Systems Engineering & Software Engineering, Contrasts and Synergism

David W. Oliver
Model Based Systems, Inc.
133 Ashdown Road, Ballston Lake, N.Y. 12019

Abstract

Systems engineers and software engineers work together in the development of modern complex systems. The two engineering cultures, the concepts, and the best practices have developed independently over four decades. Notations and naming conventions for the same things are often different. Yet the efficient exchange of engineering information and wisdom between the two professions is important to the successful development of large complex systems.

The present record of success for complex computer intensive systems is that for every six systems put in operation two are canceled; on the average projects are 50% over schedule; and three quarters are failures that do not function as intended or are not used at all, (Gibbs 1994). Incomplete specifications, ambiguous specifications, and misunderstood specifications are a major contributor to these problems. Development of rigorous specifications that match user needs is critical.

The need for synergism between systems engineering which develops specifications to meet user need and software engineering is particularly important because software portions of systems are increasingly complex and are often being coded in countries far from the country where the system is defined and utilized.

1 Introduction

This paper is a step toward contrasting systems and software methodologies and identifying synergistic elements of both engineering fields. This is accomplished by defining a process meta-model and related information models for each step in the process. Software and systems engineering methodologies are shown to be particular cases of the process meta-model.

Though the meta-model must be expressed in some notation, it is re-expressible in a number of notations and can be tailored to particular methodologies and notations. It is tailored to particular methodologies by removing steps that are not wanted in that methodology, or by transforming independently concurrent steps into particular desired sequences. This approach results in identification of the differences between methodologies and it establishes features that are useful to exchange between methodologies. Suggestions are made for transfer of wisdom between the two fields.

The approach of this paper separates process and methodology, (Martin 1994). The meta-process model captures what the engineer does as a set of work tasks, independent of the details of how the work is to be done. Some of the tasks are inherently sequential. Changing the order changes the result. Some tasks are inherently independently concurrent. They may be done concurrently, or in any ordered sequence since physical reality does not require any particular ordering. A meta-process description captures the intrinsic behavior of engineers in doing their work. It is highly tailorable.

A methodology is a prescription of exactly how to carry out each engineering step, how to express the information, and how to order the steps. It is a set of directives that can be followed by a large group so that they will be able to synchronize their activities, and will express their results with the same collections of information, views, and notations.

2 Some Historical Developments

Any history of an engineering field is both subjective and controversial. In this section a few of the significant past events are summarized to illustrate parallel origins in systems engineering and software engineering. Systems engineering, software development with structured analysis methods, and software development with objects have their origins in roughly the same time period, the 1960's and 1970's.

1.1 Systems Engineering

Systems engineering practices owe much to the work at TRW that developed a structured engineering approach for ballistic missile development. The definition of functional flow block diagrams (FFBD's) traces to this work and standards emerged in the 1960's. The FFBD's explicitly modeled the ordering of functions by control operations which was difficult to describe in English text. The practices have been described well by Blanchard and Fabrycky, (Blanchard and Fabrycky 1990). Over many years the methodology was extended by Alford

0-7803-2531-1/95/$4.00

who introduced valuable techniques of rigorous modeling, including an executable behavior model, (Alford 1977), (Alford 1992). He was a leader in the development of tools to automate the process, SREM, DCDS, and RDD-100.

1.2 Software Engineering

Structured analysis for software engineering traces to early work at IBM Laboratories, (Stevens, Meyers, and Constantine 1974), and was subsequently published as a methodology, (Yourdon 1989). Refinements that deal more completely with issues of structure were added, (Hatley and Pirbhai 1987).

Object oriented practices have their roots in the development of abstract data types and the introduction of classes to programming languages, such as Simula67, (Dahl, Dijkstra, and Hoare 1972). The application of database set concepts to data structures introduced ideas of relationships among data entities, (Bachman and Williams 1964). The object approaches have more recently been extended to analysis and design of software by a number of authors and graphic languages have been defined. One of the several approaches will be discussed in this paper, Object Modeling Technique, OMT, (Rumbaugh et al. 1991).

Most of the methodologists are active in publishing and updating their methodologies. Any comparisons made are likely to be out of date with respect to the recent version of methodology. It is useful, however, to examine how these methodologies are related to one another and to attempt to find a meta-model which is a generalization of the collection of methodologies. The reason that a successful meta-model may be possible is simply that the engineering work to define and specify a complex system requires that certain engineering steps be taken and certain information captured.

2 A Systems - Software Contrast

The culture, points of view, and practices of disciplines arise from the basic problems being solved as the fields emerge. For systems engineering the emergent problem was to establish the value of a system to a users and to develop the specifications needed by designers to implement that system. The system engineer worked from the top down and provided specifications for implementation largely in english text. Great care was taken in considering alternative things or objects to use and alternative allocations of behavior to the things. Trade-off against well defined criteria were used to find a near optimal solution among the alternatives. Risks involved were identified and appropriate measures taken.

The systems engineering product left much of the implementation detail for the designer. However, the specification to the designer was ambiguous rather than rigorous because it was written in English language.

In the case of software, the emergent problem was to rigorously express the complex machine code that would cause the computer to do what was wanted when implemented code executed on the computer. Optimization was frequently considered after the code was running. The product contained the implementation detail in a rigorous form.

The emergent software problem has developed bottom up. Developments in the field have produced user friendly higher order representations of the information with generators or compilers to create the lower level details. They have produced a practice of separating the specification of a module from its implementation, information hiding, so that the implementation may be modified without affecting other modules of an application.

3 Systems - Software Synergism

The major viewpoint of this paper is that the two fields - systems engineering and software engineering - have much to gain from each other. The basic knowledge and the low cost computer power to do this are available.

Software engineering practices may benefit by infusing systems practices of:

- Assessing value to the customer
- Using a hierarchical process, tiers of development
- Considering alternative objects and alternative allocations of behavior to the objects
- Defining optimization criteria for trade-off
- Applying trade-off to select a near optimal solution from alternatives at each tier of development
- Considering risk, validation, and sequential builds at each tier of development

Systems engineering practices may benefit by infusing software best practices of:

- Rigorous modeling
- Generators to rigorously transform information
- Full use of abstraction
- Application of information hiding.

4 Basic Modeling Concepts - Hierarchy, Behavior

In this paper the static structural views of how things are built will be described using Object Modeling Technique, (Rumbaugh et al. 1991). The dynamic description of processes, what is to be done by things, will be described using Functional Flow Block Diagrams, (MIL-STD-499 1968), (Blanchard and Fabrycky 1990), and Data Flow Diagrams, (Yourdon 1989), as two views of behavior.

A complete model of the semantics needed and a list of the possible views of behavior and structure has been de-

scribed elsewhere, (Jackson and Oliver 1994), (Oliver 1994a), (Oliver 1994b).

An object model for hierarchy is shown in Figure 1. In that figure each box represents a class of real objects. The diamond is a symbol for aggregation, a parts list. The dark circle means "many of". The upper box in the figure with a recursive aggregation says that a System is built from Systems.

The straight lines connecting classes indicate a relationship and are labeled with a relationship role name that has a pointer associated to show the direction of the relationship named. The black dot means many, i.e. Subject System interfaces with many External Systems.The upper part of Figure 1. says that a System has a System Role, and three roles are identified by the specialization (classificati9on) shown with a triangle symbol. The System Role depends upon the tier of decomposition and how the system is being considered by engineers. For example, the engine of a car has the role Subject System for the engineers specifying and designing the engine. The engine has the role External System to the engineers specifying and designing the drive train. In this role the engine stimulates the drive train and receives responses from it.The engine has the role Component to the engineers who are specifying and designing the entire car.

The relationship between External System and Subject System is often called an interface and is sometimes promoted to the status of a real world class. Clear interface design and specification is critical in large system projects. This is particularly important where separate teams develop different components of the system. The interface specification is critical where components from any of a variety of producers may be connected to the system as in the telecommunications industry. Engineers developing the Subject System examine the interactions between it and the External Systems to understand what the Subject System must do, its behavior.

Often one or more Components will be handed off to a team of engineers who will deal only with the development of that Component For example, engine designers will take on the task of engine development. To simplify their work they re-interpret the Component, engine, as a Subject System and consider only the other Components of car with which they must interface as External Systems.

Any of the things one may be developing is part of a huge parts or aggregation hierarchy which starts with the universe and ends with sub-atomic particles. A particular thing may be represented in a Context View which is built from the thing as a Subject System and the External Systems with which it interfaces. It may be represented alternatively in an Assembly View which shows the Components from which it is assembled. Finally, it may be represented in an Assembly View which shows it as one of the Components that are assembled to make a part at the next higher tier of the parts tree.

The procedure of decomposing Systems into Components, and of re-interpreting Components as Systems defines tiers of decomposition.

- Each of these tiers corresponds to what is frequently referred to as a phase of development.
- The engineering work, or process, at each tier is the same and can be described in a behavior meta-model.

Typical phases are often represented in a waterfall picture, (Boehm 1986). (This is a picture, not a model because it does not have a well defined semantics.) These phases are listed in Table 1., which identifies the traditional Phase names, and for that Phase the Subject System, External Systems, and Components.

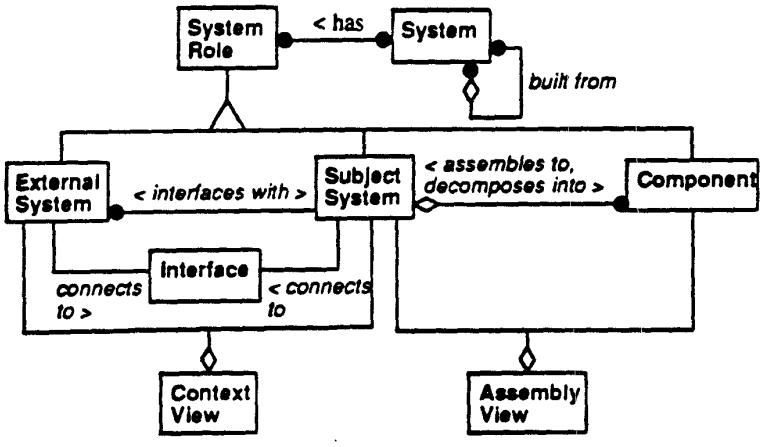

Figure 1. An Object Meta-model for System Hierarchy

In the systems Concept Analysis phase or tier the Subject System being modeled is the customer's business with our product imbedded in it. The key information to be obtained is the value of the product to the customer. Trade-offs between alternatives are made among product features (behavior) and structure until high value is found for the customer. The product must be feasible in the sense that it can be built at a cost that is consistent with value to the customer and a profit is possible.

In the systems Requirements Analysis phase the Subject System is our product. The key information to be obtained is a specification of exactly what our product will do, its behavior, and thoroughly established cost and performance targets. The major segments or subsystems of the product are identified.

In the Systems Design phase the major architecture and partitioning of the product is confirmed. The behavior of each segment is defined with targets for cost and performance for each product segment. Trade-off is performed among alternative architectures. The trade-off decisions are based on meeting criteria at the System level. The System Design phase ends when the decomposition reaches the point that hardware, software, and operator/user behavior and performance can be separately specified. In very large systems the Systems Design phase may involve several tiers of decomposition: System into Segments, Segments into Sub-Segments, etc.

Some modern methodologies support the concept of delaying the decision to separate hardware from software to the lowest level of decomposition possible, termed co-design. It is important that at all tiers of decomposition the criteria used for making a decision are applied at the top tier to quantify their impact on product performance and value to the user. The alternatives need to be rolled up to system level for trade-off.

When the separation has been made between hardware and software, it is possible and traditional for engineers in the individual disciplines to continue the decomposition and specification of their components. The specification process ends at Code and Manufacture with specifications sufficiently detailed for coding and manufacture. In modern automated practice many of these specifications are computer executable and can be interpreted by code generators or generators of manufacturing tool instructions.

Table 1. includes Domain Analysis as a phase at a higher tier than Concept Analysis. It is the phase that is critical to obtain reuse.In Domain Analysis one considers our product in a collection of businesses or in a business as that business matures over time. It involves modeling multiple businesses, looking for value of our product to all of them, and identifying common components of our product which are useful to all of the businesses.

Table 1:

Phase or Tier	Subject System	External Systems	Components
Domain Analysis	Collection of Customer Business	Customer suppliers and his customers	Our product segments which can be reused
Concept Analysis	Customer Business with our Product	Customer suppliers and his customers	Our product Segments, Customer Business Segments
Requirement Analysis	Our Product	Customer Business Segments	Our Product Segments
Systems Design (several tiers)	Our Product Segments	Customer and Our Segments	Our Product Sub Segments
continue	until HW/	SW/people	separate
HW &SW Requirements Analysis	Our HW & SW Segments	Other HW & SW Segments	HW & SW Sub Segments
Preliminary Design	HW&SW Sub Segments	Other HW & SW Sub Segments	Detailed Design Segments

At each of these tiers there is engineering work to be done to define the context of the Subject System with the External Systems, to define the behavior of the Subject System, and to allocate that behavior to Components using Trade-off to find a near optimal solution. The engineering work at each tier is the same. An implementation plan is made at each tier considering risk, validation, etc.

The description of the engineering work is a process which can be described as an executable behavior. A model of a process is a behavior. Figure 3. shows the semantic entities that need to be captured to express behavior.The structure operations apply to behavior and to the entities from which it is composed.This makes the description of behavior hierarchical.

Behavior is built from two major pieces, functions which receive input and transform them into outputs, and the items which are inputs/outputs. Some of the items trigger functions to start or to stop. The functions are ordered by control operations. A minimum set of control operations is shown in Figure 3.

Figure 3. shows that the concurrency control operation can be represented with parallel functions or by utilizing state, which defines the union of concurrent functions as a state. This description of behavior has been discussed elsewhere, (Oliver 1994a), (Oliver 1994b). In the models that follow, this paper will use primarily one view of behavior, the Functional Flow Block Diagram which

engineers think about objects while developing behavior, and they think about scenarios as they define objects. The alternative selections of objects and the alternatives in mapping behavior to objects results in not one, but a set of possible architectures to meet the requirements. In this work they are guided by knowledge of effectiveness measures.

The seven Core Steps are consistent with published treatments of systems engineering, (Blanchard and Fabrycky 1990). Note that the functional flow block diagrams are only a partial view of the process behavior because they do not include the I/O for each function.

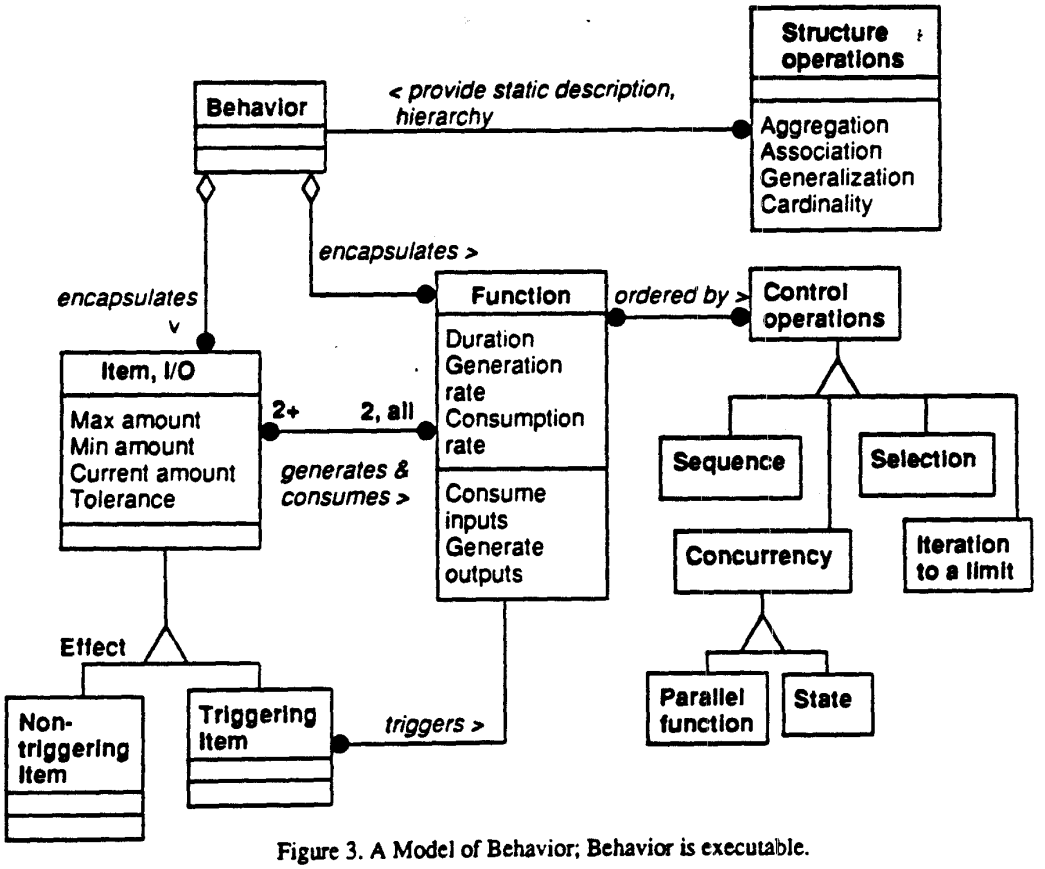

Figure 3. A Model of Behavior; Behavior is executable.

captures the functions as ordered by control operations, but omits the items, the input/outputs. This is done to simplify the figures and for clarity

5 A Process Meta-Model

A process meta-model for all of the tiers is shown in Figure 4., (Oliver 1993). Note that three of the major steps are concurrent, but they are not independent Some methodologies put these steps in a particular order. Practicing

The meta-model has been developed so that existing methodologies can be represented as special cases, found by omitting steps, or using particular sequences of the concurrent steps. This paper considers only some of the methodologies as they have been published at a particular time. Because the methodologists are constantly updating their work, it is beyond the scope of this paper to relate the current state of methodologies to the meta-model.

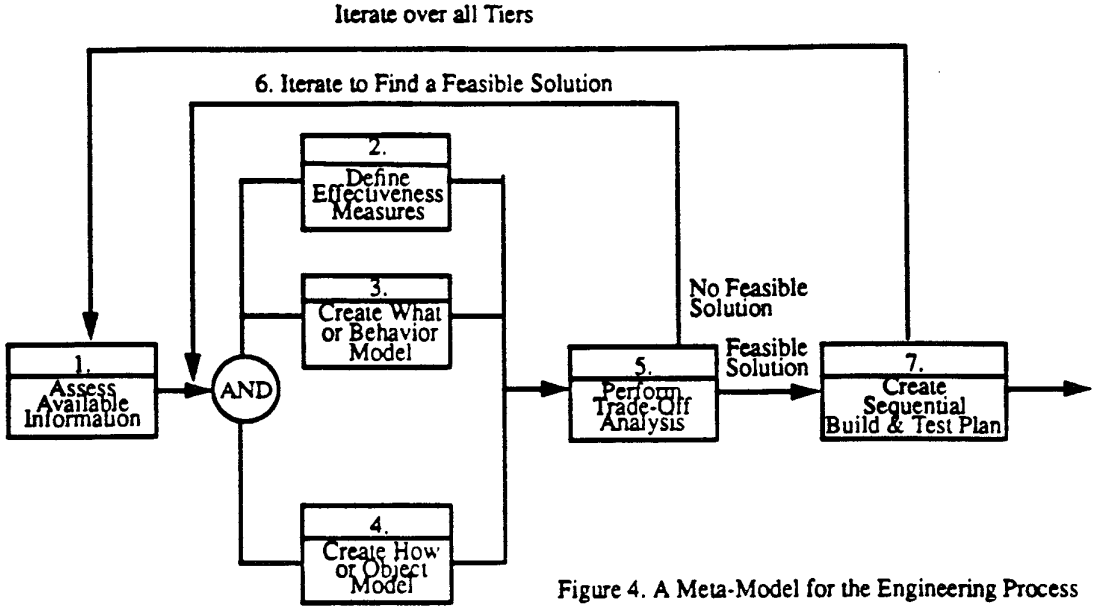

Figure 4. A Meta-Model for the Engineering Process

Information models have been developed for each of the core engineering steps and subjected to review, (Jackson and Oliver 1994).

6 Systems Engineering

The systems engineering process is represented accurately by the meta-model of Figure 4. The meta-model abstracts the descriptions in textbooks, (Blanchard and Fabrycky 1990), and instruction from the Defense Management College.

The meta-process sets great importance upon the capture of effectiveness measures. They are a small subset, frequently less than ten, of the many requirements a large system must meet. These are the criteria upon which system optimization is based. They encapsulate the driving business and user realities which the system must meet. Mathematically they serve as regularization functions to transform an ill posed problem into an optimization problem for which a near optimal solution can be found. They bear a strong relationship to the cost functions used in calculus of variations, optimal control theory, and regression analysis.

The effectiveness measures also join the inputs from management, marketing, and customers with engineering. In the development of systems of all kinds it is extremely important to capture this information and obtain a consensus. Otherwise continuous requirements change may stalk the project and drive up the cost. The finished system may not be something customers really wants and will purchase. Very early the Domain or Concept tier trade-off establishes much of the system cost and the system match to the marketplace.

A second hall mark of the systems process has been the development of alternative architectures for trade-off by examining different ways of allocating the desired behavior to alternative objects. The evaluation of the resulting alternatives involves measuring or simulating performance, showing an architecture meets requirements to be feasible, and optimizing the system by using the effectiveness measures to choose between the identified alternatives which are feasible.

Much of the information traditionally captured by systems engineers has been in text. The advantage of this is that text and pictures have great impact and persuasiveness when well prepared and presented. Such pictures and the status of system effectiveness measures are valuable in communication upward toward management. The difficulty is that text and pictures are ambiguous when used to specify the details of a complex system. Rigorous models which can be transformed and generated in alternative forms are needed for communication to other engineers.

Systems engineering practice has lacked the use of rigorous models to augment, not replace, the use of text and pictures though methodologies and tools have been developed.

One major relationship that has been largely missing in systems engineering practice is that of classification, generalization/specialization. It is critical for reuse, for consistent definitions, and for rigorous communication with software engineering.

7 Yourdon SA/SD

A FFBD for SA/SD is given in Figure 5. Steps 2., 5., and 7. are largely omitted in the methodology. It is a methodology for representation primarily, and does not consider optimization except in the later stages of implementation where performance is tested.

The views of behavior that are prescribed are data flow diagrams and control flow diagrams. Control is not shown unless it is necessary to complement the data flow diagrams. This convention with respect to control makes it difficult to automate execution of the data flow, control flow information must be interpreted by an engineer.

The concept of tiers of decomposition are present but they are applied to the behavior which is decomposed to lowest level without trade-off. The resulting essential model is then transformed into a set of structure charts

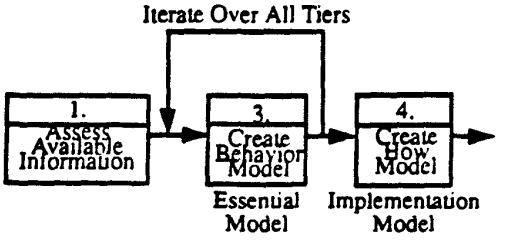

Figure 5. The Core Set of Steps in Structured Analysis

which define a calling tree among modules. This is an incomplete model of structure.

8 Hatley-Pirbhai

The Hatley-Pirbhai methodology uses the same major three steps as SA/SD. However the description of structure is much more complete. It is called an architecture model rather than an object model. It incorporates concepts of components and their hierarchical decomposition. It incorporates the associations among the components, i.e., their interconnection. It distinguishes between the concepts of assembly and context. Optimization and trade-off are not considered.

9 Object Modeling Technique, OMT

Object Modeling Technique was developed to consider problems that were free from timing issues, for non-real time systems. For these systems it is necessary to represent the sequences in which functions act, but detailed control modeling and timing are not required. It is interesting that the views and notations selected are not only semantically complete and executable for the development of code and database schema, but that they can represent behavior and be applied to real time systems. This is a major advance over the earlier software methodologies.

Like the two methodologies discussed above, OMT does not consider process steps for trade-off and optimization: Define Effectiveness Measures, Perform Trade-off Analysis, and Create Sequential Build and Test Plan. It utilizes the same three major process steps as SA/SD and Hatley-Pirbhai, organized differently, and a more complete set of information representations is used. The tiers of decomposition for software are recognized, and the object models lack hierarchy.

The process model for OMT is shown in Figure 6. The first step is to Assess the Available Information. This is followed by Create an Object Model, which produces a static model. This step is followed by creating a dynamic view using State Charts and a functional view using Data Flow Diagrams. These two views together constitute a description of behavior and can be made executable as has been done in the tool Statemate, though this information is not treated in this manner in the text, (Rumbaugh et. al. 1991). Event trace diagrams are used to define the interfaces between classes. They capture

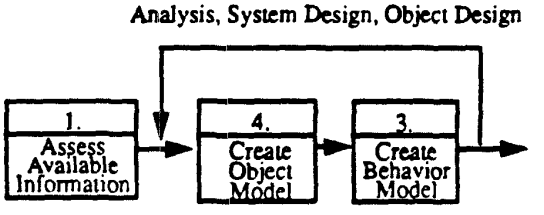

Figure 6.The Core Set of Steps in Object Modeling Technique

sequences, but not the full behavior of scenarios which could be captured with state charts and data flow diagrams, or with behavior diagrams. They lack the reusability in analysis that is available with alternative allocation of behavior to objects.

The three phases or tiers of decomposition iterate over steps 4. and 3. of Figure 6. In the first phase, Analysis, an object model emerges which is the representation of the requirements. Because it is an object model, it contains structural elements, classes and objects, which define system architecture at the lowest level without hierarchy. In the second tier of iteration, System Design, the earlier object model is modified to include subsystems, modules which are collections of classes and objects, with improved abstraction and information hiding. In the third tier of iteration the earlier object models are modified to provide detailed design of classes and objects to meet performance requirements. There is no equivalent to effectiveness measures or trade-off analy-

sis to select a best set of objects and their behaviors based on criteria at system level.

10 Conclusions and Recommendations

Software engineering needs an infusion from systems engineering of effectiveness measures, early formalized trade-off applied throughout the work, and sequential build and test. It needs consistent application of hierarchy.

Systems engineering needs to receive from software engineering an infusion of rigorous modeling and information hiding, the use of generators, and application of the classification relationship in modeling.

The mutual infusion of best practices from the other discipline strengthens both, and ameliorates existing problems of providing specifications to software engineering on large projects. The methods used here apply equally well to the development and specification of the engineering process and the products of that process. The development of information models for each step of the meta-process is critically important for communication between the engineering professionals and the vendors who are developing tools to automate their work. The information models are a critical step in moving from point tools to an integrated tools set.

11 References

(Alford 1977) Mack Alford, "A Requirements Engineering Methodology for Real Time Systems", IEEE Transactions on Software Engineering, Vol. 1, No. 1, 1977

(Alford 1992) Mack Alford, "Strengthening the Systems/Software Interface for Real Time Systems", Proceedings of the Second International Symposium of the National Council on Systems Engineering, pp. 411, Seattle, WA. July, 1992

(Bachman and Williams 1964) C.W. Bachman and S.B. Williams, "A General Purpose Programming system for Random Access Memories", Fall Joint Computer Conference, 1964.

(Blanchard and Fabrycky 1990) B.F. Blanchard, BF and W. Fabrycky, *Systems Engineering and Analysis*, Second Edition, Prentice Hall, 1990.

(Boehm 1986) B.W. Boehm "A Spiral Model of Software Development and Enhancement", ACM Sigsoft Engineering Notes, vol. 11, no. 4, August 1986, pp. 22-42.

(Dahl, Dijkstra, and Hoare 1972) O.J. Dahl, E.W. Dijkstra, and C.A.R Hoare, *Structured Programming*, Academic Press, N.Y., 1972

(Hatley and Pirbhai 1987) Derek J. Hatley and Imtiaz A. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, 1987

(Jackson and Oliver 1994) Ken Jackson and David W. Oliver, "Report of the ECBS Process and Information Model Working Group", 1994 Tutorial and Workshop on Systems Engineering of Computer-Based Systems, pp. 170, May, 1994, Stockholm, Sweden

(Gibbs 1994) W. Wyatt Gibbs, "Software's Chronic Crisis", Scientific American, pp. 86, Sept. 1994

(Martin 1994) James N. Martin, "The PMTE Paradigm, Exploring Relationships Between Systems Engineering Process and Tools", Proceedings of the Fourth International Symposium of the National Council on Systems Engineering, pp. 187, San Jose, CA. August, 1994

(MIL-STD-499 1968) *Functional Flow Diagrams*, AFSCP 375-5 MIL-STD-499, USAF, DI-S-3604/S-126-1, Form DD 1664, June 1968

(Oliver 1993) David W.Oliver, "Descriptions of Systems Engineering Methodologies and Comparison of Information Representations", Proceedings of the Third International Symposium of the National Council on Systems Engineering, pp.97, Arlington.VA, July, 1993

(Oliver 1994a) David W.Oliver, *Systems Engineering with Models and Objects*, A tutorial at the Fourth International Symposium of the National Council on Systems Engineering, pp. 315, San Jose, CA. August, 1994

(Oliver 1994b) David W.Oliver, "Systems Engineering and Object Technology", Proceedings of the Fourth International Symposium of the National Council on Systems Engineering, pp. 315, San Jose, CA. August, 1994

(Rumbaugh et. al. 1991) James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

(Stevens, Meyers, and Constantine 1974) Wayne Stevens, Glen Meyers, and Larry Constantine, "Structured Design", IBM Systems Journal, May 1974

(Yourdon 1989) Edward Yourdon, *Modern Structured Analysis*, Yourdon Press, 1989.

12 Author's Biography

Dr. David Oliver retired from GE-Corporate Research and Development after thirty-two years in science and management. He is currently providing services in training, modeling, and consulting in Systems Engineering and Software Engineering. At GE he led the development of Systems Engineering tools and processes, the Teamwork Ada CASE tool, and a X-Ray tomographic inspection system for turbine blades. He has contributed to medical and quality real-time diagnostic systems, high temperature crystal growth and materials processing. He managed the CRD Automation and Control Laboratory, the CRD Microwave Branch, the CRD Computer Science Branch and GE-CRD technical transition and liaison to all the GE businesses.