SOFTWARE ENGINEERING LABORATORY SERIES SEL-87-003

GUIDELINES FOR APPLYING THE COMPOSITE SPECIFICATION MODEL (CSM)

JUNE 1987

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY SERIES

SEL-87-003

GUIDELINES FOR APPLYING THE COMPOSITE SPECIFICATION MODEL (CSM)

JUNE 1987

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/ Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The author of this document is William Agresti (Computer Sciences Corporation).

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

The Composite Specification Model (CSM) is an approach to representing software requirements. This document provides guidelines for applying CSM and developing each of the three descriptive views of the software: the contextual view, using entities and relationships; the dynamic view, using states and transitions; and the functional view, using data flows and processes. Using CSM results in a software specification document, which is outlined in this document.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

SECTION 1 - INTRODUCTION

The Composite Specification Model (CSM) addresses the problem of accurately and completely representing the requirements for a software system. CSM itself is not a software tool, although it is supported by several commercially available software products, as discussed in Section 2. CSM is a software method developed in the Software Engineering Laboratory (SEL) (Reference 1) to support the production of flight dynamics software at the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC). The objective of this document is to offer guidance and examples for those who are applying this method. This document is intended for individuals who want to use CSM for requirements specification. This section defines CSM and its use, discusses its origin and purpose, and outlines the remainder of the document.

1.1 WHAT IS CSM?

CSM is a representational medium for software requirements. It is one response to the widely recognized problem of developing effective techniques and practices for the earliest, predesign phases of the software development process. CSM differs from other requirements specification approaches, for example, structured analysis (Reference 2), in its use of multiple perspectives (as in Reference 3) for viewing requirements.

The rationale for the multiple views of CSM is that no single view of a complex object should be expected to be satisfactory. The most obvious analogy is with the multiple representations used in architecture. A scale model or artist's rendering of a building, which may be appropriate to show the planning commission, is not the representation needed by the plumbers or electricians. Considering the

number of relations present, software can be more complex than buildings. A strong case has been made elsewhere that the largest software systems are the most complex objects humans have built.

CSM currently uses the following three perspectives:

- Contextual view--The entities and relationships being modeled by the software
- Dynamic view--The behavior of the system over time
- Functional view--The transformation of input data flows into output data flows

The viewpoints should represent mutually orthogonal dimensions of system description. The analogy is to the representation of a three-dimensional object with a two-dimensional medium, for example, representing a statue on paper. One approach would show the orthogonal projections of the statue onto the x-y, y-z, and x-z planes. Similarly, CSM gives the "projection" of a software system onto the contextual, dynamic, and functional planes.

The three perspectives of CSM complement each other to provide a comprehensive understanding of a particular system. With a batch processing system, for example, the functional view may be the most meaningful as it depicts the transformations of input quantities through intermediate stages to yield output. With the requirements for an interactive software tool, the dynamic view may be the most valuable for communicating the intended operation of the system.

CSM is an inherently flexible medium with no limit on the number or nature of the viewpoints used. Currently, not all software requirements can be represented in the three existing views of CSM. Performance requirements, in particular, are not currently captured in CSM. With increased use, CSM may be expected to evolve, encompassing more than three

views or replacing the existing views with different ones. As a further indication of CSM's flexibility, the notation for capturing each perspective can be changed. Currently, the following notations are used:

- Entity-relationship diagrams (contextual)
- State-transition diagrams (dynamic)
- Data flow diagrams (functional)

While there is flexibility in the choice of notation, the clear preference is for graphical, nonnarrative approaches. This preference may be explained by noting that CSM was a byproduct of the SEL investigation of specification measures (Reference 4). That investigation concluded that the requirements documents typically used in the flight dynamics environment were not useful as a basis for defining specification measures. The documents use narrative text and mathematical equations to express software requirements. CSM was motivated by the need for a requirements representation that facilitates the definition of specification measures. Consequently, an integral part of the CSM philosophy is to use diagrams, lists, and tables rather than narrative text. The graphical and tabular representations yield simple counts that are the foundation for specification measures.

The three CSM views and their notations are described in Sections 2 through 5, respectively.

1.2 WHO USES CSM?

The potential users of CSM--and the audience for this document--are software developers during the requirements analysis phase of a project. Section 2 discusses how the use of CSM affects the work of the software development team.

1.3 OUTLINE OF THIS DOCUMENT

Section 2 gives an overview of CSM application and describes the dictionary and CSM tool support. The three views of CSM are described in Section 3 (contextual), Section 4 (dynamic), and Section 5 (functional). Section 6 shows the suggested format for the resulting CSM specification document.

SECTION 2 - OVERVIEW OF CSM USE

The issues and procedure involved in applying CSM are summarized in this section. Also discussed is the central role of the dictionary and availability of software tools to support CSM preparation.

2.1 THE DECISION TO USE CSM

Some key issues affecting the decision to use CSM are as follows:

- Experience base of prior CSM usage
- Characteristics of projects for which CSM may be most effective
- Expected cost of applying CSM
- Role of CSM in the development process

CSM is a recent SEL conception; experience with it is very limited. It was first used on the Earth Radiation Budget Satellite (ERBS) Yaw Maneuver Control Utility (YMCU), a FORTRAN system of 11,000 source lines of code (SLOC) (Reference 5). This initial use of CSM was in the context of the previously noted research program in specification measures and was unusual in that CSM was applied after the YMCU system was implemented. The second experience with CSM was consistent with its expected use during the requirements analysis phase to specify the requirements for the Gamma Ray Observatory (GRO) dynamics simulator in Ada[1] (GRODY) (Reference 6). This system is larger than the YMCU, with GRODY exceeding 100,000 SLOC. (The use of Ada accounts for some of the relatively high number of SLOC; the corresponding FORTRAN GRO dynamics simulator is 44,000 SLOC.)

---

[1]Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

In both cases, a CSM specification document, the product of using CSM, was written. Excerpts from both the YMCU/CSM (Reference 5) and the GRODY/CSM (Reference 6) specification documents are used to illustrate features of the CSM in later sections of this document.

Although limited, this CSM experience has provided some guidance on the nature of projects that would benefit most from using CSM. When the software developers do not have a strong legacy of experience with the application area of the software, CSM appears to be most helpful. In contrast, applications (such as attitude ground support systems in the flight dynamics environment) for which the developers have ample experience and reusable designs and code would not seem to benefit significantly from CSM. A key determinant of the expected benefit from CSM is whether the developers are producing a new design rather than using essentially the same high-level architecture from previous systems. The GRODY/CSM is an example of a development team's using CSM to help them understand the requirements and thus to create an original design for the dynamics simulator application.

Along with the expected benefit, the expected cost is a central issue in the decision to use CSM. The only available cost data shows approximately 9 staff-months of effort expended on the GRODY specification activity, leading to the GRODY/CSM document (Reference 6). It is not clear, however, how much of this effort was in excess of that which would have been expended if the development process did not use CSM. The experience with the GRODY/CSM indicates that much of the effort expended on CSM was essential to understand the system requirements and would have been attributed to either requirements analysis or preliminary design if CSM were not used. However, effort specifically to produce the GRODY/CSM document would, of course, not be incurred in a non-CSM project.

This discussion of the marginal effort of adding CSM to a project leads to the more general issue of the role of CSM in the development process. The flight dynamics software development process is well defined in References 7 and 8. CSM would be introduced during the requirements analysis phase. When this phase begins, the developers receive a preliminary version of the functional requirements and specifications document (FRSD). The FRSD describes the system requirements and provides supporting mathematical analysis. If CSM were used, it would complement, rather than replace, the FRSD. The GRODY/CSM specification (Reference 6) serves as an example of this complementary relationship: it refers to specific pages in the FRSD (Reference 9) containing algorithms that define how the input data flows are transformed into output data flows to support the functional view of CSM.

The principal influence of CSM on the development process is the creation of a new intermediate product: the CSM specification document. The typical flight dynamics development process includes the production of a requirement analysis summary report during the requirements analysis phase. Using CSM, the additional CSM specification document would be produced. The GRODY experience suggests that this extra product is very useful as a starting point for preliminary design.

2.2 HOW TO APPLY CSM

Table 2-1 summarizes the steps involved in applying CSM. The key feature is the gradual evolution of the CSM data base containing the three views and the CSM dictionary. At each stage in Table 2-1, more elements of the CSM specification become known. Ideally, storing and enhancing this evolving CSM data should be assisted by an automated software tool like one of those mentioned in Section 2.4.

Table 2-1. Steps To Follow in Applying CSM

Steps To Take	Contribution to Evolving CSM Specification
1. Acquire CSM software support tool (Section 2.4)	Establish project library to maintain elements of CSM specification
2. Analyze sources of requirements--documents and personnel (Section 3)	Begin building CSM data base on CSM support tool, entering the following: a. Contextual view (entities, relationships, attributes) b. Events (preliminary list) c. Dictionary (preliminary) with data items, entities, events, etc.
3. Prepare selected entity-relationship-attribute (ERA) diagrams to illustrate key entities and relationships (Section 3)	Add selected ERA diagrams; update contextual view
4. Prepare first the dynamic view, then the functional view; (see Sections 4 and 5)	Enter the following: a. Events (transitions) update b. States defined c. State transition diagrams d. Data flow diagrams e. Dictionary update f. Process specifications including references to separate requirements documents for details
5. Analyze interfaces among views (Section 6)	Add mappings that show the interfaces among views; verify consistency of dictionary for contextual and functional views
6. Trace progress from requirements documents to CSM specification (Section 6)	Add the following: a. Requirements traceability table b. List of requirements not addressed in CSM specification

The recommended CSM plan in Table 2-1 is based on the experience with GRODY. While the three views are generally approached in the order of contextual, dynamic, and functional, the table shows that there is a considerable interleaving of parts of each view. This continual shifting of viewpoint is a constructive feature, leading to successvie refinement of each view. For example, after some time is spent reaching an understanding of the required dynamic behavior of the system, the team should reexamine its lists of entities and relationships to determine if updates are needed. Some degree of iteration over steps 3 and 4 in Table 2-1 is necessary to continue refinement of each of the views (see Sections 3, 4, and 5).

Two aspects of CSM application seem clear. First, information produced by the functional view will generally be the largest and most detailed. Second, the determination of the system as being either more control-oriented (e.g., in embedded or real-time applications) or data-oriented (e.g., in file handling or transaction processing) will help to answer whether the dynamic view or functional view will be the most revealing and useful perspective.

2.3 CSM DICTIONARY

The CSM dictionary is similar to one used in structured analysis (Reference 2). However, for CSM, the dictionary supports all of the views by defining all of the names used in the CSM specification and the roles they fulfill. The CSM dictionary defines, for example, all data names used with data flow diagrams (DFDs) in the functional view and all attributes used in the contextual view. In practice, most data names appear in both views. For example, "fuel density" may be a data flow name on a DFD, while "density" may be an attribute of the entity "fuel" in the contextual view. The entry in the dictionary follows the names used in

the functional view, so "fuel density" would be defined. The contextual view lists, for attribute "density" of entity "fuel," that the corresponding dictionary entry is "fuel density."

The preparation instructions for a dictionary entry of a data name are as follows (Reference 10):

1. Data Name--Actual name used on a data flow diagram, in a process specification, or elsewhere in the dictionary (e.g., in the data element composition of another data dictionary entry).

2. Aliases--Other name(s) by which this dictionary entry is known. Each listed alias should also be entered in the dictionary.

3. Abridged Description--A concise, one-line statement about what the data name is or means.

4. Item Description or Composition--If the data name is a composite of other defined data names, this is the definition of the composition. The notation to be used to define the data's composition is as follows:

=	is composed of
+	AND
|	OR
**	enclosed text is commentary only
""	enclosed text is literal character string
[]	any one of the enclosed elements
{}	sets or iterations of the enclosed element(s)
ll{}ul	ll is the lower limit of sets/iterations; ul is the upper limit
ll{}	ll is the lower limit of sets/iterations; no upper limit

{}ul there is no lower limit; ul is the upper limit of sets/iterations

() enclosed element(s) is optional

Example:

```
day_of_week    =    [week_day|week_end_day]
week_day       =    [Monday|Tuesday|Wednesday|
                    Thursday|Friday]
week_end_day   =    [Saturday|Sunday]
phone_number   =    area_code+local_number
area_code =         3{digit}3
local_number   =    3{digit}3+4{digit}4
digit          =    [0|1|2|3|4|5|6|7|8|9]
```

If the data name is a data element (i.e., not composed of other defined data), enter the following:

a. Units. Include units of measure (e.g, meters, degrees) if meaningful. Otherwise, leave the units field blank.

b. Constant or Variable and associated Value. If the data item is a system constant, enter the constant's value. Otherwise, enter "variable." If the variable requires an initial value, specify it; otherwise, leave the initial value field blank.

c. Range. Enter the range of valid values for this data element (e.g., Sunday..Saturday, 1-1000, etc.)

d. Dimensions. If the data element is a vector or matrix, enter the appropriate dimensions and enter a particular coordinate system (if applicable) that is assumed in the values.

e. Data Type. Enter data type--either character, logical, integer, floating point, etc.

Such information should be maintained, in alphanumeric order, by a software support tool (Section 2.4) or at least in computer-readable form.

2.4 SOFTWARE TOOL SUPPORT FOR CSM

The desired CSM support would maintain all of the elements of the CSM specification, provide consistency checking, and support document production and maintenance. Commercially available software tools support the generation of CSM diagrams and the maintenance of the dictionary. Some examples are Excelerator (Reference 11), CASE 2000 (Reference 12), and Analyst Tool Kit (Reference 13). The GRODY/CSM specification (Reference 6) used Excelerator. Examples in Sections 3 and 5 show diagrams produced by Excelerator.

SECTION 3 - THE CONTEXTUAL VIEW: ENTITIES AND RELATIONSHIPS

The contextual view describes the objects and relationships that are being modeled by the software. This section discusses the rationale for this viewpoint, the elements that constitute it, and an example of extracting the contextual view from narrative text.

3.1 RATIONALE FOR CONTEXT MODELING

The contextual view describes the environment or information space in which the system will reside. Capturing the context of a system has been relatively undervalued as a tool for requirements engineering. A partial explanation may be that, for small programming exercises (e.g., sorting numbers or solving an equation), the background environment is either nonexistent or not a major concern; thus, there is no need to try to represent it. Many of the guidelines for addressing large system development have begun as attempts to "scale-up" the approaches (e.g., structured techniques) that were successful with small programs. Because the context is not important in understanding small programs, it has not been one of the techniques that investigators pursued in this scaling-up process.

With larger systems, the context or environment is a significant element in understanding the system's behavior. The software system is modeling some portion of an environment. When it is completed, the system will be taking its place in that environment, interacting with other objects (e.g., hardware, sensors, and other software) that are producing behavior in the environment. To describe its behavior relative to these other objects, the system must refer to specific attributes of the objects, for example, the mean radius of the Earth or the size of fuel tanks. Likewise, events in the environment (e.g., loss of signal, thruster on-time) may

trigger behavior by the system. Not all of the attributes or events in the environment are modeled by the system. In this sense, the model of the environment is not complete, nor is it ever intended to be complete. An individual attempting to understand the functioning and behavior of the software will be aided by seeing a representation of precisely those objects, attributes, and events that the system needs to know about in its environment.

This modeling approach is a natural introduction to object-oriented design (References 14 and 15), often used on Ada implementations like GRODY (Reference 16). Reference 14 recommends underlining the nouns and verbs in a requirements statement as a way of identifying objects and operations for object-oriented design. The contextual view is a form of object-oriented specification, identifying entities, relationships, and attributes that form the basis of design-level structures.

3.2 THE ENTITY-RELATIONSHIP-ATTRIBUTE APPROACH

CSM captures the contextual view through the entity-relationship-attribute (ERA) approach (Reference 17). Brief definitions and examples will be presented for each of the three ERA elements: entities, relationships, and attributes. Reference 17 contains a more thorough introduction.

Entities are identifiable objects in the environment. For example, the YMCU/CSM specified these entities: Earth, fuel, momentum wheel, pressurant, Scanwheel, spacecraft, Sun, surface model component, tank, thruster, and user (Reference 5). This list illustrates that entities often have some physical significance, like instruments or fuel tanks. Relationships are associations among entities and are

described as are relations in discrete mathematics (Reference 18). Examples of relationships are Earth-spacecraft and fuel-thruster-tank.

Information about entities and relationships is expressed by attributes. An attribute is a property or feature of the entity or relationship. For example, the entity "Earth" may have attributes of radius and magnetic field. Attributes correspond to data items that are listed in the CSM dictionary (Section 2).

A valuable conceptual feature of the ERA approach is the ability to associate attributes with relationships as well as with entities. As an example, the attribute "unit vector from the spacecraft center of mass to the center of Earth" is associated with the "Earth-spacecraft" relationship, not with the entities "Earth" or "spacecraft" alone. Table 3-1 shows the attributes defined for some of the entities and relationships of GRODY (Reference 6).

3.3 AN EXAMPLE

Figure 3-1 shows an excerpt from requirements statements that appeared in an FRSD. The contextual view regards the system being developed as modeling some real objects and relationships in the problem domain. The entities are underlined in Figure 3-1. The text also provides information about attributes of entities and relationships, that is, the characteristics of which the system must be aware. Attributes appear in bold italics in Figure 3-1. For example, volume is an attribute or feature of a tank. An implicit assumption in CSM is that entities and attributes would not appear in requirements statements unless they had a role in the activity of the system. For example, because the volume of the tank is mentioned, it is assumed that the system will need to know the volume at some point during the processing. By implication, other possible attributes of a tank

Table 3-1. Attributes Associated With Selected Entities and Relationships of GRODY (Reference 6) (1 of 2)

Entity	Attribute
Earth	Atmospheric-density-ref-data Earth-gravitational-constant Earth-radius Magnetic-field-model
Fine Sun Sensor (FSS)	Failure-indicator FSS-half-angles FSS-noise-parameters
Fixed-Head Star Tracker (FHST)	Circular-cone-vertex-angle Failure-indicator FHST-cutoff-angles FHST-noise-parameters FHST-occultation-angles fhst-temperature Intensity-threshold
Gamma Ray Observatory (GRO)	Body-inertia-tensor Body-moment-arm Center-of-mass Coefficient-diffuse-reflection Coefficient-specular-reflection Drag-coefficient
Earth - GRO	Aerodynamic-torque Argument-of-perigee Atmospheric-density Earth-unit-vector Eccentricity Epoch-time Geomagnetic-field Geomagnetic-field-torque Inclination Rt-ascension-ascending-node Semimajor-axis
Earth - Moon	Computed-earth-moon-vector
Earth - Sun	Computed-earth-sun-vector Earth-velocity
Earth - Magnetometer (TAM)	Tam-vector-measurement
FHST - GRO	Body-to-fhst-rotation-matrix
FHST - ground	FHST-command

Table 3-1. Attributes Associated With Selected Entities and Relationships of GRODY (Reference 6) (2 of 2)

Entity	Attribute
FHST - Onboard Computer (OBC)	FHST-command FHST-data FGST-status
FHST - stars	FHST-angular-coordinates FHST-star-intensity
FSS - GRO	body-to-FSS-rotation-matrix
FSS - Sun	FSS-alpha FSS-beta FSS-sun-present

The spacecraft has ***eight*** thrusters and ***two*** tanks with ***volume*** and ***location*** given in... [The system] will treat each tank individually and will have mathematical models to predict the spacecraft ***center of mass*** and ***moment of inertia*** as functions of the ***weight*** of fuel remaining in each tank.

Note: Entity is underlined; attribute is in bold italics.

Figure 3-1. Extracting Entities and Attributes From Requirements Statements

(e.g., its color or its material composition) need not be part of our contextual model because these other attributes are not discussed and, it is assumed, not a factor in the system's processing.

Table 3-2 shows the list of entities, relationships, and attributes extracted from Figure 3-1. It should be noted that some attributes, which may seem to refer to an entity, are associated with relationships. For example, the tank by itself has volume, but its location is a characteristic of the relationship between the tank and the spacecraft.

3.4 ERA DIAGRAMS

The contextual view can be depicted graphically using ERA diagrams (Reference 17). Figure 3-2 shows the ERA diagram corresponding to the information in Table 3-2. Figure 3-2 was drawn using the Excelerator support tool (Reference 11), one of the software systems (discussed in Section 2) that can support CSM. ERA diagrams can be helpful for visualizing the logical connectedness of entities and relationships. However, as Figure 3-2 suggests, the graph can become cluttered if more than a few entities are selected for display.

Table 3-2. ERA Approach to the Requirements in Figure 3-1

Entity	Attribute
thruster	number
tank	number, volume
fuel	
spacecraft	
Relationship	**Attribute**
fuel/tank	weight
fuel/tank/spacecraft	center of mass
	moment of inertia
spacecraft/tank	tank location

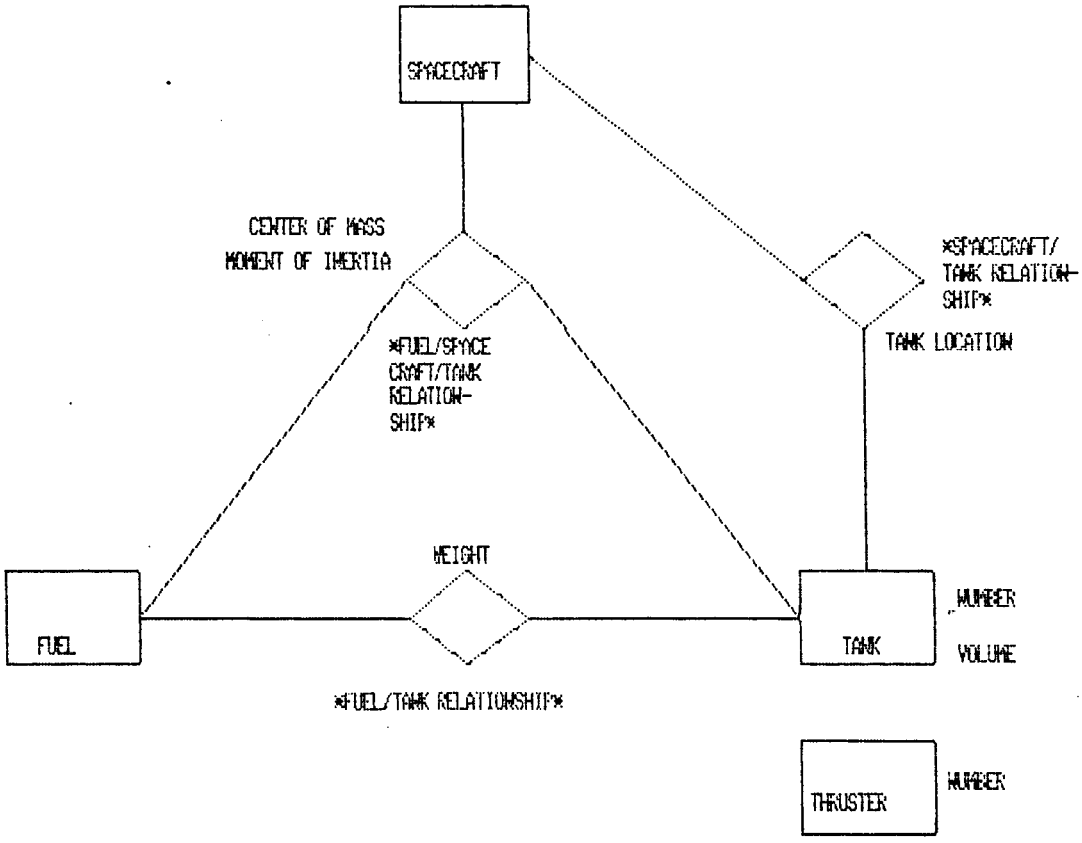

Figure 3-2. ERA Diagram of Information in Table 3-2

SECTION 4 - THE DYNAMIC VIEW: STATES AND TRANSITIONS

The dynamic view describes the behavior of a system over time. This section discusses the modeling of dynamic behavior in terms of states and transitions. Examples will help to illustrate the recasting of textual requirements into state-transition diagrams, which have wide use for system description (e.g., Reference 19).

4.1 MODELING WITH STATES AND TRANSITIONS

When a software system is executing, it can be considered to be moving through various states. A state is a record of the system's condition and characteristics at a particular time. A system changes state when an event occurs, altering some aspect of the system's condition. The identification of a system's states and events is a useful exercise in trying to understand the required dynamic behavior of the system. To be effective, this identification process must begin by taking a very high-level view of the system.

Only major events and states should be recognized initially. Each major state, for a large system, may be the aggregate of a wide range of system behavior. But, by grouping this behavior under a single state, the CSM user has performed a valuable simplification. For example, a system may have an initialization or start state during which the user of the system sets initial conditions, opens files, and writes initialization reports. In another example, the system may be in a "maneuver support" state, triggered by the event of the user commanding that a maneuver occur. This state name suggests that the application area will strongly determine the most reasonable assignment of states.

Major events may correspond to actions of the user or occurrences in the environment being monitored by the system. For example, the user may issue commands to start, restart, or stop processing, or the system may detect that transmission of a telemetry stream has stopped.

Listed below are the major events defined for the YMCU. These events marked state transitions in the CSM dynamic view of the YMCU:

- Maneuver start time
- End of integration step
- Yaw rate exceeding cutoff yaw rate
- Yaw angle equaling target yaw angle
- Yaw rate changing sign
- Pitch angle exceeding maximum pitch angle
- Roll angle exceeding maximum roll angle
- Thruster on-time
- Thruster off-time
- Maximum number of correction burns reached
- Yaw angle within epsilon of final yaw angle
- Maximum number of targeting phase iterations reached
- Maneuver stop time

4.2 STATE TRANSITION DIAGRAMS

A state transition diagram consists of only two symbols:

- A node (circle)--Used to represent a particular state of the system; the name in the circle is the system's state
- An arc (arrow)--Used to indicate the transition from one state to another resulting from the occurrence of an event; the name on the arc is the event causing the transition

Figure 4-1 is a representative state transition diagram and will be used to help explain the content of these diagrams. The diagram illustrates a simple bank Automatic Teller System (ATS) (Reference 10). It shows that the initial state of the ATS is called the IDLE state. The system is simply awaiting the insertion of a customer's access card. Upon detecting the insertion of a card, the ATS advances to the next state (VALIDATE USER), during which the user's access is validated. The user's card is checked for recognition by the ATS; the user is then prompted to specify his/her special access code or password. Upon successful verification of both, the system advances to the state in which it expects the user request (AWAIT REQUEST). If the user is not verified, the system returns to the IDLE state to await the next card insertion. Once in the AWAIT REQUEST state, the ATS expects a request valid for this user. A deposit, withdrawal, transfer funds, or balance request will cause the system to enter the PROCESS REQUEST state, during which the user's request is performed. Upon completion, the ATS returns to the AWAIT REQUEST state for the next request. Any unrecognized entry (e.g., entering a series of numbers) causes the system to remain in this state. A terminate request causes the ATS to return to the IDLE state, and the process begins again. The unlabeled arrow on the left-hand side of the IDLE state is used to indicate the state into which the system enters upon startup.

Figure 4-2 shows a high-level state transition diagram for the Trajectory Computation and Orbital Products System (TCOPS) (Reference 20). States are shown as nodes in the diagram, with transitions appearing as arcs between nodes. The node "TCOPS Readiness," without any incoming arcs, is obviously the initial state. Notice that the state transitions describe events occurring in the system or its environment.

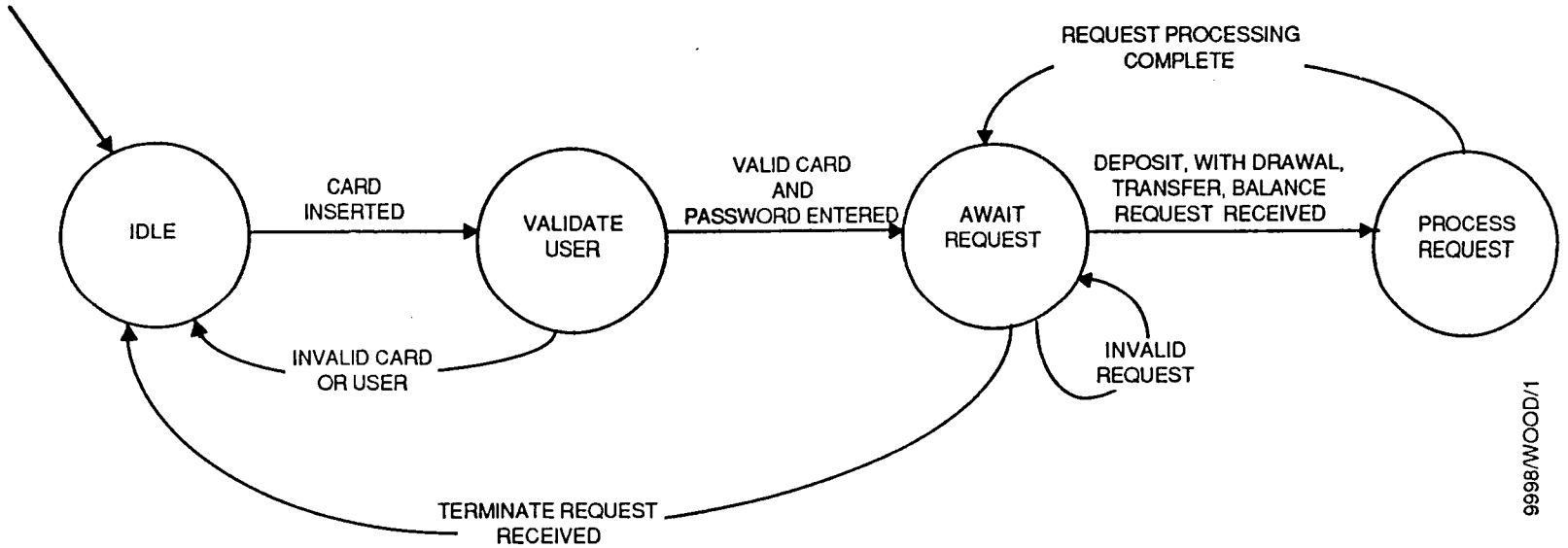

Figure 4-1. Automatic Teller System State Transition Diagram

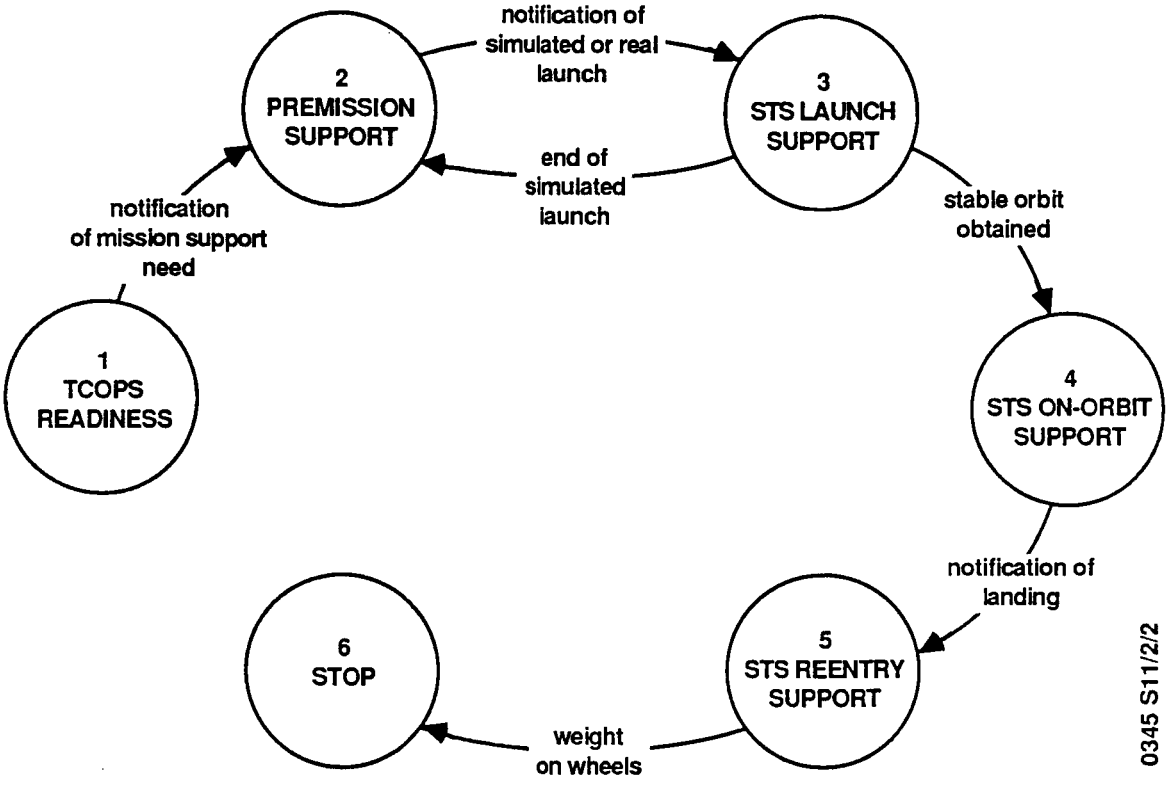

Figure 4-2. State Transition Diagram Showing TCOPS Space Transportation System (STS) Mission Support

Figures 4-3 and 4-4 show the dynamic view from the GRODY/CSM specification (Reference 6). In Figure 4-3, an arc without a source shows that "Presim Setup" is the initial node. The events causing state transitions are either explicit user commands or the simulation being completed.

Figure 4-4 addresses the difficulty of representing different levels of dynamic behavior. Figure 4-4 can be considered nested inside the "Dynamic Simulation" state of Figure 4-3. The transition "Start Command" takes the system into the major state "Dynamic Simulation" and simultaneously (Figure 4-4) into the secondary state "Standby Mode" within the "Dynamic Simulation" state. The "Restart Command" returns the system to the secondary state from which the "Interrupt Request" transition was taken. This latter interpretation is not apparent from the diagrams in Figures 4-3 and 4-4. If state transition diagrams are nested in some way, the CSM user must clarify all of the relationships (e.g., in supporting text) among diagrams and their states and transitions.

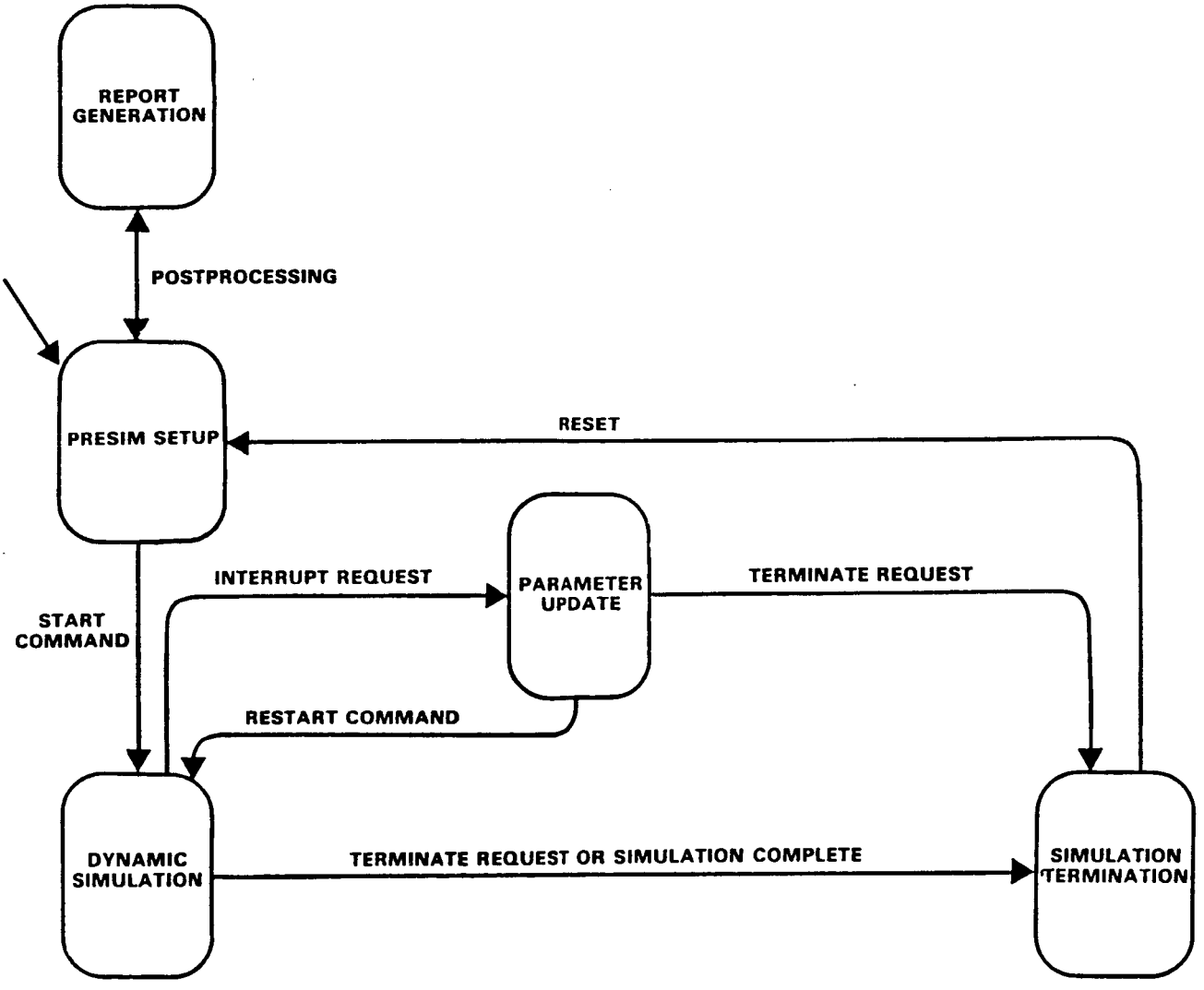

Figure 4-3. State Transition Diagram of GRODY

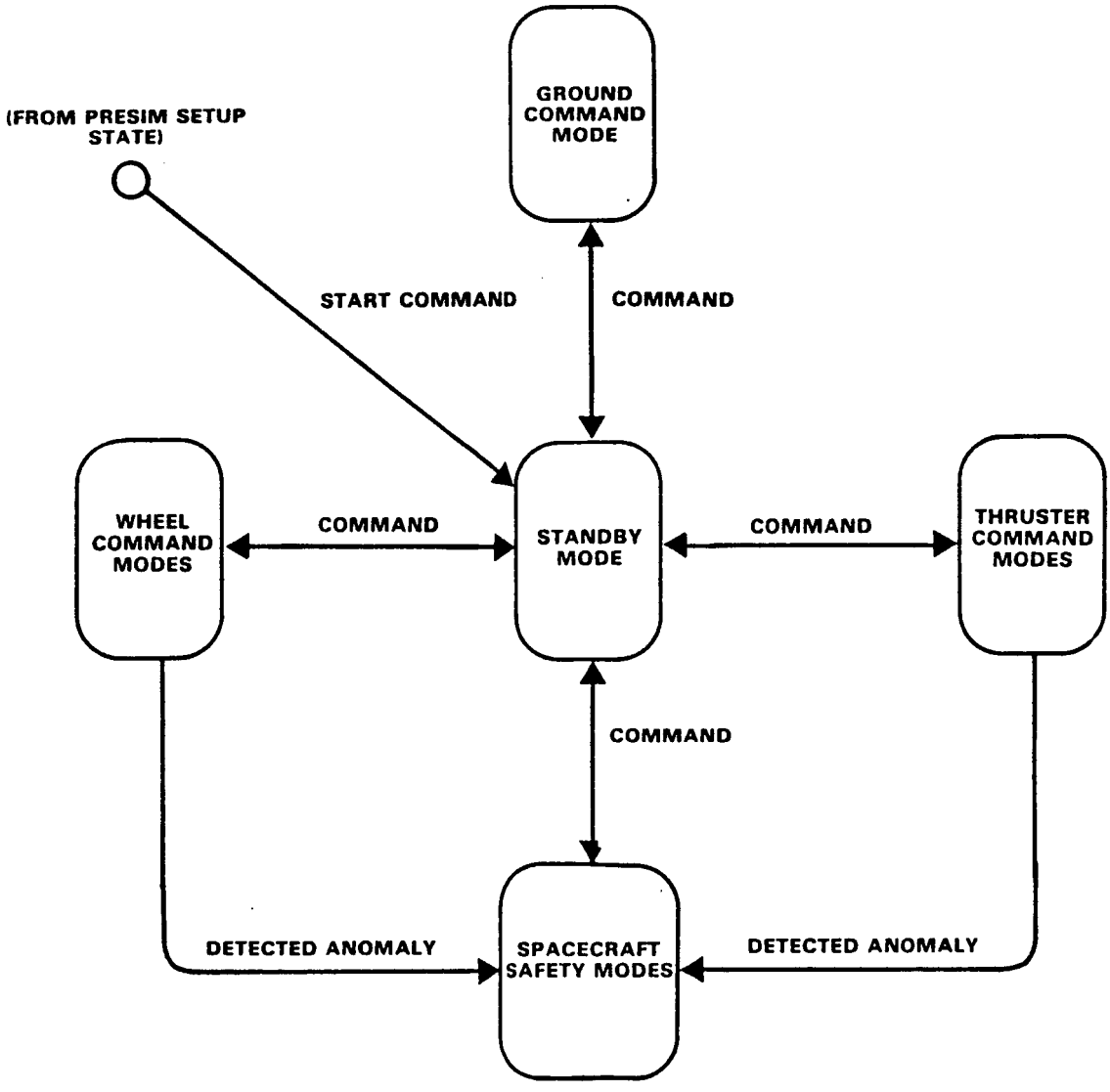

Figure 4-4. Decomposition of GRODY DYNAMIC SIMULATION State

SECTION 5 - THE FUNCTIONAL VIEW: PROCESSES AND DATA FLOW

The CSM functional view describes the successive transformation of input data flows to provide output data flows. CSM uses structured analysis methods (Reference 2) for data flow analysis and process descriptions. This section summarizes those methods. Reference 2 contains a more detailed description of structured analysis.

5.1 DATA FLOW DIAGRAMS

DFDs show the flow of data through a system and the transformations that data undergo. DFDs do not show flow of control. Four symbols are used in DFDs:

- Data flow (a named arc)
- Process (a node or "bubble")
- Data store (parallel straight lines)
- External entity (a box)

Figure 5-1 is a simple DFD (of a portion of a hypothetical Automatic Bank Teller System--the same example used in Section 4) that shows all four symbols (Reference 10). The data flow is a named arc, connecting processes, data stores, and/or external entities. The name appearing on the arc is that of a data item, record, file, or logical collection of any of these. In Figure 5-1, the data flow "savings account withdrawal request" is actually a logical group of data items (account identification and withdrawal amount). The direction of the arc indicates the direction of the data flow. The name must appear in the CSM dictionary. (The dictionary (Section 2) completely defines the type and composition of the data flow.) The name itself should be a noun.

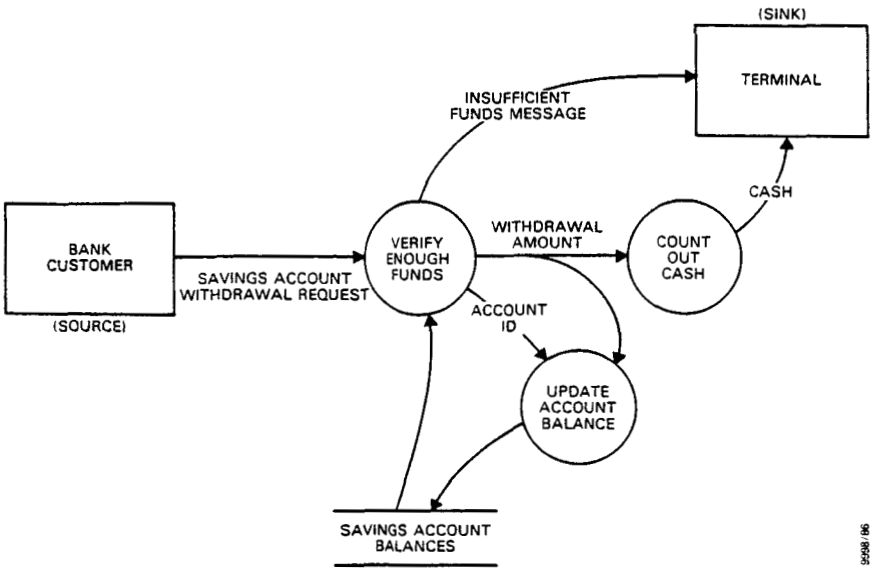

Figure 5-1. Sample Data Flow Diagram

The process is the transformation that converts input data flows to output data flows. The name of the process should indicate what is done to the input data to produce the output. This name is typically a verb followed by a noun.

The data store is a repository (temporary or permanent) of data. The external entity lies outside the scope of the system and is an orginator or receiver of data. Examples of external entities are people, organizations, terminals, or other hardware or software. Figure 5-1 shows that the external entity "bank customer" is a source, an originator of data flows, while "terminal" is a sink, an ultimate destination of data flows.

The context diagram is the topmost DFD for a given system (or subsystem) and should be the first diagram made during design. It shows at the system level "what do I have to produce?" (output) and "from what can I produce it?" (input). It consists simply of one process bubble and as many

system input and output data flows as are appropriate. Figure 5-2 is a context diagram.

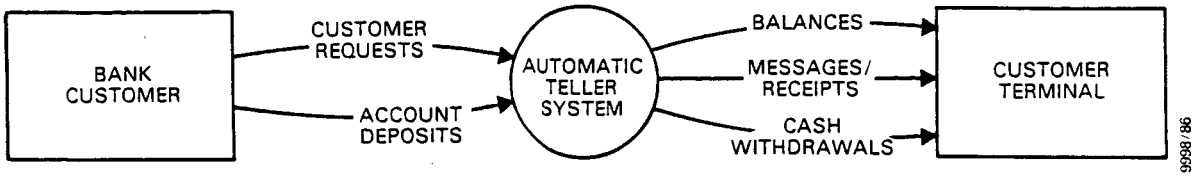

Figure 5-2. Sample Context Diagram

DFDs are hierarchical. The context diagram is decomposed into finer and finer detail in subsequent DFDs, which provide the user with as much or as little detail about the system as needed. Figure 5-3 illustrates this decomposition. Some general DFD guidelines are to use three to seven bubbles per page, and three to nine data flows per bubble; that is, do not try to put too much information into any one DFD. Reference 2 is an extensive introduction to DFDs.

5.2 PROCESS SPECIFICATIONS

When the DFDs have been completed, process specifications must be written for all primitive processes--that is, those not decomposed into lower levels. In Figure 5-3, the primitive processes are 1, 2.1, 2.2, 2.3.1, 2.3.2, 2.3.3, 3, 4.1, 4.2, 4.3, and 4.4.

The process specifications (mini-specs in Reference 2) describe in a program design language (PDL)-like notation how the input data flows are transformed into output data flows. The specification should use the same data flow names that appear in the DFD for that process. In this way, someone examining the DFDs and process specification will understand how the data flows from the diagram are processed. Figures 5-4 and 5-5 illustrate the desired consistency between DFD and process specification.

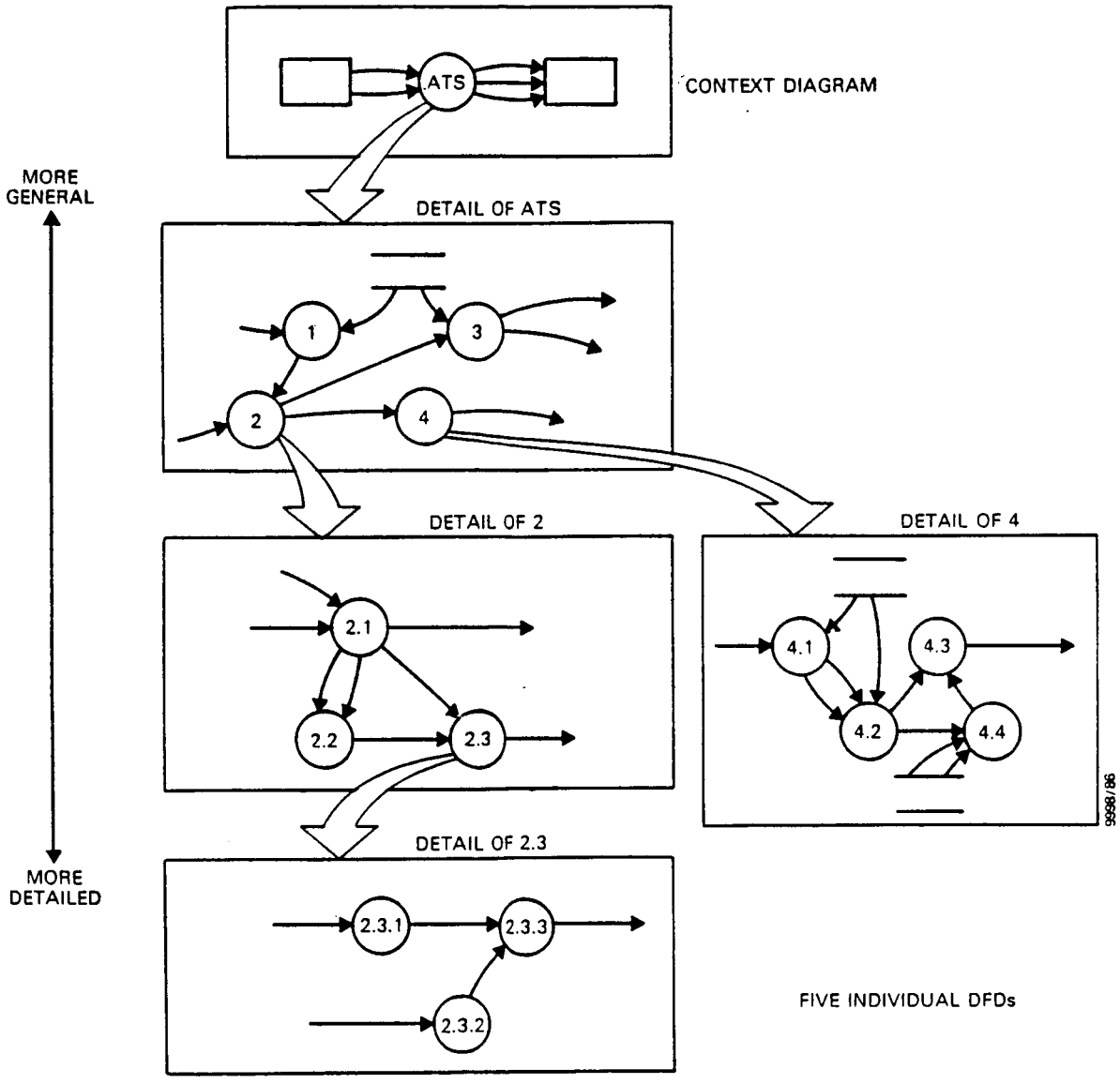

Figure 5-3. Hierarchy of Data Flow Diagrams

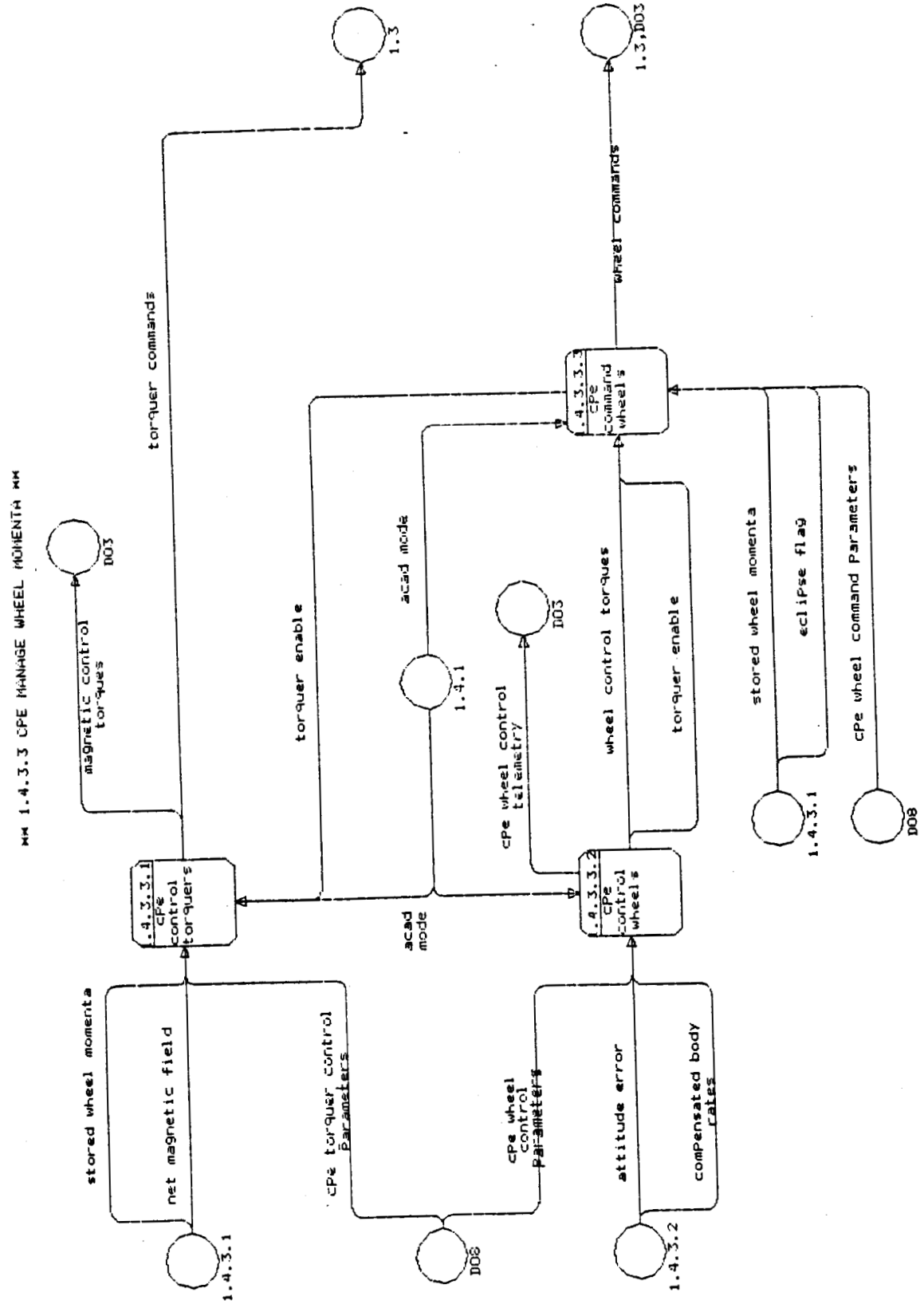

Figure 5-4. Sample GRODY DFD

Process 1.4.3.3.1 cpe control torquers

```
    If acad-mode is not safe hold or contingency Then
      For every 512 msecs Do
[a]      Adjust for a failed wheel if any
[b]      Transform stored-wheel-momenta to body coordinates
[c]      Compute the dumping momentum using cpe-torquer-
               control-parameters
[d]      Compute magnetic-control-torquers to dump this
               momentum using net-magnetic-field
[e]      Limit magnetic-control-torquers
         If torquer-enable and acad-mode is sun-referenced Then
           Send magnetic-control-torquers as torquer-commands
         Else
           Send torquer-commands to zero the torquer current
         Endif
      Enddo For
    Endif
```

Figure 5-5. Sample GRODY Process Description

Figure 5-4 is a DFD from the GRODY/CSM (Reference 6). Some notation in Figure 5-4 differs from the DFD overview of Section 5.1 because of the use of the Excelerator support software. Processes appear as rounded rectangles rather than circles as in Section 5.1. The small circles in Figure 5-4 are off-page connectors to processes, data stores, or external entities that are identified fully on higher level DFDs. The primitive process 1.4.3.3.1, "cpe control torques," has five input data flows:

- Stored wheel momenta
- Net magnetic field
- CPE torquer control parameters
- Torquer enable
- ACAD mode

and two output data flows:

- Magnetic control torquers
- Torquer commands

The 1.4.3.3.1 process specification is shown in Figure 5-5. It should be noted that the data flows are all mentioned in the specification. Further, hyphens are used to connect words comprising each data flow name to inform the reader that each name refers to a particular data flow that both appears on DFDs and is defined in the dictionary. The notations [a], [b],...,[e] in Figure 5-5 identify parts of the specification that need more explanation of the algorithm or particular method used to accomplish the processing. (Not shown is the part of the GRODY/CSM specification (Reference 6) that lists the reference document that corresponds to 1.4.3.3.1[a], ..., 1.4.3.3.1[e]). Figure 5-5 serves as an example of the complementary role, noted in Section 2, of the CSM specification and other reference documents (such as the FRSD) containing supporting mathematical equations. It should be noted that the process specification in Figure 5-5

includes a performance requirement ("For every 512 msecs..."). Although, as noted in Section 1, CSM does not yet integrate performance requirements into its system of multiple views, such requirements can be added textually as part of the process descriptions.

SECTION 6 - THE CSM PRODUCT: THE SOFTWARE SPECIFICATION

The product of applying CSM is a document: the CSM specification of the software system. This section discusses the suggested contents and organization of the CSM specification document.

Figure 6-1 shows the recommended outline of the CSM specification. The three views comprise the core of the specification. The recommended order of the views, shown in Figure 6-1, establishes first the objects and environment being modeled and leaves the most detailed (functional) view to the end.

The section on interfaces among views is recommended because it is a check on consistency and coverage.

The complete CSM specification should reinforce the philosophy of describing the system using diagrams, tables, and lists to the greatest extent instead of narrative text.

Section 1 - Introduction

- Overview of the project
- Statement on use of CSM for specification
- Relationship between this CSM specification and other requirements documents
- List of requirements not addressed by the CSM specification
- Outline of remainder of document

Section 2 - Contextual View

- List of entities and their attributes
- List of relationships and their attributes
- Statement that attributes are defined in the CSM dictionary (Appendix A)
- Selected entity-relationship-attribute diagrams to show key information

Section 3 - Dynamic View

- Lists and definitions of states and transitions (events in the system)
- State transition diagrams
- Supplementary text (as needed) to explain interpretation of multiple or nested state transition diagrams

Section 4 - Functional View

- List of data flow diagrams: name, number, and hierarchical structure
- List of process names and numbers

Figure 6-1. Outline of the CSM Specification (1 of 3)

- List of data store names and numbers
- List of external entity names and numbers
- Data flow diagrams in order corresponding to hierarchical structure, beginning with context diagram, level-0 DFD, and so on
- Process specifications in numerical order with annotations that map to more detailed (mathematical) references

Section 5 - Interfaces Between Views

- Contextual/dynamic interface
  - Table showing, for each state, what entities are modeled during that state
- Contextual/functional interface
  - Table showing, for each entity, the numbers of processes related to that entity
- Dynamic/functional interface
  - Table showing, for each state, the numbers of processes active (e.g., potentially executing) during that state

Appendix A - CSM Dictionary

- Shared by all three views
- Defines data items (attributes), entities, relationships, states, transitions, data stores, external entities, data flows, and processes

Figure 6-1. Outline of the CSM Specification (2 of 3)

Appendix B - Process Specification References

- Mapping the annotations used in process specifications to specific pages in other documents containing detailed equations or algorithms

Appendix C - Requirements Traceability Table

- Table showing how each requirement in an earlier document (e.g., the FRSD) maps to specific elements in the CSM specification, (e.g., numbered processes, entities, or transitions (events)

Figure 6-1. Outline of the CSM Specification (3 of 3)

GLOSSARY

ATS	Automatic Teller System
CSM	Composite Specification Model
DFD	data flow diagram
ERA	entity-relationship-attribute
ERBS	Earth Radiation Budget Satellite
FRSD	functional requirements and specifications document
GRO	Gamma Ray Observatory
GRODY	GRO Dynamics Simulator in Ada
GSFC	Goddard Space Flight Center
NASA	National Aeronautics and Space Administration
PDL	program design language
SEL	Software Engineering Laboratory
SLOC	source lines of code
TCOPS	Trajectory Computation and Orbital Products System
YMCU	Yaw Maneuver Control Utility

REFERENCES

1. Software Engineering Laboratory, SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

2. T. DeMarco, Structured Analysis and System Specification. New York: Yourdon, Inc., 1978

3. --, Controlling Software Projects. New York: Yourdon Press, 1982

4. Software Engineering Laboratory, SEL-84-003, Investigation of Specification Measures for the Software Engineering Laboratory (SEL), W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

5. Computer Sciences Corporation, A Case Study in Recasting Flight Dynamics Software Requirements Using the Composite Specification Model (CSM), W. Agresti, December 1984

6. --, CSC/TM-85/6108, Specification of the Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY), W. W. Agresti, E. Brinker, P. Lo, et. al., November 1985

7. Software Engineering Laboratory, SEL-84-001, Manager's Handbook for Software Development, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

8. --, SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

9. Computer Sciences Corporation, CSC/SD-85/6016, Gamma Ray Observatory (GRO) Dynamics Simulator Requirements and Mathematical Specifications, G. Coon, April 1985

10. Software Engineering Laboratory, SEL-86-001, Programmer's Handbook for Flight Dynamics Software Development, R. Wood and E. Edwards, March 1986

11. Index Technology Corporation, Excelerator Reference Guide, Release 1.11, 5 Cambridge Center, Cambridge, Massachusetts, 02142, 1984

12. Nastec Corporation, CASE 2000, 24681 Northwestern Highway, Southfield, Michigan, 48075, 1984

13. Yourdon Software Engineering Company, Analyst Toolkit, 1501 Broadway, New York, New York, 10036, 1985

14. G. Booch, Software Engineering with Ada. Menlo Park: Benjamin/Cummings, 1983

15. Software Engineering Laboratory, SEL-86-002, General Object-Oriented Software Development, E. Seidowitz and M. Stark, August 1986

16. Computer Sciences Corporation, CSC/SD-86/6013, GRO Dynamics Simulator in Ada (GRODY) Detailed Design Notebook, W. Agresti, E. Brinker, P. Lo, et al., March 1986

17. P. Chen, "The Entity-Relationship Model--Toward a Unified View of Data," ACM Transactions on Data Base Systems, March 1976

18. C. L. Liu, Elements of Discrete Mathematics. New York: McGraw-Hill, 1977

19. P. Gilbert, Software Design and Development. Palo Alto: Science Research Associates, 1983

20. Computer Sciences Corporation, CSC/SD-85/6708, Trajectory Computation and Orbital Products System (TCOPS) System Definition, July 1985

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-004, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-78-302, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3), W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve As a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wyckoff, July 1983

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-106, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, May 1985

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-203, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo, June 1984

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, August 1983

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1), W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-406, Annotated Bibliography of Software Engineering Laboratory Literature, D. N. Card, Q. L. Jordan, and F. E. McGarry, November 1986

SEL-83-001, An Approach to Software Cost Estimation, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, Measures and Metrics for Software Development, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983

SEL-83-006, Monitoring Software Development Through Dynamic Variables, C. W. Doerflinger, November 1983

SEL-83-007, Proceedings From the Eighth Annual Software Engineering Workshop, November 1983

SEL-84-001, Manager's Handbook for Software Development, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-84-002, Configuration Management and Control: Policies and Procedures, Q. L. Jordan and E. Edwards, December 1984

SEL-84-003, Investigation of Specification Measures for the Software Engineering Laboratory (SEL), W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, Proceedings From the Ninth Annual Software Engineering Workshop, November 1984

SEL-85-001, A Comparison of Software Verification Techniques, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team, R. Murphy and M. Stark, October 1985

SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985

SEL-85-004, Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics, R. W. Selby, Jr., May 1985

SEL-85-005, Software Verification and Testing, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, Proceedings From the Tenth Annual Software Engineering Workshop, December 1985

SEL-86-001, Programmer's Handbook for Flight Dynamics Software Development, R. Wood and E. Edwards, March 1986

SEL-86-002, General Object-Oriented Software Development, E. Seidewitz and M. Stark, August 1986

SEL-86-003, Flight Dynamics System Software Development Environment Tutorial, J. Buell and P. Myers, July 1986

SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986

SEL-86-005, Measuring Software Design, D. N. Card, October 1986

SEL-86-006, Proceedings From the Eleventh Annual Software Engineering Workshop, December 1986

SEL-87-001, Product Assurance Policies and Procedures for Flight Dynamics Software Development, S. Perry et al., March 1987

SEL-87-002, Ada Style Guide (Version 1.1), E. Seidewitz et al., May 1987

SEL-87-003, Guidelines for Applying the Composite Specification Model (CSM), W. W. Agresti, June 1987

SEL-RELATED LITERATURE

Agresti, W. W., Definition of Specification Measures for the Software Engineering Laboratory, Computer Sciences Corporation, CSC/TM-84/6085, June 1984

[4]Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," Proceedings of the First Pan-Pacific Computer Conference, September 1985

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," Proceedings of the International Computer Software and Applications Conference, October 1985

[4]Basili, V. R., and D. Patnaik, A Study on Fault Prediction and Reliability Assessment in the SEL Environment, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," Communications of the ACM, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P--A Prototype Expert System for Software Engineering Management," Proceedings of the IEEE/MITRE Expert Systems in Government Symposium, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. New York: IEEE Computer Society Press, 1979

[2]Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environments's Characteristic Software Metric Set," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

Basili, V. R., and R. W. Selby, Jr., Comparing the Effectiveness of Software Testing Strategies, University of Maryland, Technical Report TR-1501, May 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, July 1986

[2]Basili, V.R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: IEEE Computer Society Press, 1978

[3]Card, D. N., "A Software Technology Evaluation Program," Annais do XVIII Congresso Nacional de Informatica, October 1985

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," IEEE Transactions on Software Engineering, February 1986

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," ACM Software Engineering Notes, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," Proceedings of the Seventh International Computer Software and Applications Conference. New York: IEEE Computer Society Press, 1983

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," Proceedings of the Hawaiian International Conference on System Sciences, January 1985

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," Proceedings of the Eighth International Computer Software and Applications Conference, November 1984

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press, 1985

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," Proceedings of the First International Symposium on Ada for the NASA Space Station, June 1986

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," IEEE Transactions on Software Engineering, February 1985

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

NOTES:

[1]This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume I, July 1982.

[2]This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.

[3]This article also appears in SEL-85-003, Collected Software Engineering Papers: Volume III, November 1985.

[4]This article also appears in SEL-86-004, Collected Software Engineering Papers: Volume IV, November 1986.