

A Comparative Analysis of Object-Oriented and Other Methods For Modeling Computer Based Systems

Stephanie M. White
Long Island University
Stephanie.White@liu.edu

Abstract

Methods for Modeling Computer Based Systems are evaluated against a common generic model. A formal Entity-Relationship (ER) model is used to compare the expressive power of methods. Expressive power is important as statements that cannot be expressed in a model will be omitted from the analysis and resulting specification. Methods are also analyzed to determine whether they are based on a partial order temporal approach rather than a linear or branching approach. (Partial order is the most powerful of the temporal approaches.) In addition, methods are judged with respect to fifteen method characteristics such as comprehension and support for reuse.

1. Modeling computer-based systems

Some fundamental needs are not being met by current methods for modeling computer based systems [28]. These include: (1) Integrated views of functionality with those of specialty engineering disciplines such as reliability, (2) Support for trade-off analysis between functionality and performance, (3) Capability to analyze scale effects, (4) Support for understanding consequences of design decisions, and constraining subsequent decisions to conform to higher level decisions, (5) Standard domain architectures (6) Support for design synthesis, and (7) Improved support for human-computer partitioning.

Large systems are the primary targets of system development concern, as they are developed and maintained for many years by teams of people who need precise documentation techniques to capture and communicate system knowledge [26]. Specific techniques were developed in the sixties to divide these systems into modular maintainable parts and analyze those [5], [8], and [17]. As systems grew in size and complexity, new methods provided added capability [29], [21], and [12]. Table 1 presents the evolution of methods. The table identifies systems which developed as an outgrowth of new technology. Each type of system was more complex in some way than previous systems, and new methods

were developed to provide a solution to the new problems. Each new method added to the capability of the then current methods.

Our goal is, through comparative analysis of methods, to identify modeling problems so that these problems can be solved. This paper identifies limitations in current methods through comparative analysis, through comparison with a generic model, and by analyzing the underlying temporal model for each method.

2. Mathematical formalisms

Embedded computer systems are complex and a single mathematical formalism is insufficient to define all system aspects. It is necessary to find a set of formalisms that provides a minimal cover of those aspects necessary for system description. Quantitative models based on mathematical formalisms are needed to support tradeoff analysis and the development of quality designs. Mathematical formalisms are required because of the size of the problem. Mathematics provides the discipline, rigor, and reasoning ability that are needed to solve complex problems. The mathematical basis of current requirements analysis models includes input to output mapping, algebra (data abstraction), procedural abstraction, function composition / decomposition, sequential machines, concurrent machines (cooperating sequential processes), extended abstract machines¹, predicate logic, and temporal logic.

Temporal logic is used for specification and formal verification of ultra-reliable systems. Primary classifications of temporal logic are based on linear, branching, and partial order. Partial order has been demonstrated to be superior to purely linear and branching models for expressing system dynamics [20].

¹ An extended abstract machine is an abstract state machine that permits condition "guards" to be associated with events. This makes the specification more succinct as the machine has knowledge about states of other machines, and every event sequence does not have to be specified.

Table 1. Evolution of methods

System Type	Added Complexity	System Development Method	Added Capability
Single processor application	---	Flow Chart	---
Operating System	Real-time control	State machine	Associated I/O with temporal order
Compiler	Complex analysis	Graph theory	Analysis
Large single processor application	Domain Complexity	Structured programming	Improved program, comprehension, testability
Large unprecedented application	Lack of domain knowledge	Stepwise refinement of data, program	Ease of change, extensibility
Multiprocessor operating system	Concurrent control	Cooperating sequential processes	Concurrency & synchronization
Large transaction-based system	Size of program & persistent data	Decomposition, data flow & structure	Division into work packages, data modeling
Distributed real-time system	Communication, reconfiguration	Cooperating extended state machines	Partial temporal order approach
Distributed real-time embedded multi-system	System layers	Decomposition of extended state machines	Stepwise refinement of temporal granularity, function and data
Evolving system	Changes corrupt original design	Object oriented analysis & design	Maintainability

The linear order temporal approach defines the dynamic behavior of a system as the set of all possible runs, where a run is a sequence of states S with intervening transitions t [2].

The branching temporal order approach specifies system behavior which caused the present state as a single run, and specifies future system behavior as a tree of runs. The points at which future system runs diverge are nodes, and the set of complete runs are represented by the various paths through the tree. Branching logic is advocated for formal reasoning by McDermott [18].

Partial order methods specify the causal relationship between events and can express concurrency, feedback, and non-determinism. A partial order on a set A is a relation \leq on A such that:

$$x \leq x \text{ for all } x \text{ in } A$$

$$\text{If } x \leq y \text{ and } y \leq z, \text{ then } x \leq z$$

$$\text{If } x \leq y \text{ and } y \leq x, \text{ then } x = y$$

The causality relation between events, "causes or is in a feedback loop with or is concurrent with," defines a partial order.

Pnueli demonstrates [20] that branching and partial-order approaches are better than the linear approach for discriminating between different versions of non-determinism, such as between the decisions: "a, then b or c", and the decision: "a then b, or a then c". In the first we make decision a, then decide whether to do b or c. In the second, a decision is made immediately to do "a,b" or

"a,c". The linear approach models both as "ab,ac". A partial order approach is better than both the linear and the branching approach in discriminating between concurrency and non-determinism. A comparison of two statements is made: (1) a and b are activated concurrently, and (2) a is activated followed by b is activated, or b is activated followed by a is activated. In (1) a is parallel to b ($a \parallel b$). In (2) we have ab or ba ($ab + ba$). Neither linear nor branching logic can discriminate between (1) and (2), modeling both as ab or ba .

3. Modeling methods

A number of methods distinguished by the inclusion of temporal as well as function and data relationships have been proposed for modeling real-time requirements. Temporal relationships, which are necessary to support the modeling of system reaction to external events, are described in most methods using a structure similar to a finite state machine. In methods that incorporate both function and state decomposition, functions, data, and states are decomposed to relate decomposition of control to decomposition of function. The Unified Modeling Language (UML) decomposes state but not function. It is difficult to use UML for systems engineering because it lacks function decomposition. One method, Jackson

System Development, decomposes events but does not decompose states.

Events are data dependent and we propose that the use of event decomposition in state-machine-based methods would simplify the relationship between levels of states, functions, and data. However, none of the examined methods decomposes both state and event.

In [25], we examine eight methods for modeling reactive systems to determine the formalisms used, and to evaluate method capability for supporting embedded system specification. In this paper we extend this analysis to include recent object oriented methods that are based on UML [15], [16], [6], [3], and [19]. Methods that were analyzed previously are:

- Distributed Computing Design System (DCDS), a function flow and dataflow method [1]. DCDS was developed by TRW for the US Army Ballistic Missile Defense Advanced Technology Center and included the Software Requirements Engineering methodology (SREM) and the Systems Requirements Engineering Methodology (SYSREM). DCDS methods were employed in Requirements Driven Design (RDD).
- Yourdon Structured Analysis for Real-Time Systems, (SA-RT), a dataflow and state-machine method [23].
- Higher Order Software (HOS), a function composition method [9].
- Jackson System Development (JSD), a data (event) structure method [14].
- Software Cost Reduction (SCR), a method developed by Parnas and others at Naval Research Laboratories, based on cooperating sequential processes and event-action causality [12], [13].
- PAISLey [30], a method developed by Zave at AT&T, based on cooperating sequential processes and functional programming.
- STATEMATE, a method developed by Harel and Pnueli, based on cooperating sequential processes and state decomposition [10], [11]. This work originated at Israel Aircraft while working with Lavi.
- Pamela, an object-oriented requirements analysis method [4].

Each of these methods is defined in greater detail in [25] and is used to model a Home Heating System benchmark problem.

4. Formal Basis for Model Comparison

We compared each method's model with a generic model. The generic model is a synthesis of many of the entities and relationships in examined methods that relate to system control. This formal Entity-Relationship (ER) model was used to compare the expressive power of methods.

Methods were also analyzed to determine whether they use a partial order temporal approach and thus are capable of specifying maximal concurrency and allow non-

determinism where a specific order is not required. All methods specify some concurrency but most constrain temporal order to be deterministic. Within processes, most methods use a linear or branching order concept.

Experimentation with method models has shown that they can all be considered as special cases of the ER model [22]. Formalizing system definition methods using ER models has several advantages:

- Method objects and relationships are precisely defined.
- Requirements knowledge captured by methods can be stored in a database for analysis and retrieval.
- Expressive power of method languages can be compared.

The Entity-Relationship model in figure 1 is used as a baseline to analyze a method's underlying model. The generic model is a simplified one, but includes environment as well as system functions, which illustrates whether the method addresses environment modeling. The generic model also decomposes state, function, event, and data. This is used to demonstrate that the evaluated methods decompose either event or state, but not both (e.g., JSD decomposes events while STATEMATE decomposes state). Decomposition of both state and event would provide more power for tracing system requirements to software requirements.

In the illustration, rectangles represent objects; circular nodes represent relationships; the nodes are numbered for identification. The relationship holds in both directions.

In the generic model, both the ENVIRONMENT and SYSTEM contain FUNCTIONS. A FUNCTION is performed in certain STATES, causes and is triggered by EVENTS, is a refinement of a more abstract FUNCTION and uses and sets DATA. STATES can have subparts, and are contained in a STATE-MACHINE-ABSTRACTION, which can be a refinement (detailed definition) of a more abstract STATE. EVENTS contain and affect STATES. EVENTS are decomposed, and the levels of event refinement relate to the levels of data refinement. EVENTS contain EVENTS and an EVENT at an abstract level is started and ended by EVENTS at a more detailed level. EVENTS are a function of DATA, which is also decomposed.

5. Evaluation of methods

Three methods, Object Oriented Analysis and Design, Software Cost Reduction (SCR), and STATEMATE are evaluated in this paper. DCDS, SA-RT, HOS, JSD, SCR, PAISLey, STATEMATE, and Pamela are described in detail in [25], where they are used to model a benchmark problem and are evaluated. A set of tables which compares the eight methods (not Object-Oriented Analysis and Design) is contained in [27].

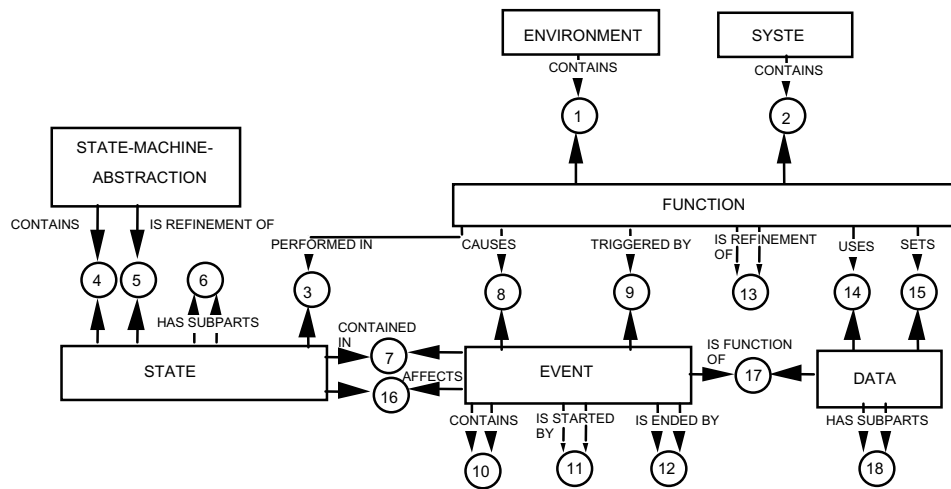


Figure 1. Generic real-time model

5.1 Object-oriented (OO) analysis and design

The OO process defined in [15] begins with definition of a Use Case diagram and a concept model. The Use Case Diagram shows external actors (users and systems) and their interaction with Use Cases. The concept model is an initial class model based on domain concepts, their attributes and associations.

Each Use Case is further detailed as required. A Use Case is a scenario that identifies the interaction between an actor (human or external system) and the system under consideration. The scenario includes normal flow of events and system actions, followed by alternate events that may take place, and the resulting system actions. Use Cases can be represented in System Sequence Diagrams (SSDs). An SSD is a trace of messages from the actor to the system. Each message represents a high level operation that the system shall perform. Input and Output parameters are included on the SSD for each system operation. Each System Operation is further defined in a Contract. The Contract pre-conditions and post-conditions specify the system state prior to Operation execution and the system state after Operation execution.

Interaction Diagrams specify a sequence of messages that objects send to each other to carry out the System Operation. Each message includes input and output parameters and their type. The object receiving the message performs the method. The method is therefore assigned to the class to which the object belongs. The concept model is refined into a class model, including attributes and methods specified in the Interaction Diagrams. Visibility that one object has of another is included in the class model.

5.1.1 Object-Oriented Model. Use Case Diagrams define context & high level system functions. Use Cases can be formalized using extended abstract machines (state transition diagrams with guards) or System Sequence Diagram traces. Class Diagrams define class attributes, methods, association, and visibility. Package Diagrams define system composition. Interaction Diagrams define collaboration among cooperating objects. Each object can also be defined as an extended abstract machine. Deployment Diagrams specify allocation of components to nodes.

We hypothesize in [25] that the mathematical basis of all requirements modeling methods are: input to output mapping, algebra (data abstraction), process abstraction, function composition/decomposition, sequential machines, concurrent machines (cooperating sequential processes), extended abstract machines, and predicate logic. If we substitute cooperating objects for cooperating sequential processes, and realize that traces are the result of unfolding state machines, our hypothesis still holds.

5.1.2 Evaluation of OO model with respect to generic model.

Use Cases can be defined at both the business/mission level and at the system level, so OO Analysis and Design can address functions in the environment informally, but not formally and therefore does not include relation (1). OO methods do not model event decomposition, and thus do not include relations (7, 10, 11, and 12). Since UML and OO methods include hierarchical state machines, these methods can express relations (3, 4, 5, 6, and 16). Since methods are functions, and methods are initiated by and can send messages, relations (8 and 9) are supported. Since a system level operation is carried out by class methods, relation (13) is supported, but only for one level of refinement. Since

methods use and set data, relations (14 and 15) are supported. Since parameters are used in the expression of events, relation (17) is supported. Since datatypes can have substructures, relation (18) is supported.

5.1.3 OO temporal approach. OO analysis and design specifies concurrency by modeling concurrent states, operations, and methods. The use of guards to specify conditions under which the system performs operations and methods, and the use of extended abstract machines support maximal concurrency. However, maximal concurrency is not a primary concern of the OO approach, as evidenced by emphasis on use cases, sequence diagrams, and message ordering in collaboration diagrams. Where ordering is sequential or branching (the method used in use cases to handle alternate events) non-deterministic order cannot be shown.

5.2 Software Cost Reduction (SCR)

The Software Cost Reduction (SCR) project was established by Naval Research Laboratory (NRL) to prove that modern software engineering practices could be used for large Navy Software projects. NRL rebuilt aircraft software using these principles.

5.2.1 SCR model. The SCR Software Requirements Specification is based on the principles of separation of concerns and information-hiding. Hardware dependent information is specified in the SRS Input and Output Data Items section. Only acronyms for input and output names are used elsewhere. Software functional requirements are described by Modes of Operation, and by Time-independent Description of Software Functions. If the definition of environmental events and required software actions change, only these sections should have to be changed. The Required Subsets section identifies subsets of services that could be useful, if isolated, in the development of similar systems. The Expected Types of Changes section identifies areas of possible future change so that the design can accommodate these.

Templates are completed for inputs and outputs. A function is defined for each output. Tables, based on a state machine formalism, define events and conditions that cause a change in output value or that activate and deactivate periodic functions.

Because functions differ greatly in different modes, modes are used to simplify the function description. A mode in the SCR methodology is a system state defined by the history of events in the system. The modes are grouped by mode class. The system can be in more than one mode class at a time. The modes within each mode class are mutually exclusive.

Text macros take the place of compound conditions and data item definitions, when the data items are not outputs or directly derived from inputs. The use of text macros

supports information hiding. If there is a change in the set of compound conditions or in the data item definition, only the dictionary has to be changed, not the entire document.

5.2.2 Evaluation of SCR model with respect to generic model. The SCR methods do not model functions in the ENVIRONMENT and cannot express relation (1) in the generic model. SCR methods do not decompose STATES and EVENTS and thus cannot express state decomposition relations (5, 6, 7, 10, 11, and 12).

5.2.3 SCR temporal approach. The SCR methods are based on a partial order concept. Since an input to output transformation is not identified, non-determinism with respect to order and maximal concurrency can be specified. The extended machine concept simplifies the definition of required temporal order.

5.3 Statemate

The Statemate model is a graphic model based on cooperating sequential processes, extended abstract state machines, predicate logic and dataflow.

5.3.1 Statemate model. The model contains templates, Statecharts and Activity Diagrams. Templates are completed for the following objects: state, condition, event, action, activity (function), signals/variables, modules, and channels (which connect modules). A Statechart is a visual extension to conventional State Transition Diagrams used in SA-RT. An Activity Diagram shows data and control flow and is similar to a dataflow diagram in SA-RT.

The Statechart is a major contribution of Statemate. A number of state transition diagrams can be shown on the same chart, displaying parallelism, selection, and decomposition. Statemate can model cooperating sequential processes or support structured analysis methods.

5.3.2 Evaluation of Statemate model with respect to generic model. Statemate does not decompose EVENTS. Statemate can express all relations in the generic model except event decomposition relations (7, 10, 11, and 12).

5.3.3 Statemate temporal approach. If Statecharts were used alone, Statemate would be based on a partial order concept. With the incorporation of user specified Activity Diagrams, a transformation from input to output is defined and Statemate is reduced to a modified branching logic. Within the branching logic, maximal concurrency and non-determinism with respect to order cannot be specified.

6. Evaluation with respect to attributes

Methods are judged with respect to fifteen characteristics, see table 2. Method evaluation leads to method improvement. Every method lacks some of the characteristics identified as desirable. As a result of evaluation and comparison, method strengths and weaknesses are revealed and each method can be augmented with capabilities found elsewhere. Limitations with OO Analysis and Design identified in this study include (1) difficulty in understanding interaction of parts in large systems; (2) hierarchical modeling of scenarios and function; (3) model tailoring for hardware versus software engineers; (4) nonfunctional requirements are not addressed; (5) OO methods do not provide data for design

tradeoff analysis, which is needed by transaction based discrete event simulation tools; (6) it is difficult to determine whether requirements are missing; (7) traceability is needed between systems and software models; (8) use cases, sequence diagrams, and message ordering in collaboration diagrams are linear or branching and non-deterministic order cannot be shown; and (9) event decomposition is not available, limiting the ability to specify behavior hierarchically.

Additional limitations with UML for integrated engineering defined by the OMG DSIG on Systems Engineering include: modeling continuous time behavior, input/output flow including data and mass energy flow, causal analysis, and system, subsystem, component and element representation [7].

Table 2. Method comparison

Method Attribute	SCR	STATEMATE	UML
1. Primary Objects (A) Function (B) Data, Condition (C) Event (D) State (E) State machine abstraction	(A) Function (B) Input, Output, Condition guard (C) Event, Action (D) Mode (E) Mode Class, Function Tables	(A) Activity (B) Input, Output, Datastore, Condition (C) Event, Action (D) State (E) Statechart	(A) Method, Operation, Use Case (B) Input & Output Parameters, Guard condition, Precondition, Post-condition, Attribute (C) Event, Action (D) State (E) Statechart diagrams
2. Formal Basis	Input to output mapping Function composition Cooperating sequential processes (CSP) Predicate logic Extended abstract machines	Input to output mapping Function decomposition CSP Predicate logic Extended abstract machines	Input to output mapping One level of function decomposition (system operation to methods) CSP, Predicate logic Extended abstract machines
3. Construction problems	Relatively easy to change as long as modes are stable or not used.	Problem choosing activities, model difficult to change.	Relatively easy to change, but only one level of functional decomposition.
4. Comprehension	Problems understanding how functions work together and the order of functions.	Problems understanding inter-relationships of activity diagrams on the same level.	Difficult to understand relationship between system level and software level specification.
5. Design Independence	Division into modes may constrain the design. This decision should be postponed until the remainder of the model is complete.	Decomposition of activity & state diagrams requires that the analyst make design decisions too early.	Division into classes that are unrelated to the environment may constrain the design. Assignment of methods to objects may constrain the design.

Method Attribute	SCR	StateMate	UML
6. Stepwise Refinement	Stepwise refinement is performed using modes and text macros. Functions may behave differently depending on the mode.	Difficult to relate decomposition of state to decomposition of function. Processes are turned on and off at each level by a state transition "controller".	Does not support hierarchical modeling of scenarios. Operations are carried out by a set of methods (There is only one-level of refinement).
7. Separation of Concerns	Document is organized according to concern/viewpoint: hardware device, hardware constraints, functional requirements (modes, functions), timing, and accuracy.	Hardware interface constraints could be isolated. Separation of concerns is not of primary importance in StateMate.	Data and related methods (functions) are organized into classes. Each class addresses a single concern. In addition, the black box behavioral view (use case) is separated from the clear box behavioral view (collaboration model).
8. Non-functional requirements	Timing is associated with function and update rates with inputs.	A timing constraint can be associated with a function.	Timing constraints may be expressed using time expressions on message or stimuli names, or by associating them with transitions and activations. The set of time functions is open-ended and includes <i>sendTime</i> , <i>elapsedTime</i>
9. Support for Design Tradeoff Analysis	No reference in literature.	Has objects called module and channel and provides primitive form of simulation.	The UML specification does not support design tradeoffs including usage trade-offs for pattern use.
10. Test Identification	Event-action causal relationships and function timing constraints provide a good definition for acceptance test.	Flows of events and actions are identified in statecharts.	Flows of events and actions are identified in behavioral models including statechart diagrams, use cases, collaboration diagrams, and activity diagrams.
11. Identification of TBDs	Omissions can easily be identified as blanks or "TBD" in tables.	Templates, which are completed for State-mate elements (states, conditions, events, activities ...) would support use of TBDs.	The UML does not assume that all of the information in a model will be expressed as diagrams; some of it may only be available as tables. Tables can support TBDs.
12. Verification	Analysis of extended state transition tables using methods in [25]. Can check tables for omissions, and for all possible output values. Negate conditions to assess the affect.	Simulation capability and consistency checking across interfaces.	Modelers may make assertions. If the assertion is violated, the model is ill formed. An execution engine is not required to verify explicitly a modeler's assertion.
13. Reusability	Input and output tables would be reusable for the same hardware devices.	If Statecharts are used to model classes, they would be reusable to the extent that the classes are reusable.	Classes are designed for reuse. Interaction diagrams are reusable if they represent a pattern. Use cases can be reused (log-in).

Method Attribute	SCR	Statemate	UML
14. System & Software Definition	Causal relationships can express system and software requirements. Without stepwise refinement, the relationship between definition levels is not easily perceived.	Can express both system and software essential models. However, without event decomposition, it is difficult to express and relate meaningful events at both levels.	OMG has a Systems Engineering Special Interest Group, which has defined requirements for a general-purpose systems modeling language, based on UML.
15. Maximal Concurrency & Non-determinism	Use of extended abstract machines support maximal concurrency. In an extended abstract machine, the state transition is caused by an event with a condition guard.	Extended abstract machines support maximal concurrency, but transformation from input to output in activity diagrams results in a modified branching logic.	Use cases, sequence diagrams, and message ordering in collaboration diagrams are linear or branching and non-deterministic order cannot be shown.

7. References

- [1] Alford, M., "SREM at the Age of Eight: The Distributed Computing Design System", *Computer*, Apr. 1985, pp. 36-46.
- [2] Barringer, H., Kuiper, R., and Pnueli, A., "Now You May Compose Temporal Logic Specifications", *Proc. of 16th ACM Symposium, on Theory of Computing*, 1984, pp. 51-63.
- [3] Booch, G., Rumbaugh, J. and Jacobson I., *The Unified Modeling Language User Guide*, Addison Wesley, MA, 1999.
- [4] Cherry, G., "The Pamela Methodology", *Proc. of Nat'l Conf. on Methodologies and Tools for Real-Time Systems*, National Inst. for Software Quality & Productivity, Wash., DC, 1987.
- [5] Dijkstra, E.W., "The Structure of 'THE' Multi-programming System", *Comm.s of the ACM*, May 1968, pp. 341 – 346.
- [6] Fowler, M. with Scott, K., *UML Distilled*, 2nd edition, Addison Wesley, MA, 2002.
- [7] Friedenthal, S., "Extending UML from Software to Systems", *Powerpoint Presentation to INCOSE Chapter, Crossroads of America*, Sept. 2002.
- [8] Gries, D., *Compiler Construction for Digital Computers*, John Wiley & Sons, NY, 1971.
- [9] Hamilton, M. and Zeldin, S., "The Functional Life Cycle Model and Its Automation: USE.IT", *Journal of Systems and Software*, Vol. 3, No. 1, Mar. 1983, pp. 25-62.
- [10] Harel, D. and Pnueli, A., "On the Formal Semantics of Statecharts", *2nd Proceedings IEEE Symposium on Logic in Computer Science*, Ithaca, NY, 1987, pp. 54-64.
- [11] Harel, D., "On Visual Formalism", *Communications of the ACM*, Vol. 31, No. 5, May 1988.
- [12] Heninger, K., Kallander, J., Parnas, D.L., and Shore, J., *Software Requirements for the A-7E Aircraft*, Naval Research Lab, Washington, DC, Memo Rep. 3876, Nov. 1978.
- [13] Heninger, K., "Specifying Software Requirements for Complex Systems: New Techniques and their Application," *IEEE Trans. on Software Engineering*, Jan. 1980, pp. 2-13.
- [14] Jackson, M., *System Development*, Prentice Hall, NJ, 1983.
- [15] Larman, C., *Applying UML and Patterns*, 2nd edition, Prentice Hall, NJ, 2002.
- [16] Liskov, B. with Guttag, J., *Program Development in Java*, Addison Wesley, MA., 2001.
- [17] Madnick, S.E., and Donovan, J.J., *Operating Systems*, McGraw Hill, NY 1974.
- [18] McDermott, D., "Reasoning About Plans", in J.R. Hobbs and R.C. Moore, eds., *Formal Theories of the Commonsense World*, Ablex Publishing Corp., Norwood NJ, 1985.
- [19] OMG, *UML Language Specification V1.5*, Mar. 2003.
- [20] Pnueli, A., "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", in *Current Trends in Concurrency, Overviews and Tutorials*, ed. by J.W. deBakker et al., Springer Verlag, Lecture notes in Computer Science, Vol. 224, 1986, 510-584.
- [21] Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Trans. on Software Eng.*, Vol. SE-3, No.1, Jan. 1977, pp. 16-34.
- [22] Teichroew, D. et al., "Application of Entity-Relationship Approach to Information Processing Systems Modeling", in P. Chen, ed. *Entity-Relationship Approach to Systems Analysis and Design*, North Holland, 1980, pp. 15-34.
- [23] Ward, P.T. and Mellor, S.J., *Structured Development for Real-Time Systems*, 3 Vols., Yourdon Inc, NY, 1985.
- [24] White, S. and Meyers, S., *Software Requirements Methodology and Tool Study for A-6E Technology Transfer*, Grumman ASD Technical Report SRSR-A6-83-001, Jul. 1983.
- [25] White, S., *A Pragmatic Formal Method for Computer System Definition*, PhD Dissertation, Polytechnic U., 1987.
- [26] White, S. et al., "Systems Engineering of Computer-Based Systems", *Computer*, Vol. 26, No. 11, Nov. 1993, pp. 54 -65.
- [27] White, S., "A Comparative Analysis of Requirements Methods", *Proc. of IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press, Apr. 1994.
- [28] White, S., Lawson, H., and Melhart, B., "Engineering Computer Based Systems, Meeting the Challenge", *IEEE Computer*, Vol. 34 No. 11, November 2001, pp. 39-43.
- [29] Wirth, N., "Program Development by Stepwise Refinement," *Comm.s of the ACM*, April 1971, pp. 221 – 227.
- [30] Zave, P. and Schell, W., "Salient Features of an Executable Specification Language and Its Environment", *IEEE Trans. on Software Engineering*, Vol. SE-12, 1986, pp. 312-325.