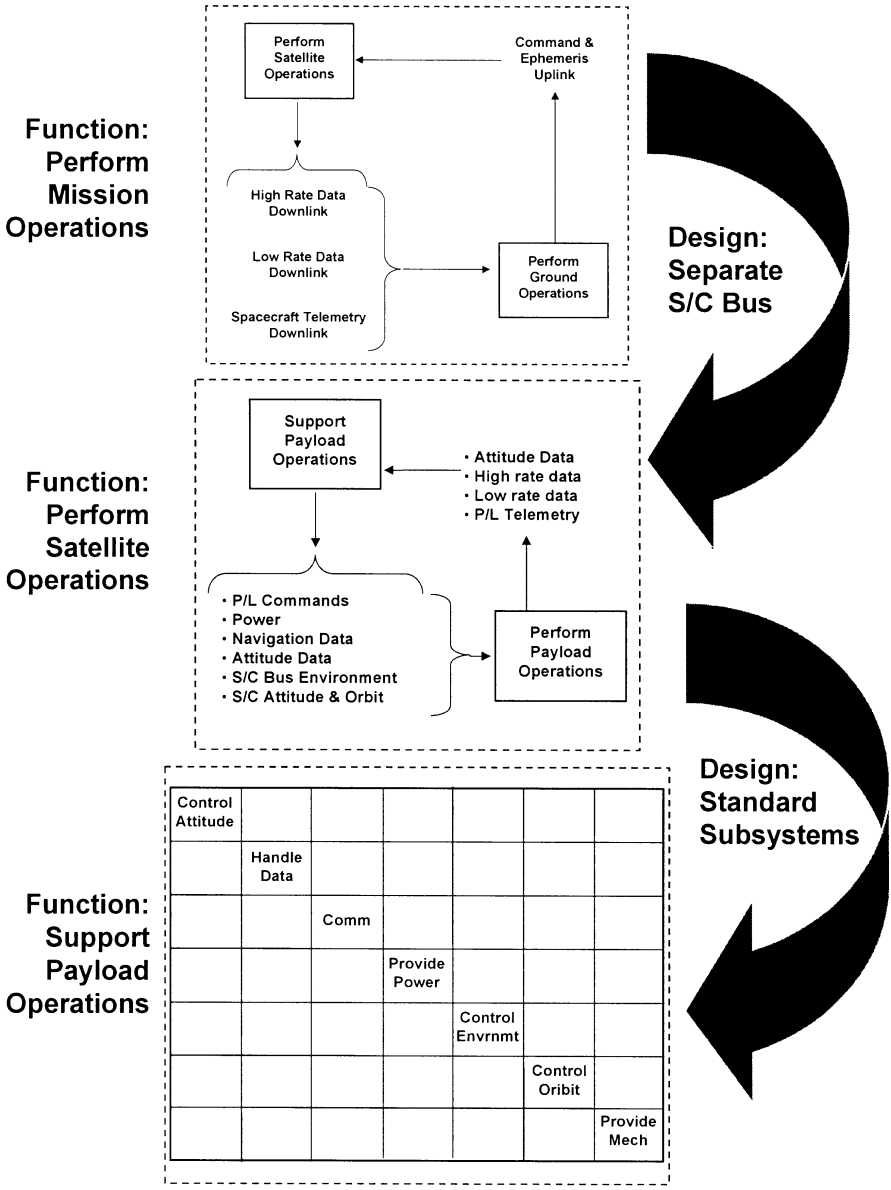



*Figure 5.12* Decomposition Continuity.

**Output**

Specification(s)
Functional models (block diagrams, flow diagrams, behavior diagrams, simulations)

### 3. *Digression: Why Functional Analysis?*

Before proceeding to the discussion concerning Rework Discovery activities, the question "Why spend precious program resources performing functional analysis and functional decomposition?" is addressed.

**Competitiveness** — One important reason is competitiveness. A customer is generally more concerned with obtaining the functionality and performance levels desired than with the way in which that functionality is implemented. This, of course, assumes all else being equal, such as reliability issues. If a company is to remain competitive in an environment where technology is rapidly changing, it must focus on the functionality it is providing to its customers and not become overly enamored with its particular implementation or design. As an example, consider the slide rule. Since its invention in the early 1600s, it remained an important tool in the hands of scientists and engineers even well into the "space age." Many a slide rule was used in the design of the Space Shuttle. It was pervasive into the 1980s, but by the 1990s slide rules were little more than collector's items. What happened? More to the point of this discussion, what happened to the companies that manufactured them?

The slide rule was a calculator. Users purchased them for their functionality — performing mathematical calculations. Users were not so much concerned with the quality of the ivory and the fineness of the scales as they were with being able to perform calculations quickly, easily, and accurately. This was proven when the electronic calculator came on the scene. Within about a decade slide rules were a thing of the past. How many companies engaged in the manufacture of slide rules jumped into the electronic calculator market? Could it be that if those companies had understood their core competency as providing the functionality needed to perform mathematical calculations, they would have vigorously pursued technologies that better perform those functions? This is the danger with thinking primarily in terms of a particular design or implementation. An organization can become so consumed with its own method of providing a particular set of functions that it cannot leverage new technologies that might better perform that functionality. This leaves such a company vulnerable to competition that might be more agile. How much more is this true in those markets where technology is changing and advancing at unprecedented rates?

**Specification Development** — Specifications should focus on functionality and performance, not implementation. This gives the experts, those engineers receiving and responding to the specification, the flexibility to design an optimal solution. Presumably they understand the pertinent technologies and are therefore in the best position to develop an optimal design.

**Exploitation of New Tools** — Also, there are an increasing number of tools available to the designer to simulate functionality and performance. Among other things, this provides a means for validating a specification —

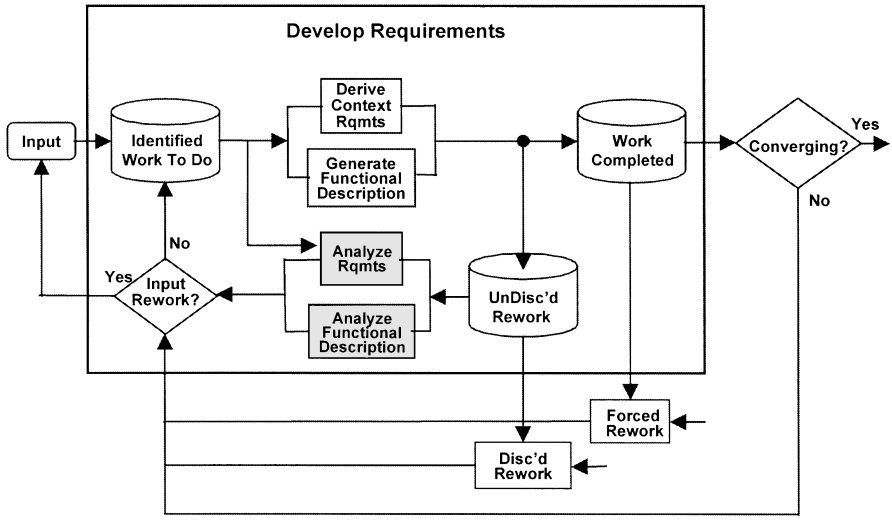



*Figure 5.13* "Develop Requirements" Rework Discovery Activities.

ensuring that it is self-consistent and complete before committing design resources to what could be faulty input data.

**Focus Research and Development (R&D) Efforts** — Finally, if functionality and performance can be identified for future systems, research and development (R&D) efforts can be directed toward developing designs that provide them. In this way, direction can be given to R&D efforts in terms of identifying where resources should be directed in the development of new core competencies with the highest leverage for the organization.

## C. *Rework Discovery Activities*

Figure 5.13 highlights the Rework Discovery activities that are performed within the Develop Requirements activity.

### 1. *Analyze Requirements*

This activity determines the validity of both the imposed and derived requirements. The goal is to ensure that the requirements are complete, self-consistent, unambiguous, and verifiable or measurable. In addition, it must be determined how the requirements will be verified and at what level verification will take place in the system build-up.

The task of analyzing the requirements set for problems is obviously a Rework Discovery activity. In terms of the logic of the overall SDF, Rework Discovery activities follow Work Generation activities — data must be generated before it can be analyzed for potential rework. Nevertheless, this task should be performed whenever new or changed requirements are input to

the development activity. This is illustrated in Figure 5.13 by the arrow from the "identified work to do" bucket, indicating work is flowing not only to the Work Generation activities, but also directly to the Requirements Analysis activity. Because the sooner rework is discovered the less its potential impact to the program, input requirements are analyzed as soon as they are introduced.

**Output**

- Identification of all "To Be Determined" (TBD) holes in the requirements, with a closure plan
- Identification of conflicting or inconsistent requirements, with a closure plan
- Interpretation of vague or ambiguous requirements in order to review them with the customer and gain consensus
- Determination of the verification method (test, analysis, demonstration, simulation, inspection) that will be used for each requirement
- Determination of where in the system build-up each requirement will be verified
- Implementation of Configuration Management activities

2. *Analyze Functional Description*

The main task is to develop and validate the functional description of the system before resources are spent developing the design. This may involve simulation of the identified functions with their respective interfaces in order to verity that the specification is valid in terms of self-consistency. The main activities here include:

- Determination if the specifications are complete and self-consistent
- Identification of all functional requirements flowing out of imposed and derived requirements
- Determination of performance requirements of each function and the relationships (interfaces, interdependencies, etc.) between functions

**Output**

Validated specification(s)
Functional models (block diagrams, flow diagrams, behavior diagrams, simulations)

**Key Point**

> It should be noted here specifically that functional decomposition does not occur in the Requirements Development activity. A function cannot be decomposed

> without some knowledge and/or assumption regarding how it might be implemented.[43]

In the preceding discussion, several decompositions were developed. The first showed the entire ESAT system which included the space segment, ground segment, and launch segment, then the system was decomposed down to the spacecraft bus level. During the process, each assumed implementation that facilitated each decomposition was pointed out. In this way, the principle that functional decomposition cannot be performed apart from some knowledge or assumption about the implementation was reinforced. This discussion was presented in the context of the Requirements Development activity in order to show how the customer-imposed requirements for the spacecraft bus were derived by the customer before the Spacecraft Bus Development activity was initiated. This in no way implies that functional decomposition is performed in the Requirements Development activity. Rather, this discussion sought to emphasize the fact that functional decomposition is performed under the assumption of a particular implementation — which is developed in the Synthesis activity. Functional decomposition follows definition of the "how," which is developed in the Synthesis activity.

Other authors assert similar ideas. For example, Hatley and Pirbhai conclude:

> [H]igher levels of the system always provide the requirements for the lower levels. For systems containing hardware and software, this means that we need to know system-level requirements, decide on the system-level architecture, and then decide on the allocation of system requirements to hardware and software before we can establish the software requirements.[44]

In another portion of the book discussing similar issues, they note:

> This leveled repetition of functional requirements definition, followed by physical allocation, is fundamental to the nature of large systems development.[45]

[43] The author does not offer a formal proof of this sometimes debated point. However, to argue from practical experience, this author is not aware of any credible example of a functional decomposition, of either hardware or software, that has been performed without some reference to a design or design concept.

[44] Hatley, Derek J. and Imtiaz A. Pirbhai, *Strategies For Real Time System Specification*, New York: Dorset House, 1988, p. 264.

[45] Ibid., p. 7.

Similarly, Professor Nam Suh states:

> There are two very important facts about design and the design process, which should be recognized by all designers:
>
> 1. FRs [Functional Requirements] and DPs [Design Parameters, i.e., implementation] have hierarchies, and they can be decomposed.
> 2. FRs at the $i^{th}$ level cannot be decomposed into the next level of the FR hierarchy without first going over to the physical domain and developing a solution that satisfies the $i^{th}$ level FRs with all the corresponding DPs. That is, we have to travel back and forth between the functional domain and the physical domain in developing the FR and DP hierarchies.[46]

Functions and their respective implementation are intimately intertwined and necessarily dependent. Therefore, it is not possible to specify requirements that are independent from implementation in any absolute sense.[47] This has implications regarding the development of specifications in a multileveled hierarchy.

**Key Point**

- First, a specification is directly coupled to the implementation at the level above. It is therefore not implementation independent and cannot be so. It is incorrect to hold the notion that a specification can be written with no reference to implementation.
- Second, the System Development process cannot replace, nor is it intended to replace, technical expertise. In fact, because of the necessary connection between requirements and implementation, the SDF cannot be effectively applied without significant technical expertise.
- Third, a change in the functional description above will likely necessitate a change in the dependent implementation adjacent to it. Likewise, a change in the implementation above will likely necessitate a change in the dependent functional description of the system(s) below it (cf. Figure 5.30).

**Output** → Functional Description

[46] Suh, Nam P., *The Principles of Design*, New York: Oxford University Press, 1990, p. 36.
[47] This is in contrast to those who emphasize the necessity of specifying requirements in such a way that they are independent from implementation.

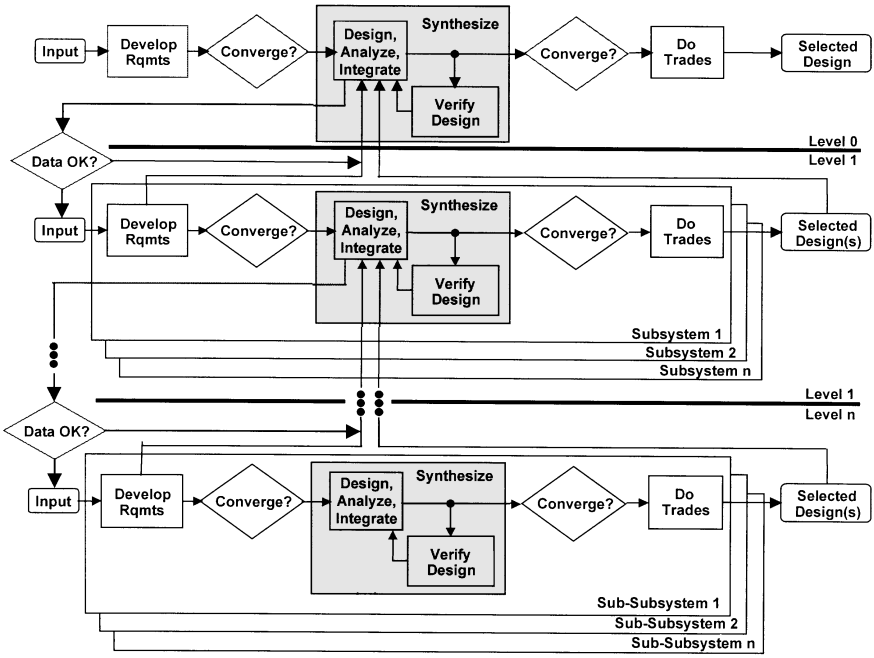



***Figure 5.14*** The "Synthesize" Activity in the System Hierarchy.

**Customer Consensus** — Although customer consensus is critical throughout the system development, because requirements drive the system design, concurrence from the customer community is especially crucial and is therefore specially noted here as an essential component to the Requirements Development activity.

## *II. Synthesis*

The etymology of the word "synthesis" is συν + τιθεναι, "together with" + "to place". Synthesis means, therefore, "to place together with."[48] Synthesis has to do with the integrating of the elements of the solution into a coherent whole. Therefore, subsumed under this primary activity are all the subactivities involved in designing, analyzing, and verifying the system implementation. Figure 5.14 highlights the "Synthesize" activity in the context of the system hierarchy.

Figure 5.15 illustrates the decomposition of the Synthesize activity derived in Chapter 2. It comprises the Work Generation activity "Design, Analyze, and Integrate," and the Rework Discovery activity "Verify Design."

[48] *Merriam-Webster's Collegiate Dictionary, Tenth Edition*, Springfield MA: Merriam-Webster, 1996, p. 1197.

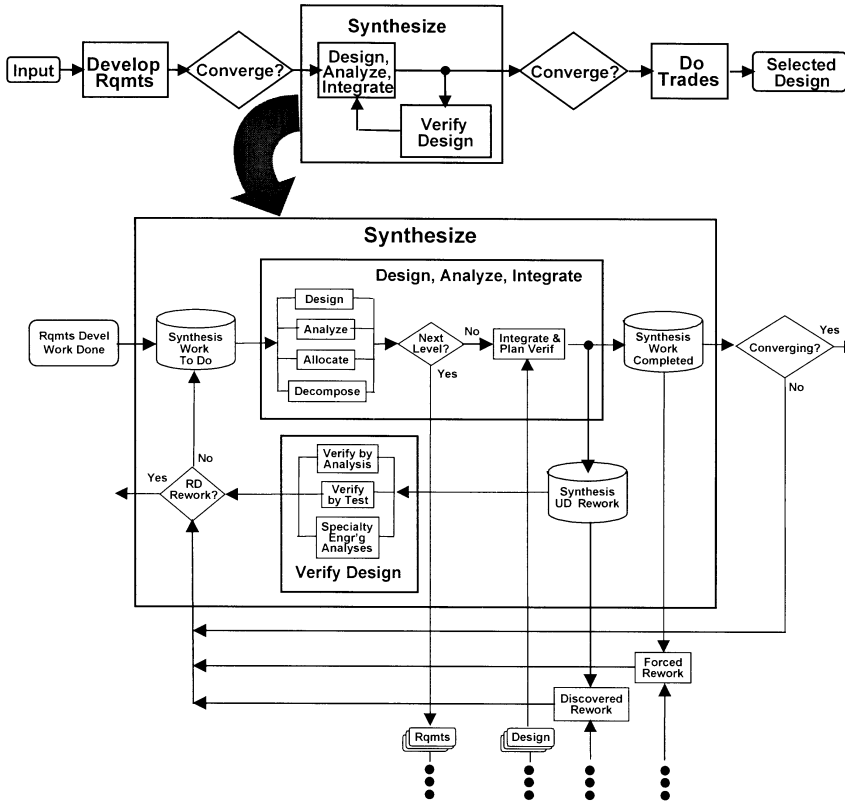



***Figure 5.15*** The "Synthesize" Activity Decomposed.

As mentioned previously, the System Development activity considers not only the development of the deliverable product itself, but also all the associated hardware, software, procedures, processes, etc. needed to produce, integrate, test, deploy, operate, support, and dispose of the system.

## *A. Work Generation Activities: Design and Integration*

**Determine "How" to Implement the "What"** — Figure 5.16 highlights the "Work Generation" activities performed within the Synthesize activity. The activities identified are focused on generating the outputs needed at a particular point on the program timeline. As the program moves forward, the focus shifts from parametric analyses aimed at defining the available design space to detailed solutions in response to the increasingly detailed requirements.

### *1. Design*

The Design activity is narrowly defined here as the set of activities that defines the initial design space, develops concepts or solutions in response

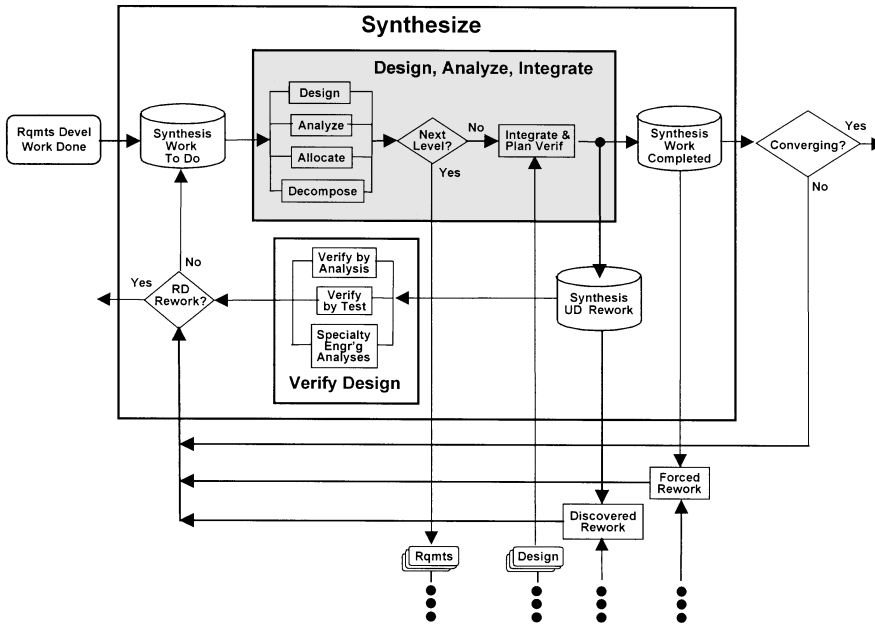



*Figure 5.16* The "Synthesize" Work Generation Activities.

to the requirements, generates the design documentation, and performs analyses needed in the development of the solution.

- Quantify Design Space (H/W and S/W)
  - Parametric analyses
  - New technologies and heritage designs are surveyed for applicability
- Generate Preliminary and Detail Design
  - Block diagrams, schematics, drawings, etc.
  - Internal ICDs
- Risk Management → Identify and Assess Risk
  - Technical performance, cost, schedule
  - Preliminary mitigation approaches
- Configuration management of all design documentation

**Output** → H/W & S/W concept(s) and/or design(s), risk assessment

As indicated above, the first activity performed in the Design activity is to quantify the design space: That is, to define the major system interfaces and environments for each mission phase as well as to quantify critical design parameters in order to understand the cause-and-effect relationships between them. Figure 5.17 defines the top-level implementation of the ESAT system for the Launch and Orbit Acquisition Phase. The three top-level

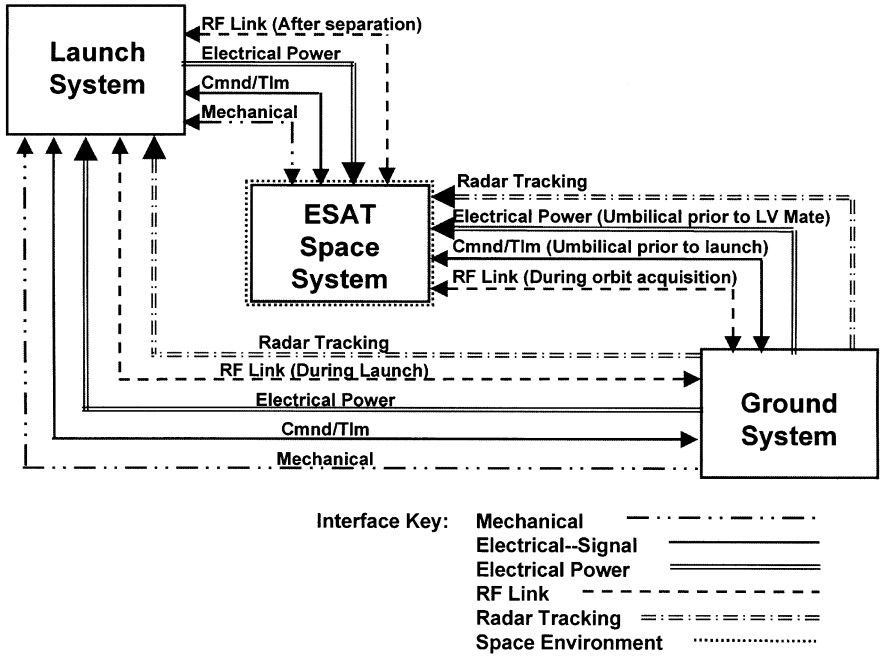



*Figure 5.17* Interfaces — Launch and Orbit Acquisition Phase.

elements of the system are now described as nouns to indicate implementation, instead of verbs which are used to signify functionality. Thus there is direct traceability from the top-level functions to the top-level design or implementation. The major system elements are the ESAT space system, the ground system, and, during the Launch and Orbit Acquisition Phase, the launch system. Also included in Figure 5.17 are the interfaces between the system elements.

Figures 5.18 through 5.26 illustrate some of the various top-level architectures that have been implemented in the past and present, which could be employed as potential solutions for the ESAT spacecraft bus concept. For the ESAT example, the focus will be on the Attitude Determination and Control Subsystem (ADACS) to illustrate functional decomposition to the subsystem level. The attitude control requirements for a spacecraft play a significant role in determining many aspects of the bus design. The figures depict five different spacecraft types — all driven primarily by ADACS requirements. Some spacecraft missions require no attitude control. The Environmental Research Satellite (ERS), illustrated in Figure 5.18, was just such a spacecraft.

Some missions require that only one axis of the spacecraft be pointed toward the earth to a fairly loose tolerance. This can be accomplished by a "gravity gradient" design. The GEOSAT spacecraft, shown in Figure 5.19, took advantage of this concept. It was designed to measure sea surface heights.

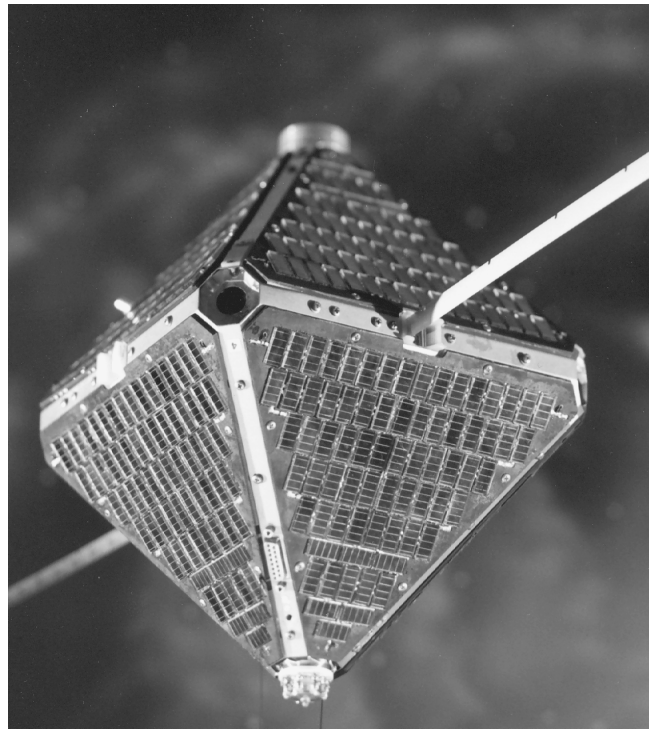


*Figure 5.18* The Environmental Research Satellite. (Photo Courtesy TRW, Inc.)

Other missions require a single axis be pointed toward the earth, but with a higher degree of accuracy. Spin-stabilization is often employed in these situations. The DSP (Defense Support Program) spacecraft — a military early warning system — is one example. Another example is the Television Infrared Observation Satellite (TIROS) II,[49] a meteorological satellite. These are depicted in Figures 5.20 and 5.21 respectively.

For those missions where all three axes must be stabilized, there are two primary designs: bias momentum and zero momentum. The bias momentum design is often used in geo-synchronous missions. One example is NASA's Tracking and Data Relay Satellite System (TDRSS), shown in Figure 5.22.

Especially in low earth orbit remote sensing missions, where a high degree of pointing accuracy is required, the zero-momentum design is often implemented. The Compton Gamma Ray Observatory (CGRO) and the Hubble Space Telescope (HST), shown in Figures 5.23 and 5.24, respectively, are examples.

Figures 5.18 through 5.24 illustrate an important aspect of system design: a thorough examination of existing designs to see if anything already developed might be suitable. This is a good approach because there is inherently less risk in designs that have already been proven, not to mention that the

[49] Newer TIROS satellites are three-axis stabilized, zero-momentum systems.

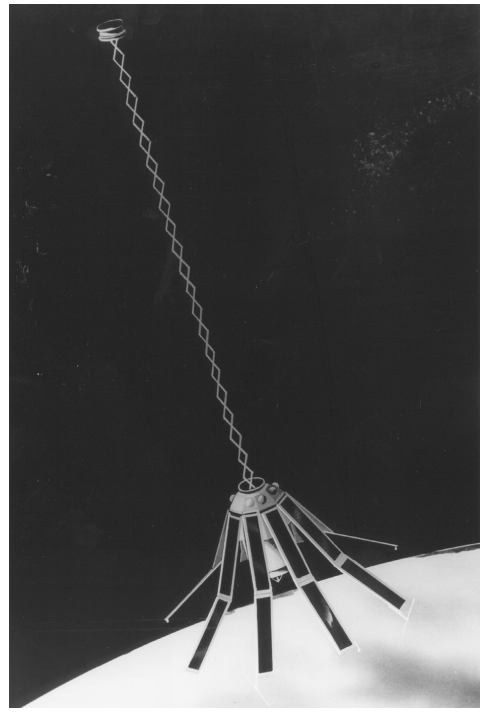


***Figure 5.19*** The GEOSAT Spacecraft. (Photo Courtesy Applied Physics Lab, Johns Hopkins University.)

cost to adapt an existing design is often less than developing a whole new concept. Of course, it is not always the case that an existing design is directly applicable. In such circumstances, the development of hybrids from existing designs might be appropriate. Other situations might necessitate that the design team start with a "clean sheet of paper" and develop a new concept from scratch.

In the early stages of development, parametric analyses are often performed. This is helpful where certain parameters are coupled in such a way that increasing or improving one parameter may have adverse effects on one or more other parameters. These kinds of analyses show how changes ripple through the design in terms of their effects on other parameters.

For example, for a mission in which high resolution images will be taken, there will be trade-offs between the resolution (sharpness) of the images taken and the footprint or coverage of each image (an entire geographic region or a small city block). This trade-off must also consider on-board data storage and/or downloading the data to ground stations. Depending upon the downlink, a fixed amount of data can be transmitted to the ground at any given opportunity for ground contact. This will limit the amount of data that should be generated by the telescope. The trade-off then becomes one

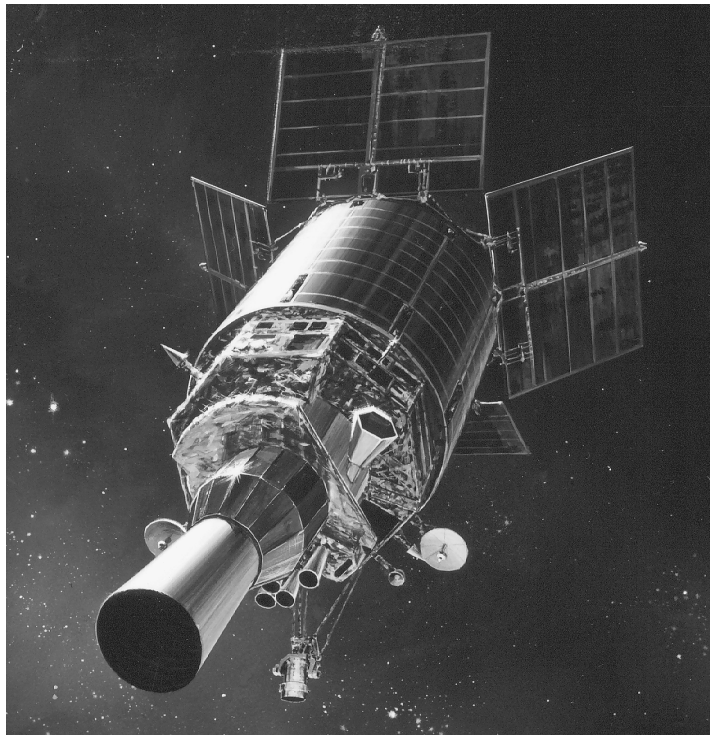


***Figure 5.20*** The Defense Support Program (DSP) Spacecraft. (Photo Courtesy TRW, Inc.)

of resolution vs. image size. If the downlink capacity is too restrictive, this analysis may indicate a trade-off of data quality vs. the cost of adding more downlink capacity. This is just one simple example of what sort of parametric analyses can be performed to quantify design space.

*2. Analysis*

Figure 5.25 highlights the Analysis and Allocation activities in the context of the "Design, Analyze, Integrate" activity. The "Analysis" activity includes any and all analyses necessary to support quantification of design space and design parameters, as well as to ascertain technical, cost, schedule, and risk performance of the system. Since the goal of this book is simply to establish a framework for complex system design, it is beyond the scope of this discussion to elaborate on these. Therefore, only some of the myriad analyses that might take place on a given system development program are listed:

- Mission, system, electrical, digital, analog, RF, mechanical, etc.
- Simulations
- FMECA (Failure Modes Effects and Criticality Analysis)

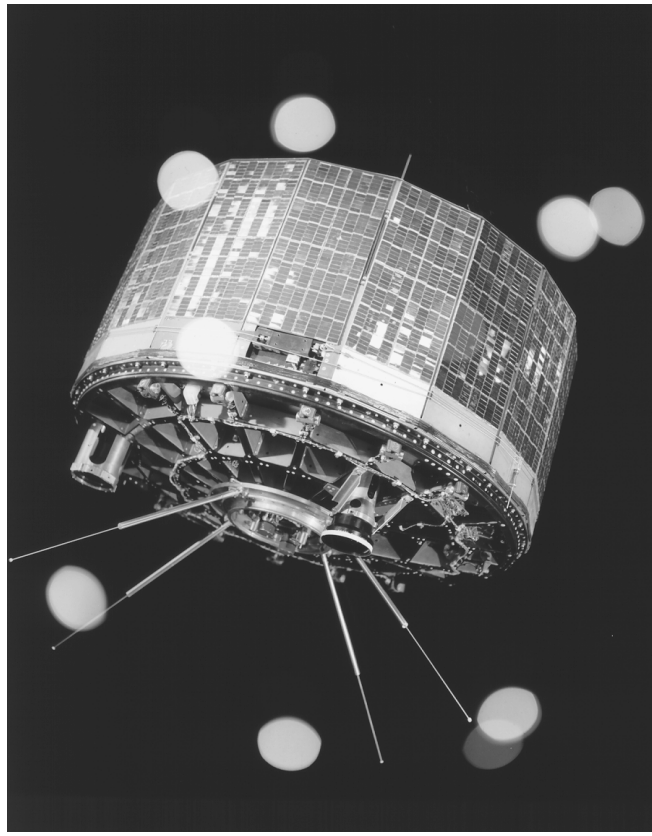


***Figure 5.21*** TIROS II Spacecraft. (Photo Courtesy NASA.)

### 3. *Allocation*

Allocation involves not only technical elements, but also the cost and schedule components of the system development. There are also several managerial activities that are associated with the allocation activity. Thus, allocation involves the following:

- Allocate functionality, performance, constraints to H/W and S/W elements
- Define budgets
  - Technical: mass, power, throughput, memory, RF links, etc.
  - Reliability, contamination, etc.
  - Margin and contingency rules
- Configuration management — Controlling the budgets
- Risk management — Assessing convergence, as per Figure 5.26
- Performance monitoring, metrics development, defining/refining TPMs
- Cost and schedule management

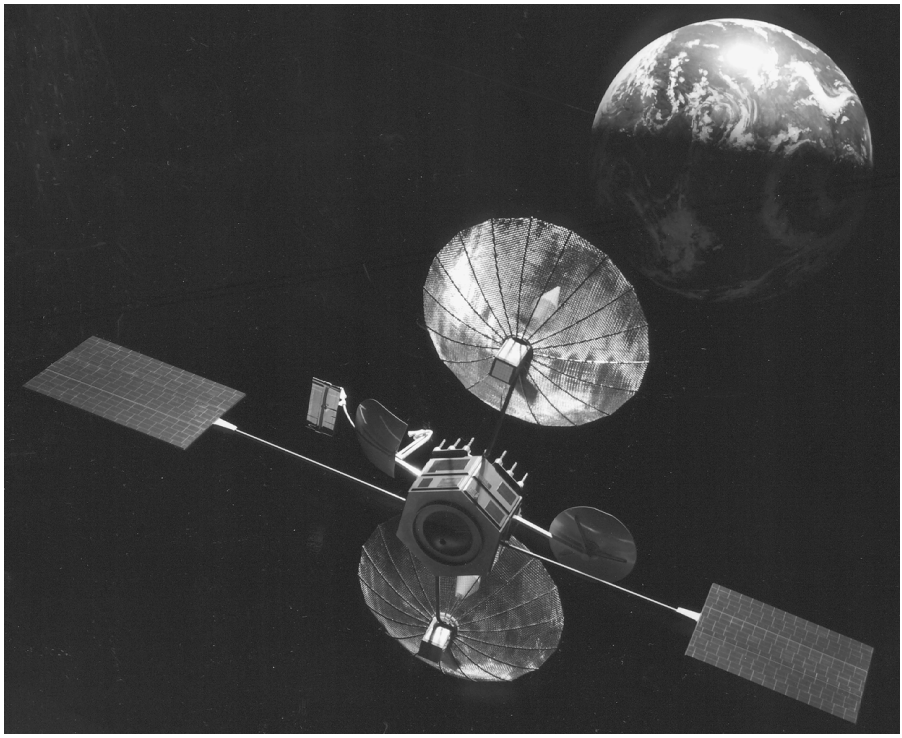


*Figure 5.22* Tracking and Data Relay Satellite (TDRS). (Photo Courtesy TRW, Inc.)

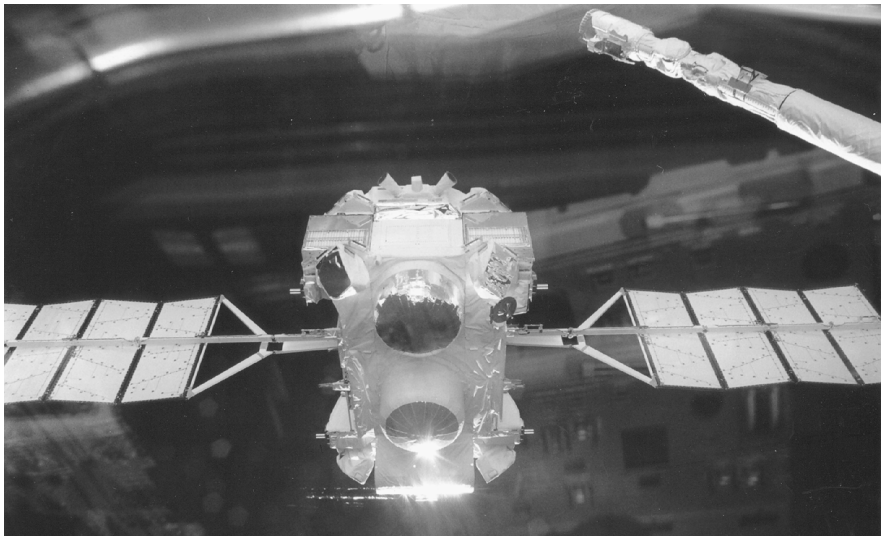


*Figure 5.23* The Compton Gamma Ray Observatory (CGRO). (Photo Courtesy TRW, Inc.)

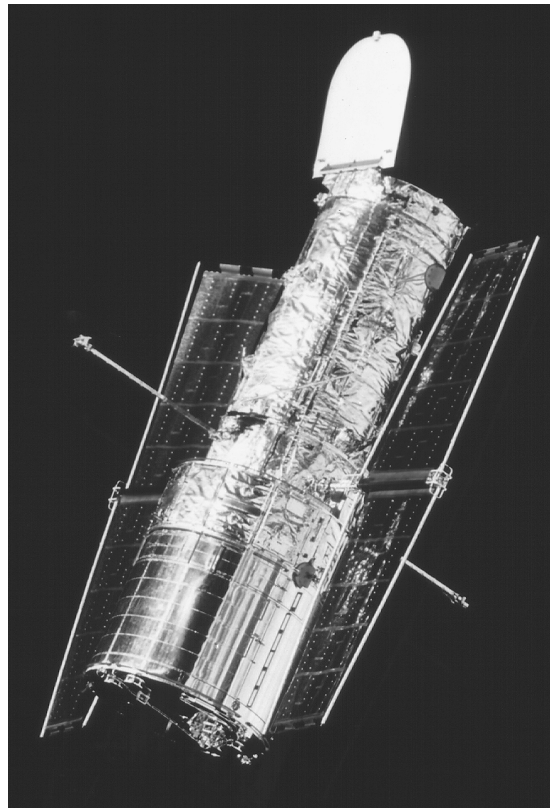


***Figure 5.24*** Hubble Space Telescope (HST). (Photo Courtesy NASA.)

Some important questions arise in any discussion concerning margin and its role in the allocated budgets: How much margin should be included? How should this change over time? Figure 5.26 provides a notional depiction of how margin should converge to a small percentage of the budget as uncertainty in the design decreases. The point of the figure is not to prescribe specific numbers, but rather to suggest that *margin must be factored into the development activity and monitored to ensure that the system is converging upon a low-risk solution*. The upper and lower curves represent reasonable amounts of uncertainty as the development progresses. If these bounds are exceeded, unforeseen risk may be indicated.

**Output** → Budgets, technical performance measures

Figure 5.27 illustrates how functionality is derived directly from the input requirements and that the implementation is driven by the functionality and associated performance required. Defining functionality without defining the required performance is not useful to the designer. As discussed

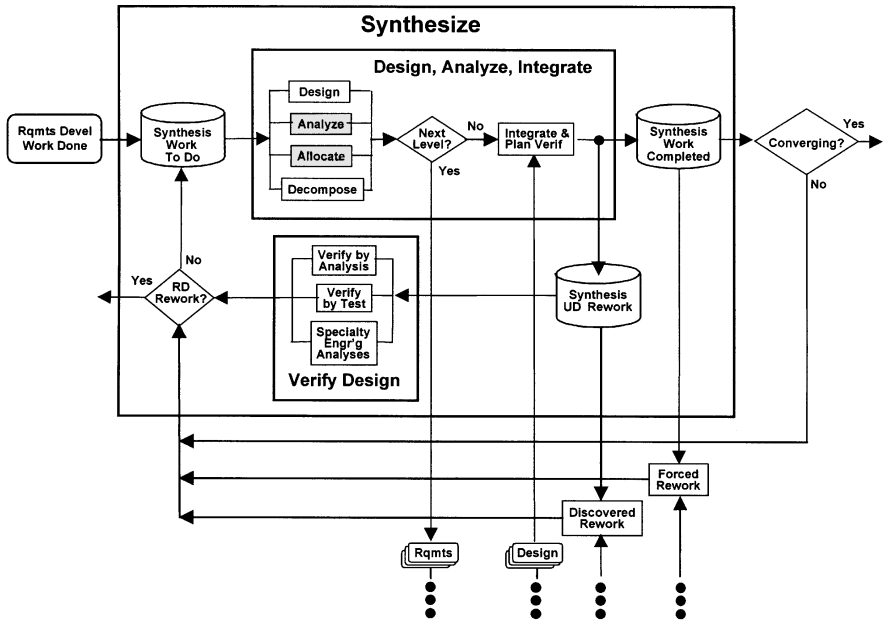



*Figure 5.25* The "Analyze" and "Allocate" Activities.

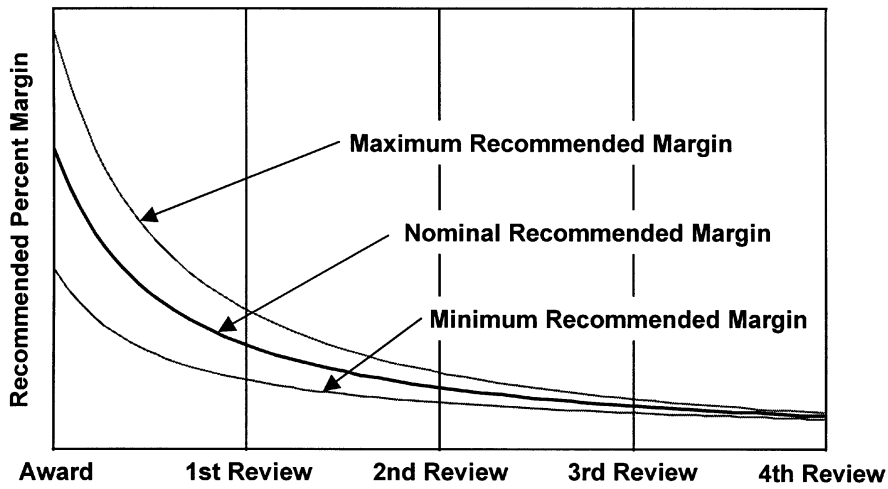



*Figure 5.26* Notional Convergence of Margin and Reduction in Uncertainty.

above, the ADACS implementation chosen is based primarily upon the accuracy the subsystem must provide to the spacecraft. Some spacecraft require no ADACS system because the payload is not dependent upon pointing the satellite in any particular direction.

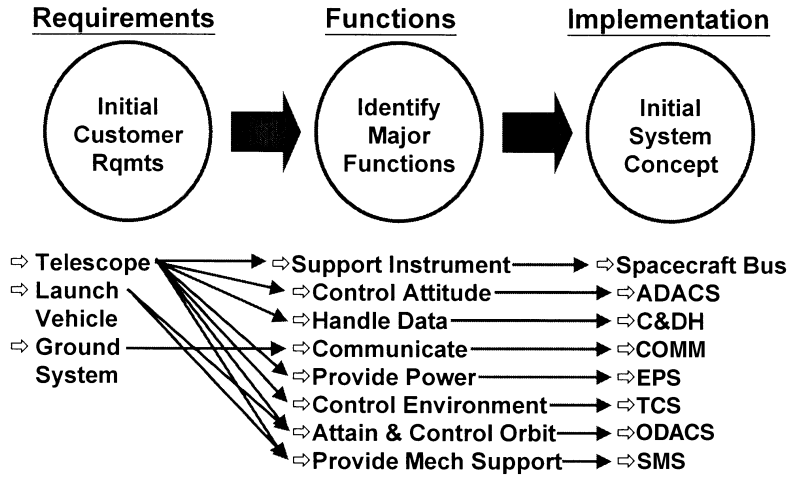



*Figure 5.27* Allocation of Functionality to Implementation.

**Key Point**

> The point here is that there must be traceability between requirements, functionality, and implementation. Functions must be allocated to specific elements of the design. This ensures that the design is appropriate for the intended use. Resources are not wasted by over-design and the mission is not unsuccessful because of insufficient capability.

If a certain system element has no functionality allocated to it, the question ought to be raised as to why that element is included in the system. Of course, where heritage designs are used, it may be cost and schedule effective to retain functionality that is not required, simply because it is cheaper and/or more schedule efficient not to eliminate it from the existing design.

As shown in Figure 5.27, seven subsystems have been identified for ESAT: attitude determination and control system (ADACS), orbit determination and control system (ODACS), command and data handling (C&DH) subsystem, electrical power subsystem (EPS), communications (COM), propulsion subsystem (PRS), structures and mechanisms subsystem (SMS), and the thermal control subsystem (TCS).[50]

[50] Software is not identified as a separate subsystem, but is included in the implementation of each appropriate system element. The spacecraft bus as a system includes both hardware and software, as do most or all of the major subsystems. Allocation of a function or set of functions to hardware or software is a decision made during the design development. While software may be developed by a distinct functional organization, it is not viewed herein as a separate subsystem in terms of the design itself. System-level software is managed at the system level of the design; subsystem-level software is managed at the subsystem-level of the design; and so on. When a particular subsystem is discussed, it is assumed that it is comprised of all its constituent elements, including both hardware and software.

At the spacecraft bus hierarchical level, technical budgets are generated for each identified subsystem. The technical budgets define mass, electrical power, memory, throughput, etc. for each subsystem. These budgets are very important because, at the early stages of the development, they can be used to determine the risk level of the program to a significant degree. Early in the program, the design effort focuses on the upper levels of the design. Often, budgets are allocated to lower-level system elements without detailed analyses to validate the budgets. There is, therefore, risk introduced into the program because there is some probability that those system elements cannot be accommodated within their assigned budgets. An element may require more allocation of mass, power, or other resources. Hopefully, such a problem can be accommodated by reallocation of margin or contingency. If not, the ripple effects can be significant.

Figure 5.28 depicts a matrix where each allocated resource is represented as a column and each subsystem as a row. The figure illustrates only mass and power. However, resource budgets should include not only these two, but also all others as well. Other resources might include: memory, throughput, communication links, etc. The matrix should include the current value of the resource used by the element, the current budget, and the margin remaining for each. Risk areas can be identified based upon the rate at which a particular resource is being consumed as the design matures.

*4. Functional Decomposition*

Figure 5.29 highlights the "Decompose" activity of the "Design, Analyze, Integrate" activity.

As discussed previously, a function cannot be decomposed without some knowledge of "how" the system will be implemented. Therefore, functional decomposition to the next level down in the hierarchy logically follows the generation of concepts at the level above. Thus, it is here in the overall process that functional decomposition is performed. The following series of activities are necessary to properly decompose a system or system element.

- Receive function and performance requirements from the Requirements Development activity
- Develop design concepts that implement the identified functions at the required performance levels
- Decompose the implementation into subfunctions for the next-level-down activity
- Identify the interfaces between the subfunctions
- Partition subfunctions into logical groups (potential subsystems); group the functions such that interfaces are minimized between logical groups
- Generate the functional model and verify the functional definition
- Generate function and performance requirements (specifications and ICDs) for each logical grouping of functions

| | Mass | | | | Power | | | |
|---|---|---|---|---|---|---|---|---|
| | Current | Budgeted | Margin | | Current | Budgeted | Margin | |
| | | | Number | Percent | | | Number | Percent |
| **Total Spacecraft** | 2170 | 2400 | 230 | 9.6% | 953 | 1105 | 152 | 13.8% |
| | | | | | | | | |
| **Telescope** | 925 | 1000 | 75 | 7.5% | 300 | 350 | 50 | 14.3% |
| | | | | | | | | |
| **Spacecraft Bus** | 1245 | 1400 | 155 | 11.1% | 653 | 755 | 102 | 13.5% |
| **Structure & Mechanisms** | 450 | 500 | 50 | 10.0% | 8 | 10 | 2 | 20.0% |
| **ADACS** | 85 | 100 | 15 | 15.0% | 20 | 25 | 5 | 20.0% |
| Earth Sensor | | | | | | | | |
| Star Tracker | | | | | | | | |
| Sun Sensor | | | | | | | | |
| RWAs | | | | | | | | |
| IMU | | | | | | | | |
| Torque Rods | | | | | | | | |
| Etc. | | | | | | | | |
| **EPS** | 400 | 450 | 50 | 11.1% | 400 | 450 | 50 | 11.1% |
| Batteries | | | | | | | | |
| Solar Arrays | | | | | | | | |
| Distribution | | | | | | | | |
| Etc. | | | | | | | | |
| **Comm** | 65 | 75 | 10 | 13.3% | 125 | 150 | 25 | 16.7% |
| Transmitters | | | | | | | | |
| Antennas | | | | | | | | |
| Etc. | | | | | | | | |
| **C&DH** | 45 | 50 | 5 | 10.0% | 38 | 45 | 7 | 15.6% |
| Command Processor | | | | | | | | |
| Data Storage | | | | | | | | |
| Command Distribution | | | | | | | | |
| Etc. | | | | | | | | |
| **Thermal Control** | 20 | 25 | 5 | 20.0% | 55 | 65 | 10 | 15.4% |
| Thermal Blankets | | | | | | | | |
| Heaters | | | | | | | | |
| Thermisters | | | | | | | | |
| Etc. | | | | | | | | |
| **Propulsion** | 180 | 200 | 20 | 10.0% | 7 | 10 | 3 | 30.0% |
| Hydrazine Tanks | | | | | | | | |
| Thrusters | | | | | | | | |
| Valves | | | | | | | | |
| Etc. | | | | | | | | |

*Figure 5.28* Allocation of Technical Budgets.

- Release function and performance requirements to lower-level development activities
- Receive feedback from lower-level development activities and refine specifications and ICDs
- Iterate as necessary

In Figure 5.30, L0 indicates Level 0 which, for this example, represents the top level of the system hierarchy. L1 indicates the next level down, or the subsystem level of the hierarchy. Notice that there are three subsystems indicated in the figure at level L1; often there are many more. Again, making use of the above definition of "system," it is asserted that this approach to functional decomposition is applicable to all levels of the system hierarchy. First, imposed requirements are input to the Requirements Development activity. Next, the functions with their performance requirements are analyzed and

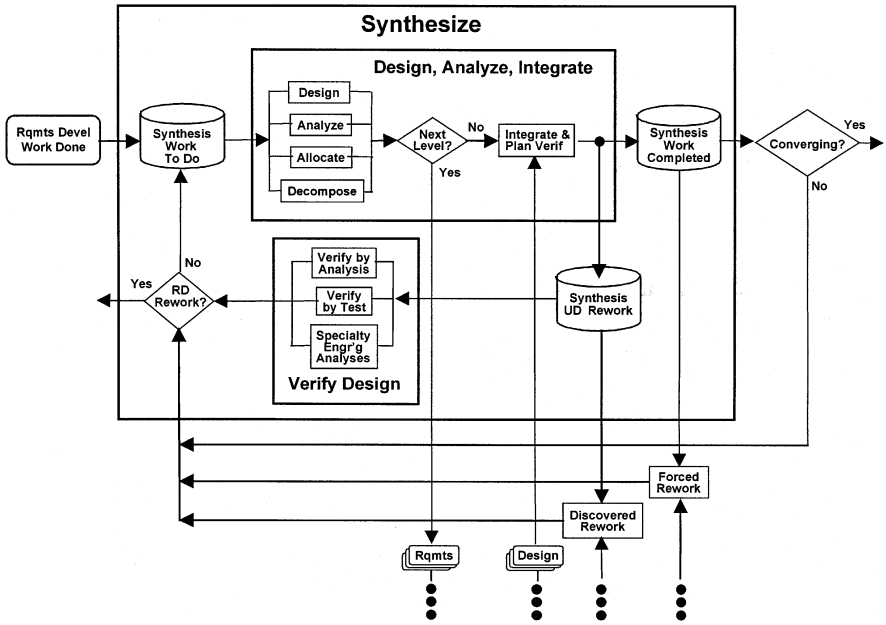



*Figure 5.29* The Decompose Activity.

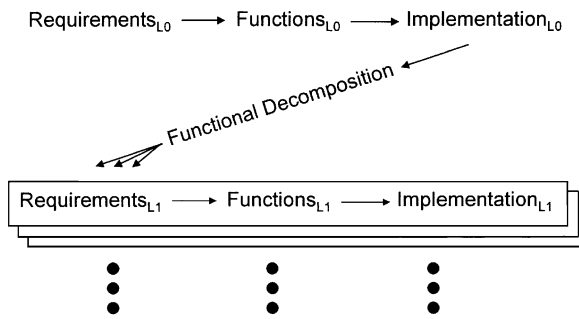



*Figure 5.30* Functional Decomposition Methodology.

coalesced into a functional model and input to the Synthesis activity. Then, implementations are developed in response to the functional model. Finally, with a concept in mind, functional decomposition is performed as next-level-down functions are derived from the implementation candidate. These functions and required performance parameters are then collated into a requirements set as input to the next-level-down Requirements Development activity and the cycle is repeated as necessary.

Figure 5.31 illustrates the organic connection between implementation and functionality. Within the same level of the hierarchy, the requirements set (or functional architecture) drives the implementation architecture. Between tiers, the implementation at the tier above drives the functional

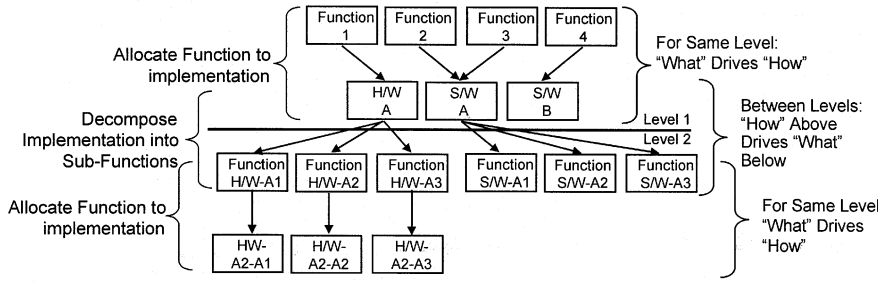



***Figure 5.31*** The "How" and "What" Relationship.

architecture of the subsystems below. Notice in the figure that more than one function can be allocated to the same design element. However, only one element implements any particular function. In other words, neglecting issues such as redundancy, the exact same function should not be provided by multiple components of the system. This could lead to confusion and cause malfunctioning of the system. This may not be the case, however, where the same function is needed in differing contexts.

One of the key issues involved in the decomposition of an implementation is the partitioning of the derived functions into subsystems. Pimmler and Eppinger note, "For a complex product . . . there are thousands of possible decompositions which may be considered. Each of these alternative decompositions defines a different set of integration challenges."[51] A key criterion to optimize the partitioning is the minimization of the number of interfaces or interdependencies between the identified functions in order to minimize the integration problem.[52]

Figure 5.32 illustrates an overview of the development by decomposition process, focusing on the attitude determination and control subsystem. The relevant input requirement originates with the customer's telescope. It requires that the spacecraft bus control the telescope attitude along all three axes to an accuracy of 0.01 degrees (requirement 2.2.8 from Table 5.1). Thus the function flowing from this requirement is "control instrument attitude." This function must be achieved at a performance level of 0.01 degrees along all three axes. This portion of the functional analysis is complete now that the function has been identified with its respective performance requirement. With the function identified, candidate designs can be generated. There are often many ways to implement a particular set of functions. In this simple example that is the case. There are several ADACS architectures that can be considered to implement the required functionality. Four potential design

[51] Pimmler, Thomas U. and Steven D. Eppinger, Integration Analysis of Product Decompositions, *Design Theory and Methodology*, DE-Vol. 68, ASME 1994, p. 343.

[52] The Design Structure Matrix (DSM) provides a useful tool for determining the best segregation of functional elements into collected sub-elements. Cf. Robert P. Smith, Steven D. Eppinger, "Identifying Controlling Features of Engineering Design Iteration," *Management Science*, vol. 43, no. 3, pp. 276-293, March 1997.

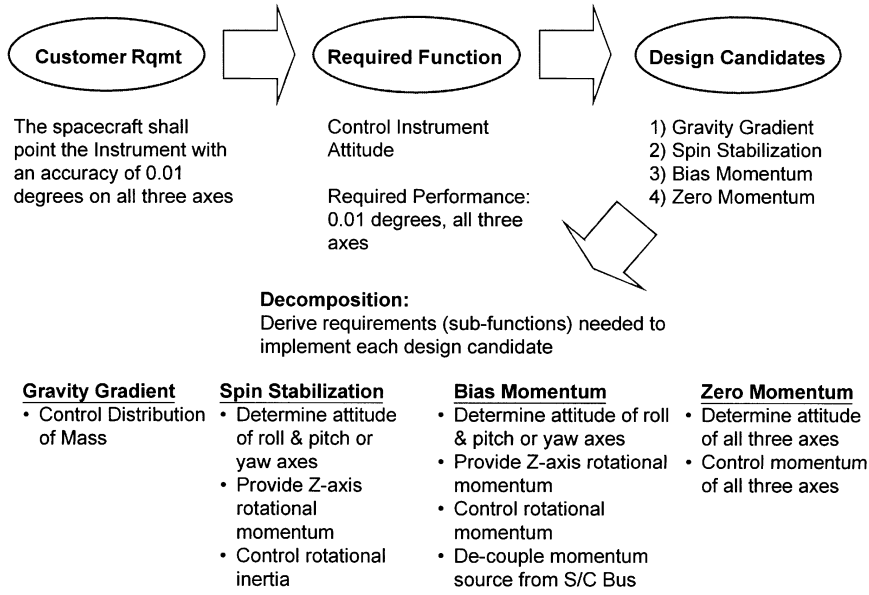



*Figure 5.32* Second Level Decomposition.

solutions have been identified: gravity gradient, spin stabilization, bias momentum, and zero momentum.

While this is certainly not a book about spacecraft attitude control, it might be helpful to give a brief explanation of how each of these designs work.

**Gravity Gradient** — Gravity gradient uses the force of the earth's gravity field to cause the spacecraft to point one axis toward the earth. The mass of the spacecraft bus and payload components is typically concentrated at one end of the spacecraft (Figure 5.19). Usually a second mass, less than the first, is deployed a relatively large distance from the first. The force of gravity on the first mass is greater than that on the second and it is this "gradient" that causes the minor axis of the spacecraft to point in the proper direction within a few degrees. In general, there is no control of spin along the axis pointed toward the earth.

**Spin Stabilization** — Spin stabilization is implemented by spinning the spacecraft about the axis pointed toward the earth (Figures 5.20 and 5.21). The inertial force created by the spinning motion creates a stabilizing force along the spin axis, much like a spinning top or gyroscope. The downside of this control system is that the spacecraft, or a portion of it, must spin (some spacecraft implement a "despun platform" to offset this problem). This is adequate for certain missions but not for others.

**Bias Momentum** — Bias momentum uses the same inertial force as the spin stabilized spacecraft, but it is contained in a device called a momentum wheel (Figure 5.22). The momentum wheel spins, but the spacecraft itself

does not. This allows all three axes to be stabilized. Momentum stored in the momentum wheel is managed by thrusters and/or torque rods that leverage the forces of the earth's magnetic field. One of the major drawbacks of spin stabilized and bias momentum stabilized systems is that maneuverability is difficult. A second drawback is that the spinning motion can induce disturbances (vibrations) into the payload.

**Zero Momentum** — Zero momentum systems also stabilize all three axes but without using any spinning motion for that purpose (Figures 5.23 and 5.24). The small disturbances encountered are taken out by storing the momentum created by those disturbances in devices called reaction wheels. Most zero momentum systems have one reaction wheel on each axis that is used to store momentum induced on that axis. Magnetic torque rods are used to "dump" momentum periodically. At opportune points in the orbit, torque rods are turned on that generate a magnetic dipole which interacts with the earth's magnetic field to apply the appropriate torque to the satellite. This torque allows the reaction wheel to slow down, or "dump," its momentum with minimal net disturbance to the spacecraft.

This explanation of spacecraft attitude control systems has been necessarily brief. The purpose is simply to facilitate an understanding of design by decomposition, using the spacecraft attitude control system as an example. If the reader requires a more extensive understanding of spacecraft attitude control systems, there are a number of excellent texts that can be consulted.[53]

Consider again the example illustrated in Figure 5.32. Once a subsystem architecture has been identified it can be decomposed into subfunctions. Notice that each ADACS architecture has a different decomposition since the functions required to implement each architecture are different. The gravity gradient architecture requires only that the mass of the spacecraft be distributed in a certain way; no other sensors or effectors are necessary. The spin stabilization and bias momentum architectures require similar functionality since both use rotational inertia created by a spinning mass and both require that the attitude of the roll and pitch or yaw axes be determined. The key difference between them, in terms of basic functionality, is that the bias momentum system requires that the source of the rotational inertia, the momentum wheel, be decoupled from the rest of the spacecraft. This decoupling allows the spacecraft to remain fixed relative to the earth while the rotating momentum wheel generates and maintains the stabilizing rotational inertia. Finally, the zero momentum architecture requires that the attitude of all three axes be determined and that the momentum of all three axes be maintained at zero.

At this point functional analysis is commenced by identifying the interfaces between the identified functions. The focus will be on the zero momentum architecture as this example is continued. Figure 5.33 represents the

[53] See, for example, Wertz, James R., Ed., *Spacecraft Attitude Determination and Control*, D. Reidel, Dordrecht, The Netherlands, 1997 reprint.

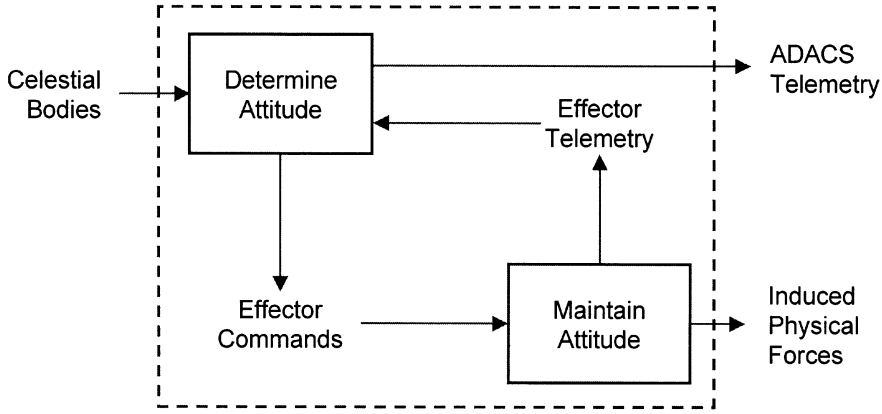



*Figure 5.33* "Control Attitude" Function Decomposed.

decomposition of the zero momentum architecture. There are two primary functions necessary: determine attitude and maintain attitude. The figure depicts these two functions and their interfaces.

As has been emphasized, in order to perform decomposition there must be some understanding as to what the implementation will be.[54] As the decomposition of the function "Determine Attitude" is approached, the question arises, "how might this be implemented?" Several different methods for determining spacecraft attitude could be conceived: rate integration using gyroscopes, data from the Global Positioning System, uplink attitude data from the ground. For different reasons, these concepts may or may not be viable. The basic questions that must be considered here are "What resources are available?" and "What performance level is required?"

For ESAT, the focus will be determining attitude by exploiting the availability of celestial bodies. Now that a concept has been identified, a functional decomposition can be performed because the functions that must be performed in order to determine attitude using celestial bodies are now known. First, the celestial bodies must be sensed and the appropriate data must be generated, allowing the spacecraft to make use of that information. Second, the spacecraft needs to receive the data concerning the sensed celestial bodies, so that it can process it to determine its attitude. Finally, appropriate commands must be generated and distributed that will tell the spacecraft effectors what to do in order to maintain the desired attitude. As before, the interfaces between these functions must be identified and characterized. Figure 5.34 depicts the results of this process.

Figure 5.35 shows the functional decomposition of the "Maintain Attitude" function. In order to counter disturbances the spacecraft will experience, the system must be able to generate rotational momentum opposite

[54] To reiterate, this does not imply that great detail must be provided. The point is simply that at least a concept of how the function will be implemented is necessary before decomposition can commence.

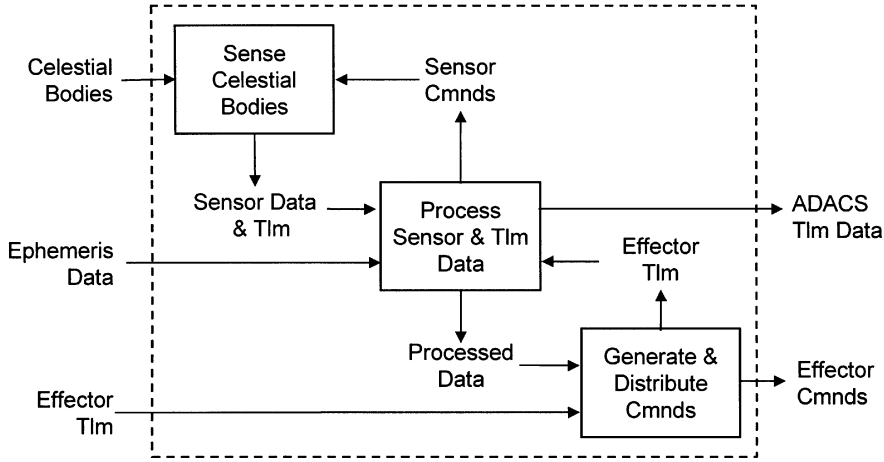



*Figure 5.34* "Determine Attitude" Function Decomposed.

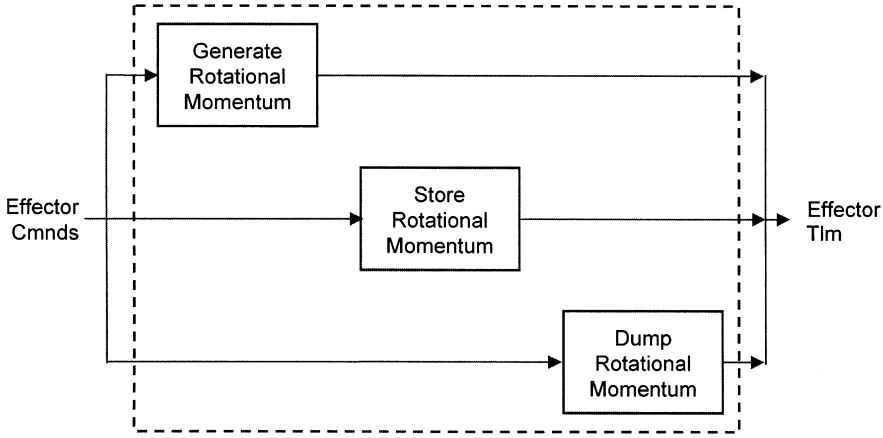



*Figure 5.35* "Maintain Attitude" Function Decomposed.

the direction of the disturbance. Therefore, the "Generate Rotational Momentum" function is included in the system. Second, any momentum removed from the satellite must be stored until it can be dumped, thus a "Store Rotational Momentum" function is provided. Finally, because the rotational momentum storage devices will not have infinite capacity, a function is needed to "dump" rotational momentum periodically. The interfaces between these functions must also be identified and characterized.

The preceding discussion provides an example of one of the architecture candidates (zero momentum). If the other architectures were serious candidates, similar diagrams for each would be developed. The next step in the framework is to develop requirements for the next-level-down Requirements Development activity.

**Output** → Lower-level validated specifications and ICD(s), lower-level simulation

*5. Inter-Level Interface*

It is at this point in the SDF where the vertical interface between system elements occurs. This is the interface between the System and Subsystem activities, for example. This same interface occurs at all vertical interfaces.[55] Requirements are passed down the hierarchy and design data and other data are fed back here.

*Information Flow-Down* — Without elaboration (this is beyond the scope of this book) it is suggested that any "design-to" data must be configuration controlled to some degree. The formality and rigor of the configuration management effort must be commensurate with the program need.

*Data Feedback* — Development issues relating to reallocation and redesign must be managed. In order to reallocate effectively, knowledge of where margin and contingency reside in the system is necessary. Therefore, this information must be documented in the system budgets.

**Output** → Configuration controlled documentation

*6. Integration*

Figure 5.36 highlights the "Integrate and Plan Verification" activity that is performed within the "Design, Analyze, Integrate" activity.

The Integration activity is a key one. This is the point in the process at which all the elements of the design are integrated or synthesized into a coherent whole. A major concern of the integration task is interfacing between the various system elements. This task is primarily concerned with synthesizing the design by updating design data, and managing the activities identified previously and bulleted below.

- Identify and characterize interfaces
- Note specifications, ICDs, databases, etc.
- Update design definition
- Update mission timeline and operations concept
- Block diagrams, schematics, drawings, layouts
- Management activities
  - Performance Measurement — Budgets, etc.
  - Subcontract Management
  - Risk Management — Identification, assessment, and mitigation approaches
  - Configuration Management — Configuration Control Board (CCB)

**Output** → Integrated design that includes the data generated above

[55] This is also depicted in Chapter 6 in Figure 6.4, "Program Team Interactions."

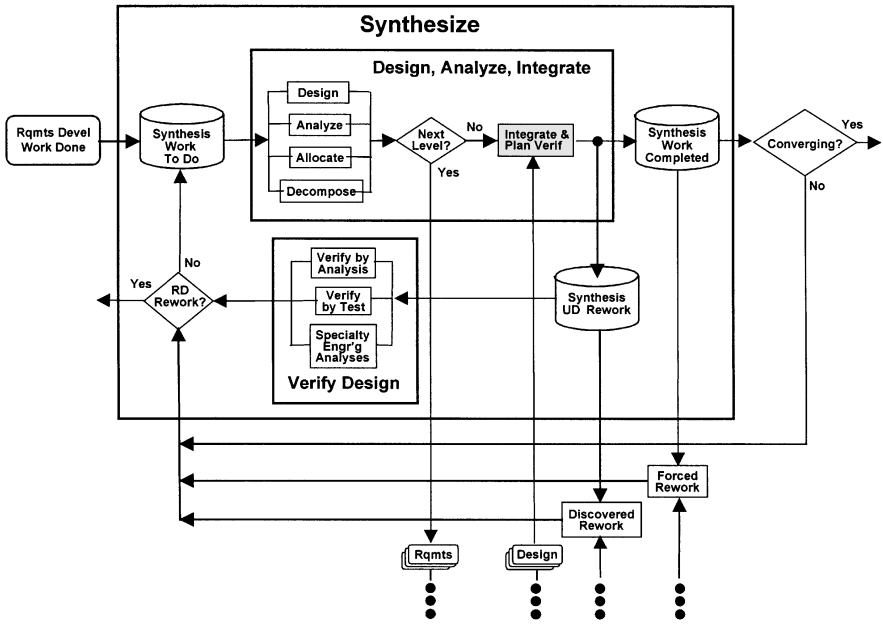



*Figure 5.36* The "Integrate and Plan Verification" Activity.

Figure 5.37 provides a view of the integrated ESAT spacecraft in the form of a system block diagram, integrating the zero momentum ADACS architecture. Both the telescope and the spacecraft bus are shown, with key interfaces between them identified. The payload system is identified as such in the figure; all else represents the spacecraft bus. Each subsystem is shown with major components. Both the internal and external interfaces are identified.

## *B. Rework Discovery Activities: Design Verification*

**Assess "How-Well"** — Figure 5.38 highlights the Rework Discovery activities performed within the Synthesize activity. The primary focus is verification of the developing design.

Verification involves two basic activities: Design Verification which focuses on the developing design, and Product Verification which focuses on the deployed system. Verification is accomplished by performing test, analysis, simulation, demonstration, or inspection. In this section, the primary concern is with Design Verification in order to show how it is involved in the development of the system design.

**Key Point**

> The Verification activity is an important part of the SDF. It is performed at the earliest stages of development and continues through the entire design phase.

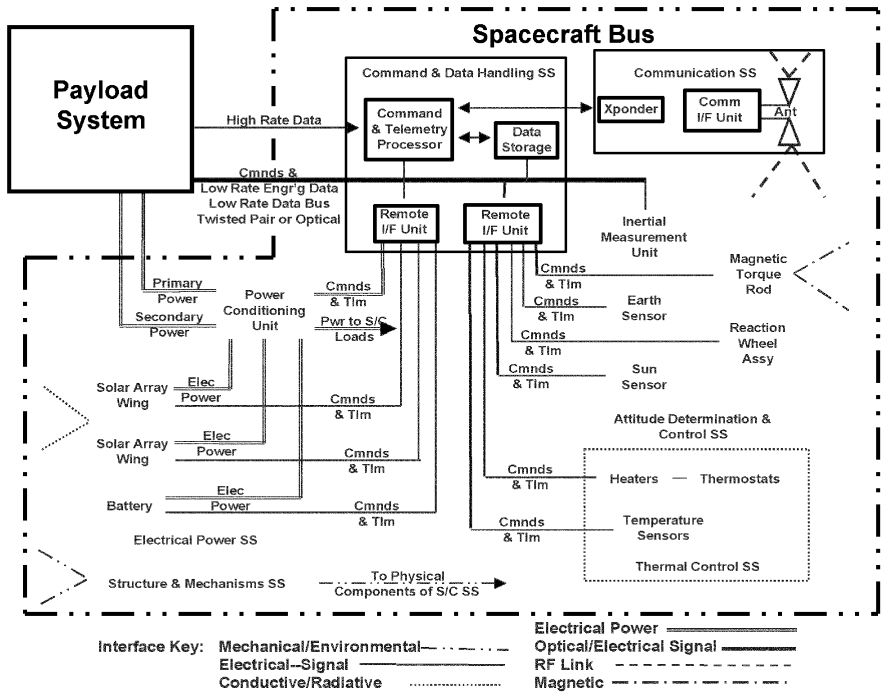



***Figure 5.37*** Integrated Spacecraft System: A Notional System Block Diagram.

### 1. *Analysis and Test*

#### a. *Analysis*

Those analyses aimed at determining "how well" the current design meets its requirements; these are in contrast to those analyses aimed at defining design space which are performed as described previously in the design activity.

#### b. *Test*

- Planning Activities (e.g., test requirements, test flow, resource planning, etc.)
- Testing Activities (e.g., engineering test models, prototypes, breadboards)

### 2. *Producibility, Testability, and Other Specialty Engineering Activities*

This activity assesses those areas of the design commonly called "specialty engineering" concerns.

- Is the design testable within resource and time constraints?
- Is the design producible within resource and time constraints?
- Is the design acceptable with respect to EMI/EMC, reliability, maintainability, affordability, supportability, etc. parameters?

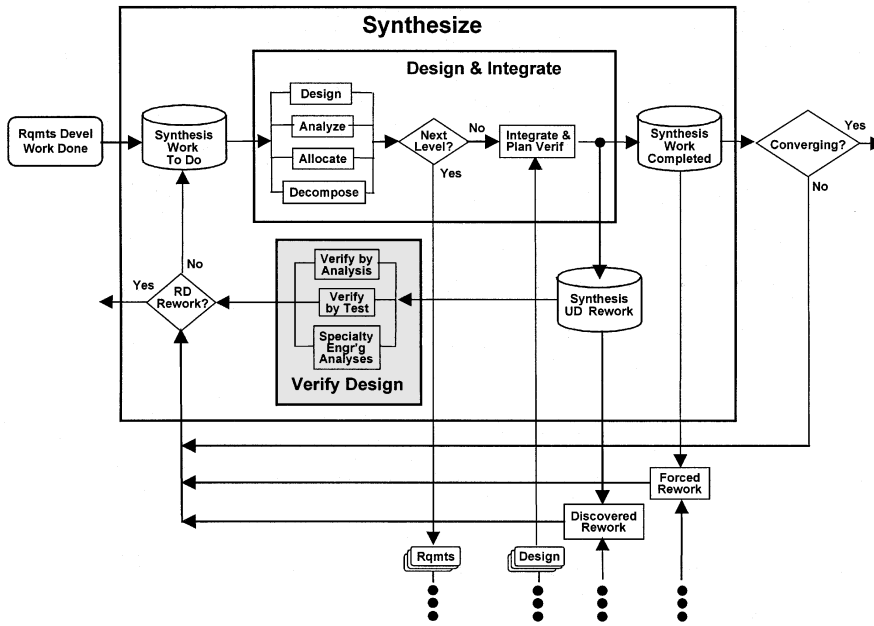



*Figure 5.38* The "Synthesize" Rework Discovery Activities.

**Output** → The output of the Design Verification activity is a design that has been assessed as to how well it meets all the requirements.

## *III. Trade Analysis*

Figure 5.39 illustrates where Trade Analyses are performed in terms of the logical sequencing of activities. Trades logically occur after Synthesis because the trade criteria (technical, cost, schedule, risk)[56] must be developed in order to make the selection.

This in no way implies that trade studies must wait until competing designs are complete. The reader is reminded that the present discussion takes place in the context of the Logical Domain, not the Time Domain. Only the logical sequencing of activities on the micro-time scale is in view here (recall Figure 3.1 and related discussion). Although generally emphasized in the early phases of development, trade analyses can occur during all development phases as the SDF is performed and iterated. The point here is that trade studies are dependent upon technical, cost, schedule, and risk criteria which can only be developed with reference to implementation. Absent such criteria, there is no basis for making a selection.

[56] Chestnut concurs, "Commonly accepted bases for judging the value of a system: (a) performance; (b) cost; (c) time; (d) reliability; (e) maintainability." Harold Chestnut, *System Engineering Tools*, New York: John Wiley & Sons, 1965, p. 11.

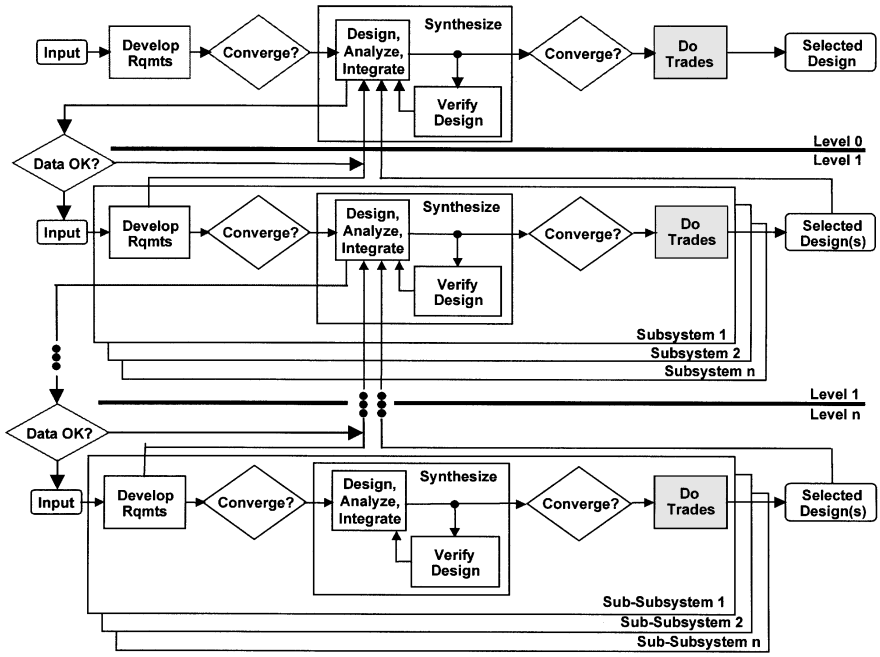



***Figure 5.39*** The "Do Trades" Activity.

There are many selection methodologies that can be employed in the selection process.[57] It is not the purpose here to discuss the pros and cons of the various trade methodologies found in the literature, but simply to delineate where in the System Development process Trade Analyses occur. The amount of rigor applied to the Selection process should be commensurate with customer requirements, program need, and other criteria as determined by the development team.

If multiple candidates emerge from the design and analysis activity, the selection process is implemented. However, if multiple candidates are compliant and equally acceptable to the design team, integrate each into the element of the level above and analyze for system benefit there. The selection is then made at that above level.

Here in the Trade Analysis section, it is appropriate to highlight the classic trade-off that occurs in most any System Development activity: cost, schedule, and technical performance. Issues relating to risk, robustness, safety, etc. could also be included, but certainly cost, schedule, and technical performance are central concerns. This is illustrated in Figure 5.40.

It is not often that all three of these issues can be improved simultaneously. In general, one or two can be improved at the price of the third or other two. The author once worked with an engineer who had a similar

[57] e.g., Pugh; DSMC; McCumber, William H., *System Performance Representation: Standard Scoring Functions*, NCOSE 1995, P003; Ulrich and Eppinger, pp. 105-122.

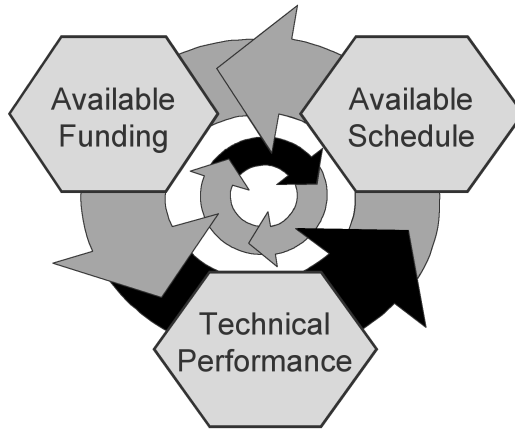



***Figure 5.40*** The Classic Trade-Off.

graphic in his office. It had the caption, "faster, better, cheaper — pick any two!" While this may not be an absolute truth in every situation, it does serve to emphasize the non-trivial trade-off that often must take place.

Returning again to the ESAT example, the relative merits of each of the four concepts identified above are considered. Figure 5.41[58] describes the four ADACS architectures with a brief critique of each. The key requirement here is the pointing accuracy, customer input requirement 2.2.8 from Table 5.1. That requirement states that the spacecraft bus shall point the telescope to an accuracy of ±0.01° on all three axes. The only architecture that has the capability of meeting such a requirement is the zero momentum design. Therefore, the zero-momentum concept is the design of choice.[59]

## *IV. Optimization and Tailorability*

### *A. Optimization*

Similar to the argument just asserted for Trade Analyses, optimization techniques are myriad and can be very specialized. It is not the purpose of this book to discuss various optimization approaches. Rather, the purpose is to describe where optimization occurs in the process and how it impacts the overall development process.

[58] Attitude control accuracy numbers are taken from Larson, Wiley J. and James R. Wertz, *Space Mission Analysis and Design*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992, Table 11-4, p. 346.

[59] In this simple example the selection was made purely on the basis of technical performance. As has been discussed above, in the real world cost and schedule must also be considered. Note also that, as is often the case, there are several ways to implement the selected system. A zero momentum attitude control system is no exception and this will likely lead to more trade-offs.

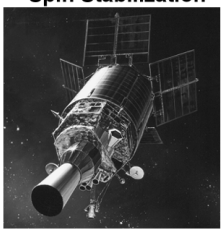
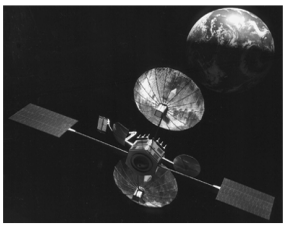
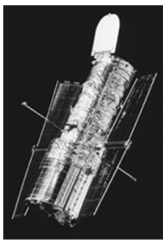
| Architecture | | | |
|---|---|---|---|
| **Gravity Gradient** | **Spin Stabilization** | **Bias Momentum** | **Zero Momentum** |
|  |  |  |  |
| **GEOSAT**<br>*Photo Courtesy APL JHC* | **DSP Spacecraft**<br>*Photo Courtesy TRW* | **TDRS**<br>*Photo Courtesy TRW* | **Hubble Space Telescope**<br>*Photo Courtesy NASA* |
| **Design Elements** | | | |
| Passive<br>Mass Dist | Aspect ratio<br>Mass Balance<br>Earth Sensor<br>IMU, OBC<br>Thrusters<br>Mag Torqures | Earth Sensor<br>Sun Sensor<br>IMU, OBC<br>Thrusters<br>Momentum Wheel<br>Mag Torquers | Earth Sensor<br>Sun Sensor<br>Star Tracker<br>IMU, OBC<br>Thrusters<br>Reaction Wheels<br>Mag Torquers |
| **Accuracy** | | | |
| ± 5° two axes | ± 0.1° to ± 1° | ± 0.1° to ± 1° | ±0.001° to ± 1° |
| **Assessment** | | | |
| Cannot meet 0.01 accuracy rqmt | Spin not OK<br>Cannot meet 0.01 accuracy rqmt | Cannot meet 0.01 accuracy rqmt | Meets all rqmts |

***Figure 5.41*** ADACS Candidate Architectures.[60]

[60] Note that while the design elements identified in the figure are representative of the design of the spacecraft shown, they do not necessarily represent the exact design of the actual spacecraft depicted in the figure.

Optimization, by definition, implies a change to the system. This change may be reflected in the requirements, which, as has been emphasized above, are organically connected to the implementation. Therefore, if optimization is to be done, the SDF provides a feedback to the Design and/or Requirements Development activities (Figure 5.42). While optimization can occur within virtually every element of the process, it is explicitly addressed here because the options being considered are "complete" in the sense that they have been defined to the point that technical, cost, and schedule criteria have been developed. At the other points in the process, each option is in the process of being developed so optimization occurs as a natural part of the Requirements Development and Synthesis activities through the iterations that occur.

### *B. Tailorability*

Tailorability is usually a topic of discussion when a generic system engineering process is to be applied to a specific development program. At this point, it is asserted that, to the level of decomposition provided above, the SDF is applied to each hierarchical level and for each development phase. While not all activities represent significant effort in every situation, generally all are performed to some level of fidelity. Therefore, tailoring is *not* achieved by changing the process. Rather, it is achieved by:

- Modulating the kinds and extent of documentation required
- Modulating the level of detail and the scope of the activities performed
- Prudent partitioning of the system hierarchy to effectively satisfy program needs.

## *V. The Integrated System Development Framework*

Figure 5.42 represents the fully-integrated, second-level decomposition of the basic SDF building block. It provides a two-tiered view of the SDF, defining the flow-down and feedback paths both within the same level and between levels of the system hierarchy.

Initial inputs are fed into the "Identified Work to Do" bucket. The first steps of the process involve both Work Generation activities as well as Rework Discovery activities. Because it is desirable to discover any problems with the requirements as early as possible, the "Analyze Requirements" activity is initiated. The Work Generation activities are also initiated. Progress is periodically assessed for convergence. If the Requirements Development activity is failing to converge, it may be the result of a discrepancy in the input requirements. A feedback to the "input" is shown in the figure, indicating that the customer should be consulted in order to review the input for consistency, clarity, and completeness. Nonconvergence may also be the result of less-than-adequate quality and/or insufficient effort focused on

discovering rework. As discussed in Chapter 4, these are the control mechanisms with the highest leverage for enabling convergence.

If the Requirements Development activity is converging, the output data is passed along to the Synthesis activity. A key decision must be made regarding the timing of the release of the output data to the Synthesis activity. Data released prematurely will result in more rework generated than data released in good condition, albeit later on the timeline. Once data is passed to the Synthesis activity, the Work Generation activities commence. A decision block in the process asks if supporting work at the next level down is necessary. If the answer is "yes," requirements are generated and released to the lower-level activities. If the answer is "no," the effort moves to the "Integrate and Plan Verification" activity, where input from lower-level activities is integrated into the system design. As the design development ensues, the Verify Design activities commence. Most of the rework discovered will likely be redone in the Synthesis activity. However, there is the potential that some of the rework discovered may be Requirements Development rework. There is, therefore, feedback to the Requirements Development activity via the RD Rework decision box. It is shown as the "Discovered Rework" box. Note also the inclusion of a box called "Forced Rework." This occurs in those cases where the output from the preceding activity is valid in and of itself, but cannot be implemented or is difficult to implement for some reason. This situation can arise, for example, when technologies included in the design are obsolete or are otherwise unobtainable. Or, due to technical, cost, and/or schedule concerns, a change in the requirements set or current design is called for. Both the "Forced Rework" and "Discovered Rework" feedback boxes also occur as feedback from the lower levels to the upper-level Synthesis activity as shown in Figure 5.42. If the Synthesis activity is not converging, the same controls of quality and rework discovery effort can be adjusted to facilitate convergence.

As discussed earlier in this chapter, the "Do Trades" activity commences as needed after two or more designs emerge from the Synthesis activity. If optimization is necessary, there is feedback to the Synthesis activity.

Figure 5.43 depicts the basic SDF building block along its one-level-down decomposition. It is provided in order to emphasize the consistency of the decompositions, as described in the foregoing.

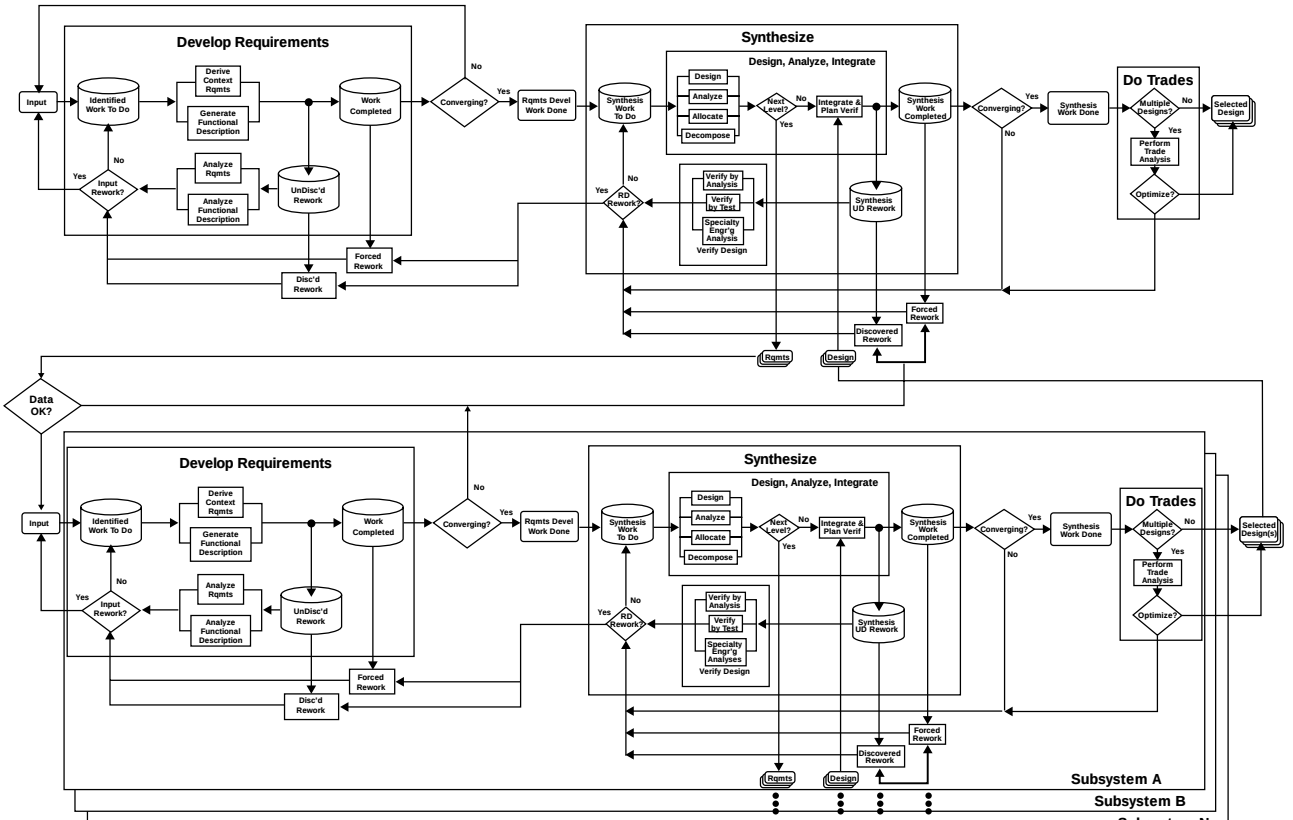
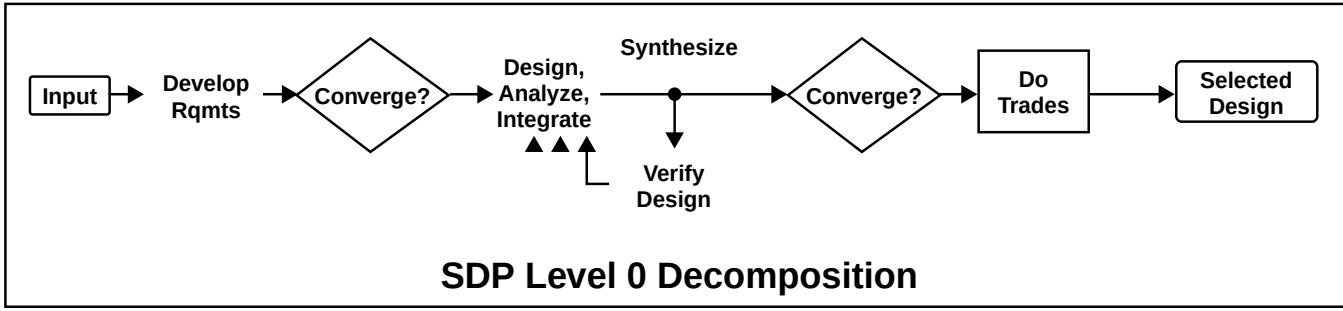



*Figure 5.42* The System Development Framework (SDF), Second Level Decomposition.



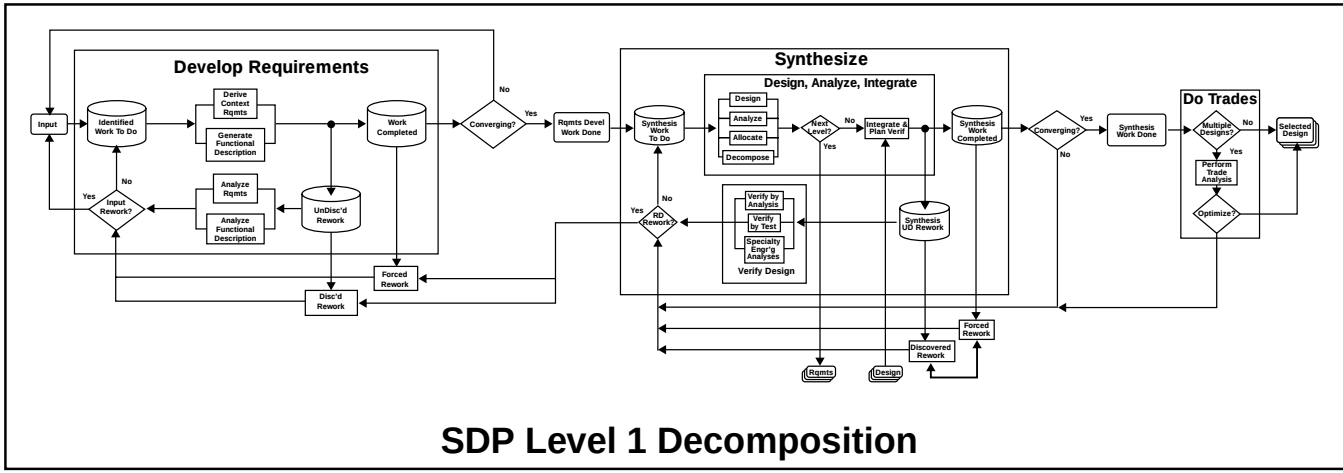




*Figure 5.43* SDF Decomposition Consistency.