Technical Report

CMU/SEI-89-TR-036
ESD-89-TR-047

Comparative Evaluations of Four Specification Methods for Real-Time Systems

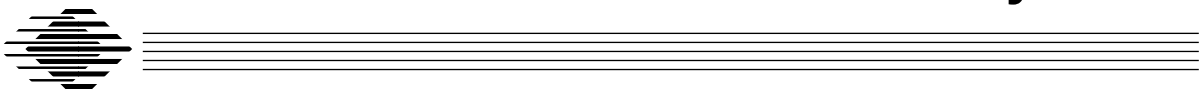
David P. Wood
William G. Wood

December 1989

Technical Report

CMU/SEI-89-TR-036
ESD-89-TR-047
December 1989

Comparative Evaluations of Four Specification Methods for Real-Time Systems



David P. Wood

William G. Wood

Specification and Design Methods
and Tools Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the SEI Joint Program Office HQ ESC/AXS

5 Eglin Street

Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF, SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Asset Source for Software Engineering Technology (ASSET) / 1350 Earl L. Core Road ; P.O. Box 3305 / Morgantown, West Virginia 26505 / Phone: (304) 284-9000 / Fax: (304) 284-9001 / e-mail: sei@asset.com / WWW: http://www.asset.com/sei.html

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service / U.S. Department of Commerce / Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides acess to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential con tractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: 1-800-225-3842 or 703-767-8222.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Preface

Which methods and tools are the best? is a question frequently heard at meetings, conferences, and symposia for software engineering, software development methods, Computer-Aided Software Engineering (CASE), and other activities related to software technology. The question begs a simple, straightforward answer, but is often given subjective, erroneous, or misleading treatment.

How may we provide an answer to this critical question? What knowledge must we have at our disposal? What analyses must we perform? To begin, consider the following issues:

- Identifying the best toolset might require knowledge of the best methodology.
- Identifying the best methodology might require knowledge of the best life-cycle strategy.
- Identifying the best life-cycle strategy might require knowledge of the best methods for each phase.
- Identifying the best methods might require knowledge of the best paradigms for specification, design, verification, and so on.
- Identifying the best paradigms might be highly dependent on the application domain.

There is little agreement to be found in any of the above issues. In addition, the relationships between these and other issues can be circular: knowledge of A requires knowledge of B which requires knowledge of A. Further, definitive answers can be derived only from comprehensive research. Declaring X the best toolset and Y the best method is meaningless if one has evaluated only a few methods and tools from an inventory of many hundreds, an inventory whose boundaries are vague, poorly defined, and continually evolving.

Which methods and tools are the best? Answering this question definitively requires meaningful metrics and controlled experimentation, neither of which appears within the grasp of existing resources or technology. This report does not intend to provide definitive answers to the question. Rather, our focus is necessarily humble: we examine a small subset of methods related to one part of the development life-cycle. Through this report, we hope to assist each organization in asking the right questions and in developing a series of workable answers.

Comparative Evaluations of Four Specification Methods for Real-Time Systems

Abstract: A number of methods have been proposed in the last decade for the specification of system and software requirements for time-critical systems. The emerging CASE technology is based heavily on a subset of these methods; yet little *objective* attention has been paid to the methods themselves. This report describes our objective evaluation of four methods (identified as *ESML, Harel, Hatley-Pirbhai*, and *Ward-Mellor*), from identification through detailed assessment. We have avoided the use of small sample problems as the sole basis of our evaluation. We depart from this approach by involving software developers from various application domains, including extended interviews of those who have applied the methods to large-scale projects. The resulting recommendations and conclusions focus on method selection criteria, and on the large-grained impact of using these methods on a given project.

The primary audience of this report is the software development practitioner involved in the method selection or adoption process. The paper attempts to provide proper context to assist the practitioner in making appropriate method adoption decisions. Secondarily, the results of the paper also should be of interest to tool vendors, method developers, and program managers.

1. Executive Overview

The Software Engineering Institute (SEI) Methods and Tools Project completed an investigation of software development methods that has resulted in a comparative evaluation of several methods for the specification of system and software requirements of time-critical systems. The methods investigated were *ESML, Harel, Hatley-Pirbhai,* and *Ward-Mellor*. These methods were selected because they reflect the current "state of the practice," in that they have a track-record in industry that can be examined, they have a solid technical basis, and they form the foundation of a large proportion of the emerging front-end CASE technology. We expect evaluation of these methods to provide a more significant point of leverage than would evaluations of less mature, more experimental technologies. In other words, software houses and program offices can use these evaluations today.

We followed an approach that began with a literature search and a survey of SEI affiliates in industry, academia, and the military. To ensure technical accuracy, we invited the participation of the developers of the selected methods in providing comments, suggestions, and critical reviews. To establish a basic understanding of the methods, we solved a number of sample problems using each method. To ensure the applicability of our results to large-scale, complex, real-world applications, we included the experiences of users of the methods as a fundamental input into the evaluation process. As a result, we believe that our recommendations reflect the actual operating conditions under which methods such as these are commonly used.

Using the information from actual users, the developers of the methods, and our own internal investigations, we applied the data to a predetermined series of evaluative questions established in the SEI technical report *A Guide to the Assessment of Software Development Methods* [Wood88]. The answers to these questions were then distilled into a set of observations and recommendations targeted primarily to practitioners involved in method selection and adoption, and secondarily to tool vendors, method developers, and program offices. The observations and recommendations are detailed in Chapter 6 of this report and summarized below:

- These methods have been used successfully, but there are prerequisites for success. All of the failures that we observed, and the bulk of the difficulties, were attributable to problems in management, training, and tools. Relatively few problems were attributable to the methods themselves.

- There are few significant discriminating factors that can be used to denote any method as "the best"; however, important discriminating factors are presented to assist selection on a project- or program-specific basis. Key discriminators include compatibility with process and life-cycle goals, tool capabilities, availability of training and method expertise, and the personal experiences of project members and technical leaders.

- These methods do not solve the essential difficulties of software development [Brooks87], but, used effectively, they permit developers to concentrate their attentions on those essential aspects. It is probably unrealistic to expect any method to solve the essential difficulties of software development.

- Improving current practice will provide better leverage than switching from one of these methods to another. Probably the most significant factor in the repeatable success of a software contractor is an evolving software process [Humphrey87], including evolving methods, tools, and environments. By contrast, continually jumping from one method to another is more likely to result in project disruption than in demonstrable improvements.

- Investment of resources in method and tool selection activities, up-front training, ongoing consultation, and development of problem workarounds is necessary and should be anticipated. Contractors and program offices should plan for such investment early in the acquisition process.

- Beware of method and tool hype. Practitioners should be cautious of claims of "best" methods, methodologies, tools, and environments, remembering that significant evidence of causal relationships, not to mention meaningful metrics, is essentially nonexistent. At best, educated guesses can be provided as guidance. At worst, the hyperbole can be seriously misleading. The best defense against being overwhelmed by an overload of information is to take an evolutionary rather than revolutionary approach to method and tool adoption.

- Tool vendors should concentrate on improving support for the subject methods, in preference to adding trendy features. They should also focus on flexibility, interoperability, and other development environment issues.

- Method developers should provide tool vendors with detailed requirements for appropriate support of their methods. They should also focus on the issues of transition to other software development notations, and on effective maintenance of their own artifacts.

- Program offices should require contractors to exhibit an evolving software process, including methods, tools, and environments. Where multiple contractors are involved in the development of the system, care should be taken to ensure method compatibility among contractors.

Chapter 6 presents a set of important criteria for discriminating among these methods that must be applied on a project- or program-specific basis. Also presented is an additional set of discriminating factors that permit distinctions among the methods on a more general level. In particular, the quality of available texts, the overall simplicity of notations, and the support of system design issues are discussed. Where program-specific factors are deemed to be equal, we lean toward the *Hatley-Pirbhai* and *Harel* methods (among those that were evaluated) based on some of the latter distinctions. Our reasoning is described in detail within this report.

With these recommendations, program managers, project leaders, and engineers can express rational preferences among these methods. These preferences then can be weighed against preferences in tools and environments in making appropriate trade-off decisions.

2. Introduction

2.1. Target Audience and Alternative Sources

The primary audience of this report is the software development practitioner. The report attempts to provide proper context to assist the practitioner in making appropriate method adoption decisions for real-time system specification methods. The report is also targeted to method developers and tool vendors interested in the perspective of large-scale users of the methods and to program offices that are attempting to upgrade the technology base of their contractors.

We evaluate and discuss only a few methods in this report. For the reader interested in more general information on a wide variety of methods, we have found the *Software Methodology Catalog* ([CECOM89]) to be an excellent reference.

This report focuses mostly on the large-grained impact of method adoption rather than small-grained details. For those interested in a detailed technical dissection of the many interesting semantic issues of these and other methods, we recommend *A Pragmatic Formal Method for Computer System Definition* by Stephanie White ([White87]) as a good source of information.

Based on the nature of these methods, we felt that providing trivial examples of graphical notations would be inappropriate and distracting. The notations of these methods should be viewed in the larger context of an entire model or, ideally, of a large-scale application. Therefore, we have not included graphic examples but instead recommend that the reader refer to method-specific literature for illustration, such as [Harel87], [Harel88b], [Ward85], [Bruyn88], and [Hatley87].

2.2. Background

It is generally recognized that new technologies are accepted by industry at a sluggish pace [Jenson88]. Slow technology transition can be attributed to any number of factors: the technology may be inadequate; it may be substantially different from existing technology; it may be lacking in documentation and expertise; there may be many competing technologies of apparently equal capability; there may be a lack of perception for the need for such technology; or there may be many political or economic hurdles to be overcome. In the case of techniques for the specification of system and software requirements, any or all of these factors are significant.

Discussion of specification techniques is generally mired in confusion. The topic is polarizing, resulting in many heated debates over such questions as:

- What is the difference between specification and design?

- What is the difference between system and software requirements?
- What are the significant differences between specification of a real-time system and one that is not real-time?
- Should a specification be rigidly formal or not?
- Should a specification be object oriented, functionally oriented, or data-oriented?

The difficulty in answering these and other polarizing questions is compounded by the relatively muddled state of the technology, exemplified by:

- Sorting through confusing and conflicting terminology
- Comprehending the difference between methods, CASE, and environments
- Differentiating between CASE/method hype and reality
- Understanding the role of software process and management issues

More than anything else, these questions and topics illustrate the relative immaturity of the technology. Unfortunately, software engineering practitioners generally do not have the resources or inclination to ponder such matters. What they wish to know is *Which methods and tools are the best?* Although definitive proof does not exist, we must answer as best as we can based on available literature and experience.

2.3. Purpose and Scope

The purpose of this report is to help the practitioner wade through at least some of the relevant questions in order to reach a rational decision on method selection. The purpose is *not* to evaluate automated tools, life-cycle models, or documentation standards, nor is it to attempt to evaluate methods across the life-cycle. Each of these represent important topics, but are outside the scope of this effort.

Rather than attempt a broad-based compendium of (necessarily high-level) method descriptions, we have chosen to focus on a very limited subset of methods for examination. In particular, our focus for this report is on methods for system and software specification for real-time systems. Our process for the identification of methods that meet the above criteria is described in greater detail in Chapter 3 of this report. Brief discussions of several related topics are provided for completeness in Appendix A.

2.4. Approach and Report Structure

The goal of these evaluations is to provide specific, objective, and substantiated recommendations pertaining to the selection of specification techniques. Our approach has been based loosely on two reports published previously, [Firth87] and [Wood88], which detail criteria for method classification, assessment, and selection.

The structure of this report parallels the steps of our approach:

1. Identification - A literature search and a survey of SEI affiliates were used to produce a comprehensive list of methods and to provide a basis for the selection of methods for detailed evaluation. This process is described in Chapter 3 of this report.

2. Classification - Each of the selected methods was briefly analyzed and classified in accordance with the taxonomy of [Firth87]. This process is described in detail in Chapter 4 of this report.

3. Assessment - Each of the selected methods was analyzed in greater detail in accordance with the criteria of [Wood88]. Selected sample problems were used for an initial assessment, while interviews of actual users of the methods and examination of some of their resulting specifications provided substantiation for the final assessment. This process is described in Chapter 5 of this report.

4. Recommendations - Chapter 6 provides observations and recommendations drawn from the results of the above procedure.

5. Conclusions - Chapter 7 discusses the overall conclusions arising from this evaluation task.

In addition, four appendices are provided for reference to the reader:

Appendix A (Special Topics) presents a forum for the discussion of important topics that are relevant to this report but are outside of its scope. With this section, we have attempted to anticipate closely related issues that likely would be raised in the minds of readers, and to discuss them in brief.

Appendix B (Affiliates Survey Results) describes the results of the survey of SEI affiliates conducted for this project. A concise summary of the survey is provided in Section 3.2.

Appendix C (Evaluation Questions and Answers) provides the evaluation questions and answers that were used as input to our final recommendations.

Appendix D (Comments From Methodologists) provides brief critiques of this report from the principle developers of the methods. In an attempt to be as fair as possible, representatives of each method were given the opportunity to comment on the conclusions of the report.

Appendix E (Glossary) defines terms used throughout this report. Several particularly important or controversial terms are also discussed in Section 2.5.

2.5. Key Terminology

Throughout the field of software engineering, as in other immature fields, inconsistent terminology plagues the open exchange of ideas. In some cases, multiple terms have been introduced to describe a single concept; in other cases, very different concepts are denoted by the same term; in yet other cases, new terminology and buzzwords seem to be introduced for no valid reason. The purpose of this section is to define key terms (presented in bold) that we will use throughout this report. These terms are also defined in a glossary in Appendix E.

The key terms of this report fall into three areas: methods and tools, specification processes and products, and real-time systems.

Methods and Tools

We use the term **method** to refer to an approach to solving a particular problem. A true method, in our view, is one that provides both a set of notations and a set of techniques. The **notation** is some combination of graphical, textual, or tabular languages used to describe the solution to the problem at hand. Notations can range in levels of formality from free-form text to algebraically correct statements. The associated set of **techniques** permits the user of the method to derive and examine a solution based on the notation. The techniques can be provided at varying levels of specificity ranging from loose guidelines to detailed procedures or even algorithms. The possible range of derivation techniques and examination techniques for a method can be closely related to the formality of the notations. For example, thorough dynamic analysis requires a notation with well-defined, formal, behavioral syntax and semantics. We will use the term method rather loosely, to refer to any approach providing some combination of notations and techniques.

The term **methodology** is often used interchangeably with **method**, although recent literature tends to scorn such use as misleading and incorrect. There are two correct definitions [Webster85] of methodology: (1) a set of methods integrated to achieve an overall goal; and (2) the study of methods. In general, we avoid the term. Where we use it, we use the first definition, as should be obvious from context.

The term **tool**, in the context of software development, usually refers to a product for the automation of some method, part of a method, or set of methods. The term **CASE** (Computer Aided Software Engineering) in its most general use, refers to any automated software development tool (e.g., compilers, text editors, or debuggers). In common practice the term often takes on a more constrained meaning, referring to workstation-based tools that support graphics-based specification and design methods.

Whereas a tool can be thought of as supporting a single method, a **toolset** can be con-

sidered to support a methodology as defined previously. A toolset usually consists of some group of tools exhibiting some level of integration. A software development environment implies a somewhat more comprehensive toolset, one that might support areas such as project management or configuration management in addition to technical development activities.

Please refer to Appendix A for a continued discussion of automated support of methods.

Specification Processes and Products

The terminology of this area seems to be the cause of considerable confusion. As such, we advise that you consider our definitions carefully before proceeding to the heart of the report.

The **requirement** is a description of what the customer and end-user view as their needs for the system. Often mislabeled as **functional requirements**, the requirements description is frequently an eclectic mix of functional requirements (such as a scheduling mechanism) and other types of requirements (such as dynamic behavior or ease of installation), together with customer-imposed design constraints.

Specification, requirements specification, requirements analysis, and requirements definition all refer to essentially the same part of the software development life-cycle: the process of examining user requirements and their subsequent capture into a more formal representation. These terms are typically used interchangeably. Often, specification is used to refer to the end product of the process, whereas analysis is used to refer to the process itself. [1] Note, however, that specification can refer to the process itself, as well as the end product. In keeping with previous SEI reports, we use the term specification to refer to both the process of analysis and the resulting product.

In many circles, particularly those involved in the real-time, large-scale, embedded systems domains, there is a clear distinction made between **system requirements specification** and **software requirements specification**. A **system** may include software, hardware, human, or other components. Further, a given system may be a component of a larger system. System requirements specification, then, refers to the analysis of the requirements of an entire system that is to be developed. Software requirements specification refers to the analysis of the software component(s) of the system. The notations and techniques used in both types of specification may be the same; in some scenarios, the software specification is an elaboration and refinement of part of the system specification.

In proceeding from system requirements to software requirements, a **system design** is often needed. The system design (sometimes called the **system architecture**) defines the partitioning of the system into various hardware, software, and other components, the allocation of requirements to each component, and the interfaces among components. This allocation is based on any number of trade-off factors. Please note that **system design** is not to be confused with **software design**.

To clarify, this report uses the following terms as defined:

- **System requirements specification** - the process and product of the analysis of system requirements.

- **System design** - the process and product of defining the division of the system into hardware, software, and other components, the allocation of requirements into each of these components, and the detailed description of physical interfaces between the components.

---

[1] There is a correspondence between this view and our defined components of a **method**: analysis corresponds to the exercise of a set of **techniques**, while the specification corresponds to an ordered use of the **notation**.

- **Software requirements specification** - the process and product of the analysis of the software requirements.

- **Specification** - the combination of system requirements specification, system design, and software requirements specification.

Please note that we do not endorse a particular life-cycle model, and the definitions used above should not be construed as such. However, system requirements specification, system design, software requirements specification, and software design exist in one form or another in virtually all life-cycle models. For example, most modern models provide for extensive iteration and feedback between phases, resulting in somewhat fuzzy phase boundaries. A method that is to be useful to a wide audience should provide sufficient flexibility to be adaptable to multiple models.

Real-Time Systems

We have also stated that our intent here is to focus on specification methods appropriate to **real-time** systems. Real-time systems deal with the processing of data by a computer in connection with another process outside the computer according to time requirements imposed by the outside process [IEEE83]. "Real-time systems typically sense and control external devices, respond to external events, and share processing time between multiple tasks. Processing demands are both cyclic and event driven in nature [Fairley85]." Therefore, a method applicable to real-time systems specification is one that, as a minimum, is capable of capturing both functional processing requirements and event-based behavioral requirements imposed upon the functional requirements.

3. Method Identification

Based on our general knowledge of trends and developments in specification methods technology, our initial intention was to examine, as a minimum, the techniques of *Ward-Mellor* [Ward85], [Ward86], *Hatley-Pirbhai* [Hatley87], and *Harel* [Harel88b], [Harel89]. Our perception was that these techniques were relatively well understood, and gaining in acceptance and use. This latter was a critical factor for us, as it was our intention to examine actual field use of these techniques as much as practical.

To be assured that our selection was a rational one, we performed a literature search and a survey of SEI affiliates to give us a rough idea of the present state of the practice.

3.1. Literature Search

A literature search was conducted with the purpose of producing a comprehensive (though not exhaustive) list of proposed specification and design methods, to gain a nominal understanding of each method, and to gather pointers to further, detailed literature for those methods selected for evaluation.

We identified slightly over 100 methods, and categorized a subset as semi-formal specification methods. Among the subset, the most heavily debated methods were:

- Structured Analysis and variants (including *Ward-Mellor*, *Hatley-Pirbhai* and *Harel*)
- Jackson System Development (JSD) ([Jackson83])
- System Requirements Engineering Methodology (SREM) ([Alford77])

3.2. Affiliates Survey

In September 1988, a questionnaire was distributed to all affiliate organizations of the SEI and to attendees at the 1988 SEI Affiliates Symposium. The primary purpose of the questionnaire was to identify the methods of most interest to our affiliates. A total of 447 questionnaires were distributed.

The questionnaire was brief, consisting of only five questions intended to determine the breadth of real-time application domains faced by our affiliates, the requirements specification methods used for those applications, the design methods used, and the programming languages used. A final question requested aid in the identification of suitable application domain sample problems to make method evaluations more meaningful.

The responses to the questionnaire are provided in Appendix B of this report. Several conclusions may be drawn from the results of the survey:

1. The respondents represented a broad variety of real-time application domains.

2. Taken together, the methods that we chose to examine (*Harel*, *Ward-Mellor*, and *Hatley-Pirbhai*) are the dominant specification techniques of those who use methods; however, an object-oriented analysis method seems to be desired.

3. Structured Design and object-oriented design strongly dominate the area of design methods. No other methods are close to these two.

4. Ada is the dominant programming language for the questionnaire respondents, although C and FORTRAN enjoy substantial followings as well.

5. Many affiliates expressed willingness to participate further in this study to assist us in examining methods use in the field.

3.3. Selection of Methods for Classification and Assessment

As discussed previously, our initial intent was to evaluate the *Ward-Mellor*, *Hatley-Pirbhai*, and *Harel* methods. The purpose of the affiliates survey and literature search was to confirm that our intentions were rational and would be potentially useful to the software engineering community. The results of the literature search support our initial selection, although a number of other methods (e.g., JSD and SREM) might be useful for future evaluation. The responses from our survey of affiliates also tend to support our initial selection. In addition, the results seem to indicate that future investigation of the emerging object-oriented analysis and transformation techniques will be a worthwhile endeavor.

Having justified our initial selection, we elected to look at two other methods as well, due to their obvious similarities to the other three methods: Extended Systems Modeling Language (*ESML*) and Systems Engineering Methodology (*SEM*) (see [Bruyn88] and [Wallace87], respectively).

While *ESML* is not strictly a method by our definition (it includes only a set of notations), its developers have selected attributes of the *Hatley-Pirbhai* and *Ward-Mellor* methods in deriving the *ESML* notation. In addition, the derivation techniques applicable to the other methods may be applied to *ESML* to make it a complete method.

SEM is a method derived from a combination of the Structured Analysis and Design Technique (SADT) and the Software Cost Reduction (SCR) methods, together with notations for behavioral modeling. We felt it worthwhile to include *SEM* in our classifications (Chapter 4) because of its similarity to the other techniques and because of its greater level of formalism in some aspects of notation. However, we have not included it in the evaluations (Chapter 5) due to a lack of data from actual users and applications of the method.

4. Method Classifications

Prior to a full-scale assessment, the methods identified in the previous section were classified according to a non-evaluative taxonomy based on that of [Firth87]. The [Firth87] matrix categorized methods into three *development phases* (specification, design, and implementation) and three *forms of representation* (functional, structural, and behavioral). Although this scheme is useful for general classifications, it provides only limited information when a more specific classification is desired. The classification scheme used in this report represents both a reduction and an expansion of the original matrix that facilitates the side-by-side comparison of multiple methods.

The classifications comprise four charts. The first chart provides a high-level outline of the characteristics of each of the five methods. An evaluator can determine quickly where the similarities lie (e.g., development phase coverage, functional and behavioral view philosophy), as well as the differences (e.g., notational forms and structural views). This allows the evaluator to focus energy on assessing those areas that are most likely to be discriminating factors in the selection of a method. In that light, we have further classified the representational notations of the five methods to an additional level of detail: Chart 2 classifies functional representations, Chart 3 classifies structural representations, and Chart 4 classifies behavioral representations.

It should be noted that classifications such as these are intended only to categorize methods as an initial filter in the selection process. Such classifications tell us very little about the *quality* of a particular method, its *suitability* to a given application, or the level or quality of *automation* in terms of tool support. Such matters are certainly important in the final selection of methods, but are beyond the scope of classifications such as these.

The following sections present each of the four tables with brief supporting discussion.

4.1. Top-Level Classification

The top-level classification (Chart 1) makes clear the many similarities of the five methods. Each method addresses primarily the same development phase, and has an integrated functional/behavioral modeling approach incorporating data flow and finite state concepts. The chart also indicates that only *Harel*, *Hatley-Pirbhai*, and *Ward-Mellor* provide some notion of structural representations. *SEM* mentions the need for a "physical perspective," but does not elaborate further.

Other potentially important differences include the level of dynamic analysis supported by the methods (particularly emphasized in the case of *Harel*); the extent of derivation techniques provided by the methods (*Harel* provides very few guidelines, while ESML provides only a language).

4.2. Forms of Representation - Functional

Chart 2 elaborates on the representation of functional requirements. Note that all of the methods provide a specific functional view although each method uses a different name for essentially the same (data-flow) concepts. Activity Charts, Data Flow Diagrams, and Actigrams are essentially identical. This diversity in terminology is typical and can lead to confusion. In these cases, the reader should refer to the top-level chart for common terminology.

One difference of note is the format of primitive functional specifications. Only one of the methods, *SEM*, requires the use of a specific formal notation. This notation is based on the NRL Software Cost Reduction (SCR) notation [Heninger80]. The other methods allow the use of a formal notation, but more typically incorporate some combination of structured English, decision tables, or other semi-formal specification.

Figure 4-1: Chart 1: Top-Level Classification

4.3. Forms of Representation - Structural

Chart 3 elaborates on the representations of the structural requirements. In this category, we include both system architecture and data architecture modeling. This convention is in keeping with previous SEI reports, although for the remainder of this report, we will treat the issues separately. There are two important differences that stand out in this chart. First, *Harel* and *Hatley-Pirbhai* provide the necessary notations for capturing system architecture while the other methods do not provide a distinct rotation. As discussed elsewhere in this report, *Ward-Mellor* implementation models may be used in a similar fashion in many circumstances. Second, *Ward-Mellor* and *SEM* specifically include entity-relationship diagram notations in their methods to support complex data modeling. *Hatley-Pirbhai* suggests the same approach, but does not elaborate. All of the methods provide a data repository mechanism.

Figure 4-2: Chart 2: Functional Representations

4.4. Forms of Representation - Behavioral

Chart 4 elaborates on the representations of the behavioral requirements. As with the functional view, each method uses a different name for essentially the same purpose: the modeling of system behavior. In contrast, however, the mechanisms used for the behavioral modeling are dramatically different. In particular, although each method uses some form of finite state machine representation, the methods differ in notation and emphasis. *Harel* uses a unique hierarchical diagramming technique called *statecharts*; *Ward-Mellor* emphasizes conventional, state-transition diagrams but also supports some tabular formats; *Hatley-Pirbhai* emphasizes multiple tabular formats but also supports conventional state-transition diagrams and combined formats; and *ESML* and *SEM* use some mixture of diagrams and tables.

Figure 4-3: Chart 3: Structural Representations

Figure 4-4: Chart 4: Behavioral Representations

5. Method Assessment

The following sections describe the detailed assessments of four of the five methods classified in the preceding section. The fifth, *SEM*, was included in the classifications due to its philosophical similarity to the other methods, and because we feel that its use of a formal notation for functional specification potentially is an important contribution. However, because of a lack of available information on actual applications of the method on a large scale, *SEM* was not assessed and is not included in this section. *SEM* is sufficiently divergent in heritage from the other methods that we did not feel confident in extending the conclusions from the other methods to *SEM*.

5.1. Assessment Approach

Several comparative or evaluative efforts exist in available literature ([Bergland81], [Blank83], [Davis88], [DOD82], [Floyd86], [Griffiths78], [Kelley87], [Lefkovitz82], [Mannino87], [Peters77], [STARTS87], [Ward89a], [White87]); however, few of them are supported adequately by data gathered from the field on real-world applications. For this reason, we have taken an assessment approach that includes the examination of actual field experience in this report. The evaluation process is outlined below.

1. Gain basic internal expertise in the syntax, semantics, and application of the methods. This process is discussed briefly in Section 5.2.

2. Interview developers of actual applications that use the methods. Results from this effort are summarized in Section 5.3.

3. Apply knowledge gained from the preceding steps to detailed evaluation questions from [Wood88]. Answers to these questions are provided in Appendix C.

4. Summarize evaluation results into broad categories to provide flexible method selection data. This summary is provided in Section 5.4.

The results of this assessment process form the basis for the recommendations and conclusions provided in Chapters 6 and 7.

5.2. Summary of Sample Problems

A set of sample problems were solved using each method, with a goal of establishing a nominal syntactic and semantic expertise in each method. It was hoped that the sample problems would provide significant input to the evaluation process.

The sample problems were small enough in size to be solvable given our resource constraints, yet complex enough to exercise some of the critical components of each method. The problems solved include a sonobuoy system (borrowed from [SPS88]), a "generic

avionics" system (borrowed from [IBM88]), and an elevator system. Each problem emphasized a different processing aspect: the sonobuoy problem allowed modeling of system architecture properties, the avionics problem allowed modeling of typical data transformation properties, and the elevator problem allowed modeling of state behavior and concurrent properties.

A subset of sample problem solutions was reviewed by the developers of the methods to ensure the correctness of our solutions. Feedback received from the method developers has been incorporated in our evaluations.

For the most part, the sample problem exercises did not result in large-grained, discriminating data for evaluation. During the process of solving the sample problems, it became clear that the types of issues highlighted by the exercises were small-grained in nature. We are convinced that the identification of large-grained issues requires more than the relatively small effort (approximately one person-month per exercise) involved in the sample problems. In other words, these methods are similar enough that making a selection based on small sample problems is almost arbitrary, and largely a matter of personal preference. Although the small-grained differences (such as semantic conventions) are interesting and important, they are far outweighed by the large-grained issues discussed elsewhere in this report.

Initially, we anticipated that this report would focus on the comparison of the many small-grained differences among the methods, but for the reasons stated above, we have elected to not discuss small-grained issues in great detail. The remainder of the report, and particularly the resulting observations and recommendations, focuses almost entirely on large-grained issues.

5.3. Summary of Project Interviews

While solving small sample problems is useful for determining some of the technical differences among the methods and for exercising semantics of the methods, it is necessary to investigate larger scale applications to understand the significant impact of using these methods. Lacking the resources to tackle a large-scale problem in-house, we set out to identify users of the methods in industry to learn of their experiences and examine their resulting specifications. We interviewed developers from fourteen medium- to large-scale projects ranging from twelve to 1200 person-months in effort, and have had informal communications with developers on a number of other projects that use these methods. These projects involved five distinct application domains: avionics, communications, diagnostics, instrumentation, and navigation/control. In addition, we are engaged in an ongoing consultation effort with an affiliate using one of the methods in the specification of a real-time safety-critical process control and power plant protection system.

We interviewed multiple projects that have used or are using *Ward-Mellor*, *Hatley-Pirbhai*, or *Harel*, but we were unable to make arrangements in a timely manner with projects using

ESML. Because of its heritage and inherent similarity to the notations and techniques of the other methods, we feel that our conclusions can be applied reasonably to *ESML* notations. This is especially true in that we are focusing on large-grained issues.

Together, these interviews and consultations have pointed out many high-level similarities among the methods, as well as some important differences. Most importantly, a number of common themes were reported by most interviewees. These common themes point out the positive and negative aspects of adopting any of the methods that tend to overwhelm minor semantic issues and personal preferences. The results of our interviews are summarized below:

Impressions regarding the application of the methods varied widely, from extremely positive to strongly negative. However, virtually all interviewees felt the methods themselves were very useful. In nearly every case, negative impressions can be attributed to problems with inadequate or ineffective training, management, and tool support.

The most common negatives cited by the interviewees follow, in no particular order.

- Developers had difficulty determining "when to stop" in terms of levels of detail of specification; in general, this problem diminished over time as the team became more experienced at use of the method.

- There is a perceived difficulty in proceeding from the specification to the software design. This was generally attributed to a lack of guidelines from the methods.

- The methods were often cited as being ineffective as mediums of communication between technical and non-technical personnel. Although this was not considered a significant problem, it was considered a major disappointment. In a number of cases, prototyping was used effectively to overcome this problem. The methods were seen as compatible and supportive of this approach.

- CASE tool support was universally cited as a problem. Some developers found their tools "barely adequate," while others found them entirely useless or even disruptive. Many tool vendors are considered unresponsive to making needed changes. Vendor hype set up some projects with false expectations, only to find the tools lacking in performance, robustness, and sufficient implementation of the methods they claim to support.

- Some projects found that maintenance of the specifications was cumbersome. It was felt that the methods were most useful in deriving specifications from the raw requirements, but not very useful, or even detrimental, once a stable design structure was available and accepted. A solution used in several cases (usually unintentionally) was to abandon the specifications once this level had been achieved. Otherwise, the projects found themselves maintaining two increasingly divergent sets of documents.

- Lack of management understanding of software was frequently cited as a major

obstacle. Many engineers were faulted for "drawing pictures" rather than generating code. In one case, management mandated the use of the methods to a strongly resistant team; results in this case were equally poor.

Although the concerns listed above are generally applicable to all of the methods, there were a number of method-specific concerns discussed as well. These are elaborated in the answers to evaluation questions in Appendix C.

On the positive end, those projects that had training, management, and tools that were at least considered adequate found many benefits in use of the methods. For the most part, these users are continuing to apply the methods on new projects. Some of the common observations include:

- The rigor of the methods made requirements much more visible. Discovery of gaps, inconsistencies, and incorrect statements in the customer requirements was universally cited as a major benefit.

- Most projects stated that integration, maintenance, and supportability of the software was noticeably improved due to use of the methods.

- Some projects found that their overall life-cycle was shortened, while others noted merely a shift in emphasis from coding, test and integration to specification and design.

- Behavioral modeling was considered extremely valuable in defining real-time requirements.

- Communication among the technical team was substantially improved.

- Simplicity and clarity of the graphical notations was called out as being of significant benefit.

Again, a number of method-specific benefits were cited as well, and these are elaborated in the answers to evaluation questions in Appendix C.

5.4. Summary of Evaluation Questions and Answers

[Wood88] presented a general approach to the assessment of a given method. A series of questions is provided in five broad categories summarized as:

- System Characteristics

- Implementation Constraints

- Method Usage Characteristics

- Management Issues

- Problem Workarounds

We applied what we learned from the sample problem exercises, observations from the field, and our own experiences to these pre-defined questions. Each question and its associated answer is provided in Appendix C.

In summarizing the answers to the evaluation questions, we have rejected the possibility of scoring each answer, applying complex weighting factors, and calculating overall rankings. By their nature, such weightings are highly subjective and inflexible. Instead, we offer the following summary comments that can be used as a basis for project-specific weightings if the reader so desires.

- Functional Characteristics - We give an edge to *Ward-Mellor* and *Hatley-Pirbhai* for following relatively clear and well-established data flow diagramming conventions. We find the conventions of *ESML* and *Harel* slightly less appealing, but this is primarily personal preference. Both notations seem rather more complex than is necessary.

- Behavioral Characteristics - We give an edge to *Harel* for the elegance and richness of statecharts, although tabular formats used by some of the other methods, particularly *Hatley-Pirbhai* and *ESML*, are flexible and can be more expressive in the context of manual cross-checking and verification. A mix of statecharts and tabular formats might be useful.

- Structural Characteristics - *Hatley-Pirbhai*'s architectural modeling is superior to the other methods. We feel that *Harel*'s module charts can be used about as effectively, but the documentation does not address them sufficiently; we have seen no actual applications in practice by which to judge. *Ward-Mellor* does not provide a separate notation for system design, although their implementation models can be used in many situations in a manner similar to *Hatley-Pirbhai* architecture models. They are less expressive and complete for modeling complex system designs, particularly in terms of the depiction of physical interconnections of system modules. As described in [Bruyn88], *ESML* presently does not address structural modeling, although it should be compatible with *Hatley-Pirbhai* and *Harel* notations.

- Constraints - There are no clear leaders in this category.

- Representations - Overall, this category is a draw. For the most part the representations provided by these methods may be intermixed.

- Derivation Techniques - We give a slight edge here to *Hatley-Pirbhai* for a superior text. All of the methods provide only high-level guidelines. For the most part, the derivation techniques of these methods may be intermixed.

- Examination Techniques - We give an edge to *Harel* due to his strong em-

phasis on formal semantic definition and capacity for extensive dynamic analysis. Unlike representations and derivation techniques, dynamic examination capabilities of these methods cannot be intermixed easily. In terms of static analysis, all of the methods provide similar capabilities

- Management Issues - There are no clear leaders in this category.

- Other Issues - Again, this category is a draw overall.

The next chapter pulls together our observations from the field, the sample problems, and our own experiences with the assessments considered above and filters them to determine which aspects are of the most significance in terms of method adoption and application.

6. Recommendations

In the preceding chapters, we have attempted to draw upon the expertise of the developers of methods, software developers in industry, past investigators, and our own experiences to paint a clear picture (at least a partial one) of the state of the practice in specification methods. Most evaluations and comparisons that we have found in the literature fail to provide concrete recommendations, concluding that "all of the methods have strengths and weaknesses." Although it has been our intention from the start to provide more specific recommendations, our results do tend to support the notion that discriminating factors among the subject methods are not strong.

With that backdrop, this section provides a number of observations about specification methods in general, and particular recommendations to our primary audience—practitioners involved in method selection and adoption. A few recommendations are also provided to our secondary audience of tool vendors, method developers, and program offices.

Although the observations and recommendations are discussed in an abstract fashion, each was derived from one or more of the sources discussed in this report: actual users of the methods, the sample problem exercises, or the method assessment questions. References to the primary sources for these recommendations (documented in Chapter 5 and Appendix C) are provided in square brackets for each observation and recommendation. For example, [5.2] indicates that the primary source is Section 5.2 of this report.

6.1. General Observations

1. **These methods have been successful [5.3]**. Each of these methods has been and is being used with success on significant projects. Lacking good metrics, we must define success informally; however using methods such as these is clearly superior to an *ad hoc* development approach. Two such informal measurements are (1) the existence of working, accepted products built in part with these methods, where similar products within the same organization built without the use of the methods have failed, and (2) the opinions of the users of the methods, who on the whole have reported noticeable (if unmeasured) improvement in quality, reduced integration time, and better understanding of the system on the part of the engineers.

2. **There are prerequisites for success [5.3]**. Although there have been many successful applications of these methods, there also have been notable failures. Prerequisites for successful introduction of this technology into an organization include adequate management, training, and tool support. Management must support the technology both philosophically and financially. Personnel turnover during the critical introduction phases should be avoided as much as possible; we have noted several situations where personnel were trained early, only to be shifted to other projects just as they were becoming proficient, sometimes resulting in unrecoverable delays. Significant up-front training in the use of methods (as well as in the entire selected life-cycle

methodology) is of utmost importance. In addition, ongoing training and consultation, particularly during the initial use of the method, is highly beneficial in risk reduction; the expense of such consultation should be recovered many times in saved time during the initial project. Once internal expertise is established, consultation is likely to be of less value. Finally, tool quality can have a significant impact on project success. In extreme cases, poor tools can be much worse than no tools at all. Automated support must be selected with caution to avoid show-stopping project interference.

3. **There are few discriminators among the methods [5.3, C.a]**. Our experience indicates that any of the methods can be used successfully or unsuccessfully. Probably the most typical discriminators in method selection are personal preferences of technical leaders or project personnel who have used certain methods on previous projects. Often, religious wars result among the proponents of the different methods. We have found that the individual battles fought in these wars revolve around relatively insignificant graphical or semantic features that have little importance in the overall development scenario. Where the methods are directly comparable (functional and behavioral specification), most discriminating differences are very minor indeed. More important discriminators are found in areas where the methods are *not* directly comparable. These discriminators are discussed in the next section.

4. **These methods do not eliminate the essential difficulties of software development [5.2, 5.3, C]**. With the use of methods such as those described in this report, many of the accidental aspects of specification [Brooks87] can be mechanized and automated. Further, the use of formal methods holds the possibility of automation of certain aspects of refinement and verification. The fundamentally difficult part of software development, however, probably will never be fully automatable. [DSB87] argues "that the essence (of software development) is the designing of intricate conceptual structures, rigorously and correctly. The part of software development that will never go away is the crafting of these conceptual structures; the part that can go away is the labor of expressing them." Claims of CASE tool vendors to the contrary may be safely ignored. No tools or methods (including these) remove the conceptual difficulties of software development. What these methods *do* provide is the ability to capture those conceptual structures to make reasoning about them easier and the ability to mechanize some of the more straightforward conceptual tasks and transformations. In this way, labor can be focused on the more critical crafting tasks.

6.2. Recommendations to Software Developers

1. **If you are not using a specification method, start doing so [5.3, C.f]**. The evidence that we have seen for this advice is substantial, provided the systems under development are beyond a certain conceptual critical mass (the smaller and less complex the system, the less payback is likely to be achieved in using these methods).

2. **If you are already using one, stick with it [5.3, C.f]**. If one of these methods

is being used with success, we have seen no evidence that would indicate that switching to one of the other methods discussed in this report will yield significantly better results. It would be more fruitful to focus on improving and maturing the procedures and techniques that are in place, such as merging components from other methods with those of the existing method.

3. **If you are selecting one, these are the key discriminators.** In selecting one of these methods, heaviest emphasis should be placed upon:

   - **Compatibility with your software process and life-cycle goals [C.a, C.f, C.g]** - In particular, all of the methods are equally capable of specifying functional and behavioral requirements. Given an existing scheme for specifying system architectural design, any of the methods would be sufficient for system and software specification. Lacking such a scheme, however, we recommend selection of *Harel* or *Hatley-Pirbhai*, both of which incorporate distinct notations appropriate for complete system design. Alternatively, one could successfully integrate *Ward-Mellor* or *ESML* with *Hatley-Pirbhai*'s Architectural Models or *Harel*'s Module Charts. More important than the selected notation, however, is ensuring that the process of system design in fact takes place when appropriate.

   - **Quality and capabilities of tool support [5.3, C.d, C.e, C.f]** - The capabilities desired of a toolset must be understood before an appropriate selection can be made. Although this report is not the appropriate forum for the discussion of various tool capabilities, it is important to consider those capabilities that may be important for a particular application, problem domain, or development process. Examples of tool features that might drive method selection include: dynamic analysis, executable specifications, animation, prototype code generation, documentation interface, user interface prototyping, and integration with other method notations and techniques.

   - **Availability of adequate method training [5.3, C.f]** - We recommend a minimum of one to two weeks of up-front training in the application of each method. We have noted that training has been most effective where the students are knowledgeable in software engineering concepts. Less time should be required for tool-specific training. Available training vendors should be examined to determine the appropriateness of course materials and the expertise and skills of the instructors. The availability of satisfactory training courses and materials should be considered an important factor in method selection.

   - **Availability of in-house or external consulting expertise [5.3, C.f, C.g]** - Initial training is important in providing an appropriate framework for methods use, but is insufficient for the efficient ramp-up of knowledge from "novice" to "expert." Users of these methods involved in their first application often spend considerable time agonizing over decisions which are actually unimportant. This phenomenon is primarily attributable to a lack of self-confidence in the use of the methods, and is

usually alleviated after a project has been successfully completed. The best way to deal with the problems of the novice is to provide for expert consultation (on either a full- or part-time basis, depending on need) to answer questions and to participate in artifact reviews and walkthroughs. Thus, the availability of such expertise should be considered an important factor in method selection.

- **Personal preferences and experience of the project team [5.3, C.f, C.g]** - Based on previous training or experience, project leaders or staff may have initial biases toward one or more methods. For these methods, there is little to be gained in fighting those biases unless the discriminators discussed above are significant enough to so warrant .

4. **If the above are equal, pick the best documented and simplest method.**

   - For general systems (mix of functional, behavioral, and structural characteristics), we lean toward *Hatley-Pirbhai* due to the quality of the available text and the overall simplicity and flexibility of their notations and techniques [5.2, C.a, C.g].

   - For systems with complex behavioral interactions, the *Harel* notations may be preferred. Statecharts are particularly well-suited to modeling complex timing interactions often found at lower levels of specification. A mixture of statechart notations with *ESML*, *Hatley-Pirbhai*, or *Ward-Mellor* functional notations and conventions would probably work well, but is not presently supported by automated tools. Further, *Harel*'s emphasis on semantic definition and dynamic analysis may be highly beneficial where adequate tool support is available [5.2, C.a, C.c, C.e].

5. **Invest heavily in the selection process [5.3].** Selection of methods and tools is a very difficult and costly process. Initiate the evaluation and selection process well before the target project is initiated. Anticipate needing at least six months for nominal evaluation of three or four methods and tools.

6. **Invest heavily in up-front training [5.3, C.f].** Based on our observations, projects that attempt to save money by skimping on training have experienced a higher incidence of project turnover and failure than those where training was emphasized. Projects should invest in at least one week of up-front training for the selected method (exclusive of training in other methods, such as design and verification techniques).

7. **Do not confuse method training with tool training [5.3, C.f].** Tool vendors usually provide some level of training, often at no cost, in the use of their toolset. Toolset training by itself is useful, but is also required. Tool vendor trainers often do not have the kind of expertise in complex real-time applications required for knowledgeable training. Further, they will almost certainly be biased, and they may fail to indicate where their tool does not support the methods in question.

8. **Invest in expert consulting for the first application [5.3, C.f, C.g].** Even with up-front training, software developers usually do not have the experience or the confidence needed to pursue the effort efficiently. The small cost of an expert consultant should pay large dividends in avoiding wasted time on trivial issues and in increasing confidence, consistency, and conformity of the developers. Many of the members of failed projects have observed that a lack of available expertise was a primary contributor to project failure.

9. **Beware of tool and method hype.** Claims of reduced development time, cost, and effort attributed to methods or tools may not be based on the results of unbiased assessment or empirical evidence. For a more balanced assessment, we recommend a review of the observations of experienced users, such as the summary provided in Section 5.3 of this report.

10. **Invest up-front in problem resolution and workarounds [5.3, C.g].** All methods and tools have weak points. Many of those weak points have been pointed out elsewhere in this report. To avoid poorly-timed, show-stopping problems, effort should be spent in advance of the first application of the selected method and tool to identify specific problem areas and manageable workarounds.

11. **Establish a process group [5.3, C.f].** Selecting and implementing a good specification method will not guarantee project success. All technical areas, including design, implementation, verification, and maintenance must be addressed, as well as various management and administrative issues. A good place to start in gaining control over the software development process is to establish a software engineering process group (SEPG) within the organization [Humphrey87]. Once a stable process, methodology, and environment are established, the organization can focus on areas of particular technical risk for each project with efficiency.

6.3. Recommendations to Tool Vendors

1. **Concentrate first on supporting methods as defined [5.3, C.a].** Few available tools completely and correctly support these methods. We believe that it is important to the success of the customers, and ultimately to the health of the CASE vendors and industry, to fully support the syntax and semantics of the chosen methods.

2. **Improve operational aspects of tools [5.3, C.c, C.d, C.e].** Almost as important as full method support is human factors engineering. A CASE toolset priced perhaps in the hundreds of thousands of dollars should be as easy to use, flexible, and reliable as the desktop productivity tools currently available on personal computers for a few hundred dollars. Without a usable and complete toolset, secondary features such as automatic code generation and automatic document generation are of very limited value.

3. **Investigate increased flexibility [5.3, C.a].** Lacking definitive standards in

notations and conventions, maximum utility requires open architectures and flexible syntax and semantics. For example, a flexible tool would permit the user to select and intermix any of the graphical or textual notations of any of the methods discussed in this report. Such a capability may be far off.

6.4. Recommendations to Method Developers

1. **Document specific requirements for tools vendors and selectors [C.c, C.e].** It is not entirely reasonable to expect tool vendors to fully support a method if the syntax and semantics of that method are not clearly defined. *Hatley-Pirbhai* has done a good job of defining the *static* requirements and options expected of a tool to claim support of the method [Hatley88]. By contrast, the other methods, particularly *Harel*, have done a good job of describing formalized *dynamic* semantics to varying degrees. Both types of information are important and necessary. We strongly recommend that all method developers provide both types of description.

2. **Focus on issues of transition to design [5.3, C.d, C.g].** Although the essential difficulties of extracting a design from a requirements statement probably cannot be overcome [DSB87], many of the mechanical aspects of deriving first-cut designs can be approached. Only *Ward-Mellor* has addressed this issue to any significant depth. Better descriptions and examples of transformation into structured designs and object-oriented designs are needed. A related issue largely ignored is that of verification of the design against the specification.

3. **Focus on maintenance of specifications [5.3, C.b, C.g].** Many users of these methods have found that the resulting specifications are very cumbersome to maintain. Some users abandon the specifications entirely when a stable design becomes available. This problem exists at least in part due to the difficulty in changing from specification to design, as discussed above. Maximum benefits cannot be obtained from these methods without some assistance on this issue. Of particular concern is the impact of this problem on long-term maintenance of the software (post-delivery), considering that one of the major strengths of these methods is reported to be improved maintainability and supportability.

6.5. Recommendations to Program Offices

1. **Require ongoing software process evolution [5.3, C.g].** A stable and maturing software process, of which these and other methods are only a part, is critical to reduced-risk software development [Humphrey87]. The existence of such a process is more important than the individual methods, tools, procedures, and documentation standards that comprise the process. Contractors should be required to prove in their proposals the establishment of an evolving process.

2. **Require proven track record in methods [5.3, C.f, C.g].** Given the widespread failure of software projects lacking repeatable methods and tools, program offices should require that contractors show evidence of successful development of software. This evidence should be based not only on high-quality staff (which is not usually repeatable) but also on established use of methods such as those discussed in this report. If the selected contractor does not have a record in the use of methods, the program office should anticipate and require up-front training and ongoing expert consulting as described elsewhere in this report.

3. **Require ongoing tool/environment evolution [5.3, C.f, C.g].** Program offices must be aware of the high cost of selecting and establishing new tools and environments. The selection process requires extensive time (perhaps six months to one year) that should be planned into the program schedule unless the contractor can demonstrate that an adequate environment and set of tools is in place. Similarly, the establishment of the environment, which may include changes in management and technical procedures, philosophies, and styles, come only at high initial cost. Our experience indicates that the required time probably cannot be significantly or efficiently reduced by increasing available manpower.

4. **Review technical content, not just schedule and costs [5.3, C.e, C.f, C.g].** Reviewers need appropriate domain expertise and at least some familiarity with the methods being employed; in addition, early review of requirements specifications will increase the likelihood of successful validation of the deliverable product.

5. **Ensure compatibility of methods and tools among multiple contractors for a program [5.2, C.e, C.f].** The integrating contractor should be tasked to show definitively that effective compatibility can be achieved. Otherwise, difficulties in integration and maintenance of incompatible artifacts are likely to occur.

7. Conclusions

The purpose of this report is to provide relevant information to assist engineers in the selection of methods. The report describes our approach to the comparison and evaluation of some of the more popular specification methods currently in use and provides recommendations that we hope are useful to current practitioners. As such, the report reflects current state of the practice and does not make value judgements as to the effectiveness of this set of methods versus approaches that may be on the horizon.

Likewise, this report does not attempt to compare dissimilar methods. Methods can be categorized into several broad classes reflecting, for example, the primary vantage point from which one views a software artifact. These classes are often orthogonal to, or at least significantly unlike, one another. The almost complete lack of metrics against which to measure classes of methods makes speculation as to which class is "best" highly subjective and essentially meaningless. For this reason we have chosen to examine methods belonging to roughly the same class.

Existing literature generally fails to consider the problems associated with the application of methods to large-scale or highly complex systems. We deliberately set out to avoid this failing by interviewing method users from a variety of application domains, and by involving ourselves in consultation with a moderately large project during its development. It is significant to note that our results do not arise from a controlled experiment. The negative aspect of this is that determining the factors resulting in success or failure on a given project is a risky proposition due to the impurity of the subject environment. The positive aspect of the uncontrolled conditions is that our observations are free of academic bias; we have reported on the experiences and observations of actual users suffering under actual adverse conditions. For those who must make method selections *today*, we believe it is these types of experiences that have the most potential benefit.

Software developers indicated that difficulties in management, training, and tool support were more important than method-specific problems (see Section 5.3). In citing method shortfalls, the developers discussed:

- Difficulty in determining the appropriate level of detail
- Confusion over the transition to software design
- Ineffective communication with non-technical personnel
- Difficulty in maintaining specifications over the project life-cycle

Balancing these problems were the many benefits of using these methods, including:

- Rigor of the notations made requirements much more visible
- Integration, maintenance, and supportability were improved

- There was a shift in emphasis from life-cycle "back-end" to "front-end"
- Clear definition of behavioral requirements was possible
- Improved communication among technical personnel was achieved

We applied what we learned from the users and from the literature to a predefined set of evaluative questions. Our results indicated that each of the methods can be used effectively within a suitable life-cycle and process framework. We provided recommendations for specific selection alternatives based on large-grained factors such as process compatibility, tool capabilities, and the biases of project staff. Additional discriminators included the quality of available texts, training, and expert consultation.

We also provided specific recommendations on important issues faced by software developers, tool vendors, method developers, and program managers. These recommendations, made in conjunction with a thoughtful evaluation of other methodology components (e.g., design methods) and of automated support, should assist organizations in making effective selections for current and future projects.

Appendix A: Special Topics

The following topics are related to the central issues of this report but are outside of its scope. To provide background rationale, we briefly discuss a number of these issues.

A.a. Brief History of the Subject Methods

A wide variety of traditional specification methods (such as Jackson System Development (JSD) [Jackson83], System Requirements Engineering Method (SREM) [Alford77], and Entity-Relationship modeling [Chen76]) have been proposed or used over the past two decades. However, the traditional methods that are in most common use in industry today are variants on the concepts generally referred to as *Structured Analysis*.

Structured Analysis was pioneered by Doug Ross in the early to mid-1960s, and was popularized in the mid and late 1970s by Ross (as SADT) [Ross77], Tom Demarco [DeMarco78], and Gane and Sarson [Gane79]. Each of these three variants gained acceptance primarily in the business data processing field. In general, Structured Analysis techniques focus on hierarchical data-flow analysis of requirements, popularly called the *functional* view of requirements because data flow diagrams depict the flow of information from one function to another within the system.

While functional modeling proved useful and even desirable in the data processing community, it was largely rejected by the real-time systems community as an insufficient statement of requirements. Real-time system developers required means for describing not only the functional components of the system, but also the dynamic *behavior* of the system. In addition, it became clear that basic structured analysis techniques did not provide a mechanism for capturing the details of system/subsystem *structure* required for a complete system specification.

In response to these inadequacies, in the early and mid-1980s several independent methodologists proposed variations on basic structured analysis intended to address some of these concerns. The most prominent enhancements were introduced by *Ward-Mellor* [Ward85] and *Hatley-Pirbhai* [Hatley87]. The former was developed at Yourdon, Inc., while the latter was developed jointly by Boeing Commercial Aircraft and Lear Siegler, Inc. At about the same time, *Harel* developed a unique behavioral notation called *statecharts* as a result of a consulting relationship with the Israeli Aircraft *Lavi* project. This notation was extended by Harel, Amir Pnueli, Michal Politi, Rivi Sherman, and others to include functional and structural components similar to those found in *Ward-Mellor* and *Hatley-Pirbhai*. A fourth notation, *ESML* [Bruyn88], was introduced in 1987 by a group including representatives from Boeing, Hughes Aircraft, and Honeywell, as an attempt to standardize a notation that captures many of the benefits of both Ward-Mellor and Hatley-Pirbhai notations. Other methods of similar capabilities were proposed but are less well known.

A.b. Appropriateness of Specification Phase

Battles continue to be waged over the need for a distinct specification stage in the software development life-cycle. [DSB87] states that specifying requirements is the most crucial part of any software development effort. Failing this, the remainder of the effort is more likely to exhibit errors, delays, and cost overruns.

Many of the disputes focus on the idea that there is a significant potential for wasted effort when a specification approach is taken that is substantially different from the ultimate design approach. We do not argue with this idea. We merely note that, for systems of significant size, there is a clear necessity to focus on implementation-independent analysis at some point in the development process. Extensive feedback and iteration is of obvious benefit, however, and the advantage of compatibility between specification and design notations and techniques is indisputable and self-evident.

The purpose of a separate specification is not to provide additional work for engineers, but rather to prevent them from becoming delayed by design details when it is advantageous to focus on more abstract issues. We have found through our field investigations (discussed in this report) that many projects fail due to overzealous specification of requirements, but that even more projects fail due to the lack of a distinctive specification effort. The key to success seems to be the ability to strike the right balance between the two. Unfortunately, the right balance can vary significantly among projects and application domains.

A.c. Formal Methods

There exists a class of specification and design methods that are generally referred to as *formal methods*. These methods permit the specification of systems and software with mathematical precision. Such techniques allow the analyst to reason about the specifications to prove qualities such as correctness, consistency, and absence of deadlock. For the most part, these kinds of techniques are not yet widely used in practice, particularly in the United States.

The focus of this report is on methods that have a formal basis, but are not rigidly formal in all aspects of notation and technique. These types of methods have been termed *semi-formal* or *traditional* methods. We have chosen to focus on these kinds of methods for several reasons:

1. They are relatively easy to understand, and hence the perception is that they are likely to be more readily adopted into practice in the near-term.

2. They have achieved more wide-spread application in industry. There is an historical track record of the use of these kinds of methods on real, large-scale projects. We felt it would be useful to examine this historical evidence.

3. They form the basis of the existing and expanding CASE market. Companies

are engaging in expensive commitments to particular products, and we hope to provide some basic guidance in at least part of the CASE tool selection process.

4. A separate effort is underway at the SEI that is focusing on examination of formal methods.

A.d. Object-Oriented Development

For the past few years, one of the prevalent buzzwords in the software engineering community has been *object-oriented*. It represents yet another term that has different meanings to different groups of people. Much has been written on object-oriented design ([Booch83b], [Buhr84], [Cherry87]), but relatively little has been written on object-oriented requirements analysis and specifications. Although this is a topic of recent increased attention ([Seidowitz87], [Bailin89], [Hoza89], [Bulman88]), most work that we have seen has focused on the transformation of functionally-oriented specifications into object-oriented designs. Many of these techniques require the developers to perform an initial analysis using some variant of basic structured analysis (e.g., as described by Demarco, Gane-Sarson, or Yourdon), or one of the real-time extensions discussed in this report. Having developed a functional specification, the developers are then provided with guidelines for identifying various object classes in the specifications, and carving the specification into design objects. Although this work is useful, we classify it as *transformation techniques* rather than specification techniques.

Other recent literature ([Shlaer88], [Ward89b]) has emerged that attempts to formalize information modeling as the basis of object-oriented analysis. With the current industry focus on object-orientation, we expect to see more of this type of discussion over the next few years.

Because we considered this technology to be relatively immature, we chose not to investigate object-oriented requirements specification at this time, although such topics may provide fertile ground for future evaluation.

A.e. The Relationship of Methods and Tools

The distinction between methods and tools is a moving target. This is particularly problematic in that most practitioners are introduced to methods only indirectly, through the use of some toolset. Although adequate automation is critical to the continued growth of methods use in industry, we feel it is not only appropriate, but vitally important to examine methods and tools separately. This is true primarily because few tools fully support the methods, and the tools and methods evolve independently of one another. For example:

- Due to changes in host platforms, the emergence of a better toolset, or the inadequacy of the existing toolset, a new toolset may be acquired.

- Engineers transferred to, or hired for, the project will have been trained on different toolsets.

- The toolset will undergo multiple revisions and upgrades resulting in substantial changes in method support.

It is important that engineers receive training in the fundamental aspects of the underlying methods to facilitate:

- Identification of tool deficiencies and workarounds to those deficiencies.

- Adaptation to a different toolset, with its concomitant deficiencies and workarounds.

- Adaptation to revisions and upgrades to either the toolset or the methods.

Such method training, in addition to the expected tool-specific training, will enable the team members to recognize deficiencies and react appropriately.

Other evaluative reports ([SPS88], [IDA87], [CECOM89]) generally have not made a clean distinction between methods and tools. Although one of our method identification criteria was the availability of commercial automation, our intent has been to focus on methods independent of existing automation for the reasons discussed above. Further, inclusion of tool-specific evaluation would render this report obsolete by the time of publication due to the rapidly evolving state of the CASE market.

Appendix B: Affiliates Survey Results

This section presents the results of the affiliates survey discussed in Section 3.2. Each of the five questions of the survey are listed along with a tally of responses and a brief discussion. The tallies are presented as percentages. Please note that respondents were free to reply to more than one category, so the total percentages may exceed 100%.

Note that this discussion is not intended to present a professional statistical analysis of results, but rather a simple report of tallies. Brief discussions of the results are provided to tie the survey into this report; however speculation regarding the results generally are left to the reader.

Of 447 questionnaires, 122 (27%) were returned.

QUESTION 1: **What type of real-time software is developed by your organization?**

The purpose of this question was to determine the breadth of application domains faced by our affiliates, as well as to provide cross-correlations between application domains and methods used. Nine different application domains were listed, along with space to write-in additional domains. The results are summarized below:

41	**C^3I**
40	**avionics**
38	**navigation/control**
38	**electronic communications**
28	**satellite/telemetry**
23	**shipboard**
19	**battlefield**
19	**ATE/diagnostics**
15	**robotics**
6	**simulators (write-in)**
3	**industrial control (write-in)**
10	**others (all write-in, 2 or less each)**

It is notable that the spread of applications indicates that there are no particularly dominant types of real-time systems among our affiliates. Although C^3I and avionics topped the list, several others were nearly as high. The results of the survey would seem to apply to a wide variety of problem domains, which is probably representative of our target audience.

QUESTION 2: **In developing these systems, which of the following specification methods are used?**

For this question, six methods that appear regularly in the literature were listed.

The results of this question are summarized below:

38	**Object-Oriented Analysis**

```
31      Harel
31      Ward-Mellor
29      Hatley-Pirbhai
 7      Structured Analysis (write-in)
 6      none (write-in)
 5      Systems Requirements Engineering Method (SREM)
 3      Jackson System Development (JSD)
 1      Extended Systems Modeling Language (ESML)
17      others (all write-in, 2 or less each)
```

Interestingly, more respondents reported using object-oriented analysis than any other method, despite our feeling that object-oriented analysis methods were relatively new and immature at the time of the survey. There are many possible explanations for this phenomenon, but we will leave speculation to the reader. Appendix Section C.d provides a brief discussion of object-oriented development that may provide some insight.

Of more significance for our purposes is the response on the three subsequent methods, Harel, Ward-Mellor, and Hatley-Pirbhai. These methods are very similar in approach and, along with object-oriented analysis, they seem to be overwhelmingly the most popular methods.

QUESTION 3: **In developing these systems, which of the following design methods are used?**

Five well-known methods were listed for this question, along with space for write-in responses. The intent of this question was to use the responses as a basis for the possible future selection of *design* methods for evaluation. The results are summarized below:

```
66      Structured Design
51      Object-Oriented Design
 6      PAMELA
 5      DARTS
 3      Jackson System Development
 2      SCR (write-in)
12      other (all write-in, all two or less)
```

QUESTION 4: **In developing these systems, which of the following programming languages are used?**

Although this question perhaps is not of direct importance to the evaluation of methods, it was felt that the responses would be interesting in light of the preceding questions. Seven prevalent languages were listed. The results are summarized below:

```
74      Ada
55      C
48      FORTRAN
25      Pascal
```

```
22      JOVIAL
18      CMS-2
18      assembler (write-in)
 3      HAL-S
 3      PL/M (write-in)
10      others (all write-in, 2 or less each)
```

QUESTION 5: **Would you be willing to aid the SEI in the identification of suitable sample problems from your application domain(s)?**

The purpose of this question was to identify affiliates who might be willing to continue to participate in this study beyond the initial questionnaire. In particular, our goal was to identify sample projects for up-close examination. One of the main goals behind this study is the examination of real-world experiences with the selected methods. Examination of artifacts from actual projects, and interviews of actual users of the methods would make possible evaluations that are much more meaningful and believable to our affiliates and the software engineering community in general. See Section 5.2 for further discussion.

The results are summarized below:

```
56      yes
26      maybe
18      no
```

Appendix C: Evaluation Questions and Answers

[Wood88] presents a general approach to the assessment of a given method. A series of questions are provided in five broad categories summarized as:

- System Characteristics
- Implementation Constraints
- Method Usage Characteristics
- Management Issues
- Problem Workarounds

In the following subsections, we have extracted a subset of the questions from [Wood88] that we consider appropriate for this evaluation. Some questions have been eliminated, while others have been modified slightly. A few new questions have been added as well. Data gathered from the sample problems and project interviews has been used as the primary source for answers to these questions, although we do not generally differentiate between the sources.

As a general note, many of the evaluation questions consider the *explicit* support of a particular method for a given capability or characteristic. The reader should keep in mind that although a method may lack explicit support, in many cases the method nevertheless can be used to model the desired capability in question. In these cases, some amount of forethought and customization is required on the part of the user. For example, although none of the methods provides a specific notation for modeling error-handling capabilities, the notations that they do provide are sufficiently general to permit such modeling. In other words, the methods would not *inhibit* a creative engineer from performing necessary customizations. Because all applications have domain- and project-specific attributes, such flexibility becomes an important factor.

The questions in the subsequent subsections are organized into the categories listed above. Each category is divided into subcategories and then into individual questions. Please refer to [Wood88] for clarification on the question structure.

Section 5.4 summarizes the answers.

C.a. System Characteristics

Overall Characteristics

1. **Does the method allow the representation of the functional, behavioral, and structural views?** As previously shown in the classification tables, all of the methods support representation of the functional and behavioral views of the system, using data flow and finite state representations, respectively. Only *Hatley-Pirbhai* and *Harel* offer distinct representations for system design. These notations are critical for complete system definition, although they are less important for software-only specification.

    - *Hatley-Pirbhai*: The functional notations (Flow Diagrams) are the simplest of the four methods' notations. Semantics generally are straight-forward and clear, with very few components. Behavioral notations are flexible, allowing any combination of tables and diagrams, although in practice state transition diagrams are used infrequently. Users consistently cited confusion in understanding multi-sheet control specifications. System structure modeling is supported with Architecture Models, which allow the capture of hardware/software allocation decisions and interface specification. *Hatley-Pirbhai* notations appear to scale up well; the optional splitting of Flow Diagrams into Data Flow Diagrams and Control Flow Diagrams is practical for large-scale, control-intensive systems; emphasis on state tables rather than diagrams is practical for flexible modeling of complex behavior.

    - *Harel*: The functional notations (Activity Charts) are sufficient, but by our observations they can result in more clutter than *Hatley-Pirbhai* notations, primarily in the more complex notations for merging/splitting of flows, and combined data/control flow can be problematic for large-scale systems. The behavioral notation, statecharts, is one of the major contributions of *Harel*'s work to specification. Statecharts provide excellent complexity management through a graphical state hierarchy, and permit extensive and elegant expression of behavioral patterns. The structural notation, Module Charts, can be used in a fashion similar to *Hatley-Pirbhai* Architecture Modeling, although they are less expressive and apparently not yet widely used. Model semantics are well-defined formally.

    - *Ward-Mellor*: The functional notations (Flow Diagrams) are essentially similar to *Hatley-Pirbhai*, although the integration of control notation tends to result in more clutter. We find the graphic inclusion of process triggers and activators intuitively appealing. However, the graphic inclusion adds very little additional information to the overall model, while dramatically increasing clutter. The behavioral notations consist almost entirely of state-transition diagrams, which are inadequate for the expression of all but the simplest state machines. These are less flexible than the *Hatley-Pirbhai* and *ESML* tabular formats, and far inferior to *Harel*'s statecharts. *Ward-Mellor* does not provide a distinct structural notation for system design, although it would be compatible with either the *Hatley-Pirbhai* or *Harel* notation. Implementation modeling can be used to an extent for the same purpose.

- *ESML*: The functional notations and semantics are very similar to those of *Ward-Mellor*, with minor graphical differences and the capability of placing graphic process controls on separate diagrams. The behavioral notations borrow some of the tabular formats of *Hatley-Pirbhai*, along with state-transition diagrams. Some notations are added to the basic *Ward-Mellor* trigger/activation features. *ESML* does not provide a structural notation for system design, although it would be compatible with either the *Hatley-Pirbhai* or *Harel* notation. A structural notation extension is planned.

2. **Are the views complementary?**

   - **Can there be integrated views of function and behavior as well as independent views?** - All of the methods integrate functional and behavioral views by means of control flow. *Harel*'s statecharts are most capable of acting as a standalone model of behavior due to their rich semantics and hierarchical nature. The other methods rely heavily on the functional model to provide the framework for distribution of the behavioral model, and in fact this same approach is recommended by Harel as well [Harel88b].

   - **Are there suggested techniques or rules for deriving the structural view from the functional and behavioral views?** - *Ward-Mellor* and *ESML* do not provide a distinct notation for system design. *Ward-Mellor* provides techniques for deriving an implementation model from an essential model. *Harel* provides essentially no guidance for the derivation of the structural model. *Hatley-Pirbhai* devote about a third of their text to the structural notation and its derivation, although the suggested techniques comprise fairly high-level guidelines.

Functional Characteristics

1. **Does the method provide a representation that clearly draws a boundary around the system and separates it from its environment?** *Ward-Mellor*, *ESML*, and *Hatley-Pirbhai* all use a special flow diagram called a *Context Diagram* to draw a boundary around the system. For *Harel*, a top-level Activity Chart provides the same effect. Each of the methods provides the ability to clearly identify external entities.

2. **Does the method allow the representation of data that flows across these interfaces using an appropriate level of abstraction?** All of the techniques provide for abstract definition of external interfaces. *ESML* and *Ward-Mellor* are limited to modeling the software aspects of interface and protocol, while the structural modeling facilities of *Harel* and *Hatley-Pirbhai* permit more detailed specifications at the system level. For example, the latter two are most suitable for the specification of rigidly defined interfaces to existing devices.

3. **Does the method provide a technique for representing each process, including its inputs, outputs, functions, and the exceptions that it may raise?** All of the methods support process descriptions defining inputs and outputs. Functional descriptions are not usually formal. Although the notations could be compatible with formal specifications, none of the methods generally recommend formalisms for process descriptions. "Structured English," decision tables, mathematical formulae, or any other type of specification may be used. *Ward-Mellor* does discuss the use of formal notations, such as pre- and post-conditions, although none of the developers that we interviewed were using them.

4. **If mathematical algorithms must be devised, does the method provide a representation that is familiar to the algorithm developers and can be understood by the algorithm implementors?** Mathematical notations may be used with any of the methods, although a mapping is required between variables within a formula and the data flows that they represent.

5. **Do the representations allow specification of functionality under adverse conditions such as loss of data or single sensor failure?** None of the methods specifically addresses unusual conditions. This type of processing is specified as any other type, and may require some adaptation on the part of the analysts to avoid unsightly flows across multiple levels of diagrams. All of the methods could be improved in this area, although this is not a discriminating factor in choosing among them.

Behavioral Characteristics

1. **Does the method incorporate the concept of describing the behavior of the system using a state-oriented model?** Yes, all of the methods use some form of state modeling for behavioral description.

2. **Does the representation of the model include the representation of events, actions, states, transitions, and conditions (guards) dictated by the functional environment?** All of the notations support the representation of states, transitions, and actions. Only *ESML* and *Harel* support both events (time-discrete) and conditions (time-continuous) as mechanisms of transition from state to state. *Ward-Mellor* requires the use of events to trigger transitions, which may result in the necessity for more interim states and transitions. In contrast, *Hatley-Pirbhai* generally requires the use of combinations of conditions to trigger transitions. *Hatley-Pirbhai* models can generate events through process specifications (using the keyword "issue"); this is not entirely satisfactory in that the control specifications do not visually distinguish between the two types of signals, making interpretation of the model somewhat difficult. *Hatley-Pirbhai* notations do not distinguish time-discrete external events.

3. **Is the model appropriate for the complexity of the system under** develop-

ment? Simple state-transition diagrams are adequate only for moderately small systems of low behavioral complexity. As such, modeling of complex behavior can be awkward with *Ward-Mellor*, which places heavy emphasis on this type of diagram. The state-transition tables, state-event matrices, and process-activation tables available under *Hatley-Pirbhai*, and also *ESML*, are more practical for modeling complex behavioral relationships. *Hatley-Pirbhai* multi-sheet behavioral specifications are useful but complex to build and difficult to comprehend. *Harel*'s statecharts are ideally suited to modeling complex behavior patterns that can be cumbersome using the other techniques, and they scale up well for larger systems.

4. **Can the representations for complex models be partitioned to help developers deal with complexity?** *Harel*'s statecharts are easily partitioned according to their own hierarchy. *Hatley-Pirbhai* notations can be partitioned into separate, but cooperating, tables and diagrams in multi-sheet specifications, but hierarchical partitioning is generally "slaved" to the functional hierarchy, which is less elegant and flexible than the statecharts approach. The latter is also true of *Ward-Mellor* and *ESML* notations. *Ward-Mellor* describes a mechanism permitting limited, behavioral hierarchy [Ward86].

5. **Can stimulus/response relationships be represented in a time-dependent manner?** *Hatley-Pirbhai* is the only method that addresses specification of stimulus/response timing requirements, although the mechanism ("Timing Specification" tables) is cumbersome and not easily derived. The other methods do not address external timing requirements to any significant degree; however, the same kind of approach could be used with them.

6. **Does the method allow representation of the relationship between the behavioral model and the functional model?** All of the models integrate functional and behavioral views in two ways: flow of control and process controls (process activation and triggering mechanisms). *Ward-Mellor* and *ESML* depict process controls graphically, while *Hatley-Pirbhai* and *Harel* do not. This "graphic causality" [Ward89a] is somewhat attractive initially, but in practice results in diagrammatic clutter and is not of essential value. To mitigate the problem of clutter, *ESML* permits separating the depiction of information flow from that of process control.

7. **Does the method allow the representation of periodic and aperiodic events?** Yes, although the method notations do not distinguish between periodic and aperiodic events graphically.

8. **Does the method allow the representation of discrete and time-continuous data?** Yes. *Ward-Mellor* and *ESML* distinguish between discrete and continuous flows graphically, while the others do not.

9. **Does the method allow the representation of input rates and bounds on those rates?** Yes, this information can be maintained in the data definition repositories supported by each method.

10. **Does the method allow the representation of concurrent processes?** Yes. In each of these techniques, all of the processes are potentially concurrent. By contrast, it is usually necessary to explicitly state required sequentiality. *Harel*'s statechart notation provides extensive capability for modeling the timing patterns of concurrent processes.

11. **Does the method provide representations that capture performance** requirements? *Hatley-Pirbhai* captures external timing constraints in a "Timing Specification" which specifies response times from a given input signal/event to an output signal/event. The method requires that each external flow be represented in the Timing Specification, whether or not timing requirements actually exist; however, there are no guidelines provided for deriving the stimulus-response paths. Examples of the specifications are somewhat informal. *Harel*'s methods provide for extensive internal timing and synchronization specification, with formalisms provided for such mechanisms as timeouts. By contrast, *Hatley-Pirbhai* allows a concept termed "universal access to time," which permits the reference to time (either relative or absolute) without formal semantics. Neither *Ward-Mellor* nor *ESML* capture performance requirements. None of the methods provides manual techniques for analyzing performance requirements.

12. **Does the method assist the developer in handling exception, fall-back, and recovery conditions?** None of the methods provide specific assistance in the specification of exceptional conditions. Exceptional conditions and processing must be modeled explicitly using the same notations as normal conditions. Many users have overcome this problem by "customizing" the notations to permit the depiction of exceptional processing independently without explicit control and data flow routing.

Structural Characteristics

1. **Does the method provide representations that describe all elements of the hardware system?** *Hatley-Pirbhai* provides representation of all types of components with the combined Requirements and Architecture Models. Generic notations are used to depict the components graphically, while Architecture Module Specifications describe allocation decisions and provide traceability to the functional view. Interface diagrams and specifications provide flow allocation to specific hardware channels. *Harel*'s Module Charts provide a similar capability, although the notation is more general and their use is not well defined. Neither *Ward-Mellor* nor *ESML* supports system structure modeling with a distinct notation, although [Bruyn88] hints that this will be addressed in a future revision of the *ESML* definition, and *Ward-Mellor* advocates the use of implementation models to depict processor components of the system.

2. **Does the method provide representations that detail the data and signal flow between the devices?** *Hatley-Pirbhai* and *Harel* have the capability to describe data and signal flow between devices, although only the

Hatley-Pirbhai notation differentiates between the information flows and the specific channels upon which they flow. *Ward-Mellor* and *ESML* use the same representations as in the functional view to depict structural data and signal flow.

3. **Does the method provide modularization guidelines that account for the need to map specific software modules onto specific hardware devices?** None of the methods provides much assistance in making allocation decisions, although the structural notations supported by the methods provide an appropriate vehicle for recording those decisions.

4. **Does the method provide techniques to detect and recover from failures?** None of the methods provides assistance in defining error detection and correction, fault tolerance, or dynamic reconfiguration.

5. **Does the method provide a data modeling technique, describing all entities and their relationships?** *Ward-Mellor* specifically integrates Entity-Relationship (E-R) diagramming into its modeling scheme. *Hatley-Pirbhai* makes only passing mention of the same approach. Neither *ESML* nor *Harel* addresses data modeling, but all four methods are entirely compatible with E-R diagrams. While the necessity of extensive data modeling for many real-time systems is a matter of contention, some recent work indicates that the use of E-R models eases transition into object-oriented designs.

C.b. Constraints

Integration and Test Constraints

1. **Do the representations describe the intended function and behavior well enough so that separate teams can use them to test the system?** All of the methods tend to enhance communication among technical personnel through clear and complete definition of function and behavior. In practice, extensive training is required for full benefit to be realized. Technical walkthroughs have been shown to be essential in effective use of these methods, particularly in integration and testing.

2. **Does the method provide representations that model the system's environment to allow testers to develop real-world test scenarios?** The abstract, implementation-independent nature of these methods can be problematic in most areas of testing. With the exception of higher-level verification (e.g., "validation testing" or "functional testing"), the use of these specifications in the derivation of test cases tends to be more of a hindrance than a help.

3. **Can test teams trace from requirements through the representations to develop test cases for modules and subsystems?** Trace-based testing can be very difficult. Each of the methods recommends maintenance of trace tables, but none provides guidelines for their derivation. *Hatley-Pirbhai*

provides somewhat more support of traceability at the system level than the other models, requiring that Architecture Module Specifications contain trace information from the functional model.

Evolution Constraints

1. **Do the methods' representations provide maintainers with a "road map" into the implementation that provides an overview of the system, shows the relationship of its parts, and allows them to focus quickly on areas of interest?** Each of the methods provides sufficient overview context, although in practice the functional and behavioral specifications have proved a hindrance to maintainers because the design tends to diverge from the specifications. One developer recommended "throwing away" the requirements specifications once a stable and accepted design has been produced, allowing maintenance to proceed from the design description. Another recommended conversion of the requirements models into a textual description based on primitive specifications. Neither approach seems entirely satisfactory. Users of structural modeling capabilities should suffer less from this problem in that the specifications are tied directly into physical components. Further iteration on the specifications may then map more closely to the software design.

2. **Do the representations help maintainers determine the scope of effect of a proposed change to a particular module or set of modules?** Yes, assuming that traceability is maintained between the requirements and design models. Each method makes scope of effect clearly visible.

3. **Does the modularization technique lead to architectures that accommodate small changes to the system's timing requirements without major redesign?** None of the methods addresses accommodation of timing requirements. *Hatley-Pirbhai* addresses only the depiction of required response times, and does not address design impact of these requirements. It is a fair assumption that use of various structural notations of each method will improve change accommodation.

4. **Does the method promote the notion of abstraction of hardware device functions into logical operations to support the replacement of devices over time?** *Harel* and *Hatley-Pirbhai* structural notations are sufficiently abstract to accommodate system component evolution.

5. **Does the method provide techniques for organizing its representations to support the evolution of the system into multiple versions?** By themselves, none of the methods addresses multiple versioning; however, many available tools support some variation on this theme.

C.c. Representations

Abstraction

1. **Does the method define abstraction techniques and give guidance on producing representations at various levels of abstraction?** All of the methods provide similar support for abstraction of data and procedures through leveled hierarchical representation. *Harel*'s method is the only one of the four to provide significant behavioral abstraction capability independent of the procedural abstraction mechanism.

2. **Do the abstraction techniques include the definition of balancing rules?** Each of the methods provides a similar (and intuitive) set of balancing rules to ensure consistency between the levels of abstraction.

Consistency

1. **Does the method provide guidelines for analyzing representations to ensure consistency within each representation?** All of the methods ensure internal consistency through naming conventions and balancing mechanisms.

2. **Does the method encourage the use of a common glossary or dictionary to protect against naming clashes between entities?** Each method provides some manner of common dictionary, flat in structure, as a mechanism for avoiding name clashes.

3. **If hierarchical representations are available, does the method encourage the technique of using one level of representation to derive a template for the next lower level?** For all of the methods, the derivation of one level of diagrams from another level is an integral part of the notation and techniques. Further, many available toolsets provide some crude level of automation of this relationship.

Completeness

1. **Does the method provide mechanisms to represent, examine, or understand such things as exceptional conditions, boundary conditions, error handling, initialization, fault tolerance, performance, and resource** constraints? For the most part, none of the methods specifically calls out any of these issues for examination, except inasmuch as they may be represented through functional/behavioral/structural modeling. Some users have employed minor customizations of the notations to accommodate depiction of such requirements.

2. **Does the method provide a mechanism to ensure that all of the requirements for the system have been met?** None of the methods provides mechanisms for ensuring that all requirements have been met by the design/implementation, beyond manual tracing.

Complexity

1. **Are there a manageable number of concepts expressed in a single representation?** *Ward-Mellor*, *Hatley-Pirbhai*, and *ESML* follow the conventions of Structured Analysis in suggested limits of diagrammatic complexity. Such limits result in diagrams that are generally manageable. *Harel*'s lack of guidelines is not problematic in and of itself; however, we have noted that in practice some users introduce multiple levels of poorly-partitioned diagrams that strain comprehension beyond practical limits. Users of *Harel* are well-advised to impose complexity restrictions of their own.

2. **Does the method provide techniques to partition and decompose complex representations into sets of simpler representations?** Yes, all of the methods employ similar hierarchical relationships to support "decomposition," although *Harel*'s behavioral approach is superior in this regard.

3. **Are notations semantically and syntactically simple across representations, and are the semantics and syntax relatively simple and straightforward to use?** For the most part, all of the methods provide simple notational syntax and semantics. *Harel*'s statecharts can be quite intricate and perplexing to the novice. *Hatley-Pirbhai*'s multi-sheet specifications are similarly complex. *ESML* process control notations seem unnecessarily complex in providing seven variations of "prompts"; it is not clear that fewer variations would not suffice. Similarly, the graphic distinction between depletable/non-depletable stores and continuous/intermittent data flows is not clearly of value in specifying essential requirements.

Traceability

1. **Can readers of the method's representations easily determine the paths between requirements and implementation?** Not always. Great effort must be expended to maintain traceability across the life-cycle. Nothing inherent in any of the notations facilitates this process. As far as the structural notations of the methods are carried into lower levels of design, traceability if facilitated.

2. **Does the method provide naming conventions for entities across all representations?** All of the methods follow fairly standard naming conventions across representations.

3. **Does the method provide notation for relating the name of an entity with the names of its components?** For information flows, the associated dictionary is the primary reference for relational information. Each method provides conventions for graphically merging/splitting flows, which greatly enhances the representation of relationships. *Hatley-Pirbhai* emphasizes merging and splitting flows more than the others, and is simpler and more complete in notation.

4. **Does each level of a hierarchical representation clearly identify its parent and children?** *Hatley-Pirbhai*, *ESML*, and *Ward-Mellor* each rely on a con-

ventional dot-notation numbering scheme to track hierarchical relationships. *Harel* lacks such guidelines, which can be the source of confusion among readers of the models, although, as with the other methods, parent/child representations can be identified by name.

5. **Does the method encourage recording and provide representations to record the designers critical decisions, e.g., which processes and data stores have been pulled together into which packages and which packages model real-world objects?** *Hatley-Pirbhai* is the most complete in permitting the depiction of allocation decisions at the system design level, although *Harel* structural notations can be used similarly. *Ward-Mellor* and *ESML* may follow the "essential model" vs. "implementation model" derivation approach to provide some amount of similar modeling capability. The superiority of any approach in this area is not clear.

6. **Can a time ordered sequence of events be traced through the representations to determine the behavior of the system?** Technically, event tracing is possible with all of the methods, although automation is required to make this a practical reality. Some methods presently have better automated support in this regard than others, but we do not address automation in this report.

View Integration

1. **Does the method provide techniques for relating one view of the system to another?** All of the methods relate functional and structural views similarly, through information flow and process controls. For the most part, the integration is smooth and intuitive. *Ward-Mellor* and *ESML* graphically depict process controls, which many engineers find more intuitive than the implicit relationships of *Hatley-Pirbhai* and *Harel*. Further, in *Harel*'s approach, the ability to manage independent functional/behavioral hierarchies is not well documented and can cause confusion. With these, relationships are less intuitive than where the behavioral model is "slaved" to the functional model. Nearly all users of all of the methods have some difficulty differentiating between the notions of "data flow" and "control flow," but these problems are usually alleviated with experience and local convention.

2. **Can developers use the representations to determine how alternatives in one would effect the others?** None of the methods provides guidelines in choosing alternatives, although *Hatley-Pirbhai* recommends de-emphasis of the behavioral model in general. Such an approach can be used with any of the methods.

Ambiguity

1. **Does the method prescribe a sequence of steps that allows developers to leave portions of a representation temporarily incomplete and** ambiguous? All of the methods permit the analyst to leave portions of the models incomplete temporarily.

2. **Does the method provide techniques for examining the representation to ensure all ambiguities have been resolved?** The syntax of all of the methods make checking for ambiguities and incompleteness mechanical.

3. **Can various audiences examine the representations to gain an unambiguous understanding of the system at the level of detail they are interested in?** In theory, this is possible with all of the methods. In practice, training and experience are nearly essential for all audiences involved in examination of models. Given that, an audience can focus on an appropriate level of detail relatively easily as a result of the leveled hierarchy approach.

Duplication

1. **Can you derive representations without expressing the same information over and over again?** Although only *Hatley-Pirbhai* addresses the topic, the same approach can be used with any of the methods: reference is made within the appropriate primitive specification to the duplicated material.

2. **Does the method provide a representation that can be used to record the existence and location of duplicated information, e.g. cross reference table?** *Hatley-Pirbhai* recommends referencing duplicated information only in the diagram and specification that are applicable, and does not discuss centralizing the information in a table.

Changes

1. **Does the method provide hierarchical forms for all types of** representation? Only *Harel* provides significant hierarchical representation of the behavioral model. All of the methods provide hierarchical functional and data representation.

    - **Can low-level changes be made without necessarily affecting high-level representations?** All of the methods permit localization of low-level changes, or at least make clear the scope of change.

    - **Can high-level changes be made without affecting all lower level representations?** Although scope of change is clear for high-level changes, it may prove difficult to localize such changes due to the top-down model of information flow.

2. **Can the repercussions of a change to higher and lower levels of a representation be traced; across different representations?** Yes, by tracing the applicable information flow.

C.d. Deriving Representations

Partitioning

1. **Does the method provide a means of partitioning the system at all stages?** Each method provides a means of partitioning at all stages covered by the method (i.e., none of them addresses software design partitioning beyond the top level to any useful extent).

2. **Do the partitioning techniques account for the structure of the problem as well as the hard implementation constraints?** The functional/behavioral aspects of all of the methods focus on depicting the problem space rather than the solution space. The structural views of *Hatley-Pirbhai* and *Harel* focus on system-level solution space, and perhaps to top-level software design implementation. "Implementation models" emphasized by *Ward-Mellor* can be derived from any of the methods using the same functional/behavioral notations, but the notations are really inadequate for detailed software design. In our view, none of the methods are appropriate for detailed software design partitioning. *Ward-Mellor* provides more guidelines than the others in carrying the specification into a detailed software design (using a conventional structural design notation).

3. **Are the suggested techniques flexible enough to allow experienced developers to examine the alternatives they believe are appropriate?** None of the methods provide techniques that are more detailed than general guidelines and heuristics. Therefore, all are flexible enough to permit the examination of alternatives.

4. **Do the techniques emphasize the need to define the interfaces between the partitions?** The techniques of all of the methods provide proper emphasis on interface definition.

5. **Do the techniques suggest partitioning the system so that pieces are independent enough to allow individual analysts/designers to elaborate and refine the pieces independently?** Yes, all of the methods provide sufficient technique flexibility to encourage work package distribution.

Refinement

1. **Does the method guide the analyst in determining the amount of detail to include at each level of a representation?** All but *Harel* provide reasonable high-level guidelines for determining sufficient levels of detail. The lack of guidelines from *Harel* sometimes results in highly detailed (design-level) models, as well as busy and cluttered diagrams that border on the unreadable. This problem is easily corrected with the application of project conventions.

2. **Is the amount of detail consistent across levels of different** representations? Consistency is highly variable across different analysts, although experience and training tend to increase consistency. This is true for all of the methods.

Evaluation of Alternatives

1. **Does the method allow the designer some flexibility when making design decisions?** Each method, used properly, gives maximum flexibility to the designer. An appropriate specification will define only essential requirements and design constraints. In practice, users find all of the methods difficult in terms of identifying the appropriate point at which to cease analysis. This puts the onus on the designer to interpret the specification not as a design structure, but purely as an abstract model of requirements.

2. **Does the method encourage the designer to generate a number of alternative designs?** Not specifically, nor do the methods hinder alternative designs, although many designers may improperly interpret the specifications as the imposition of design structure restrictions where the analysis did not so intend.

3. **Does the method help the designer evaluate alternative representations based upon system characteristics and constraints?** None of the methods provides techniques to the designer to aid in evaluation of alternative representations; however, the clear, graphic depiction of requirements and constraints makes the task more visible.

C.e. Examining Representations

Feasibility

1. **Does the method provide techniques to examine its representations, including operational prototypes, to assess risk and gauge feasibility?** By themselves, the methods do not provide much assistance in the examination of representations for feasibility. Coupled with its automated toolset, however, *Harel*'s method places a heavy emphasis on model execution and prototyping. Automated tools supporting the other methods are being extended to include similar capabilities. We have not examined any of these automated capabilities; however, many users of the methods have incorporated prototyping into their development process. These users have reported that they find the methods compatible with an early prototyping process.

2. **Does the method encourage identification of high risk items and their incremental development?** None of the methods specifically addresses risk identification and analysis, although their partitioning strategies would be supportive of a risk analysis process.

Conformance

1. **Does use of the method lead to a specification that clearly and completely defines the desired operational characteristics of the system under development?** Yes. All of the methods may be used to clearly and completely define the functional and behavioral characteristics of the system under development. Other types of operational requirements/design constraints are

not captured with any formality. In general, users capture them in the textual portion of the specification as an adjunct to the graphic models.

2. **Does the specification serve as a model of the system that can be understood by the customer and end user to insure the system under development will meet their needs?** Not usually. Users of the methods have reported that the models do not directly improve communications with the customer. Understanding the models requires nominal training in the notations and technical walkthroughs. If the customer agrees to these conditions, the models can be used for communication with limited success.

3. **Does the method help determine what questions should be asked during the examinations for conformance?** Not specifically.

4. **Does the method provide guidance in developing test cases by specifying which tests should be developed?** Not specifically.

Safety

1. **Do the method's representations describe the operation of the complete system, including software, hardware, human operators, and environmental conditions?** All of the methods can be used to describe all system components in an abstract fashion. Only *Hatley-Pirbhai* and *Harel* provide the distinct system design notations to describe the system components in a more concrete fashion.

2. **Does the method provide techniques to inspect the behavior of the system under exceptional conditions in the environment?** Not specifically. The methods provide the notational components needed for defining exceptional behavior, but lack techniques for inspection.

3. **Does the method provide specific guidelines for examination of safe operations, e.g., determining if the human operator is warned if he is taking actions leading to hazardous conditions?** Not specifically. The analysts must determine which scenarios are critical and exercise the functional and behavioral models accordingly to determine safe operation.

Walkthroughs and Inspections

1. **Is the syntax for each representation clearly defined so reviewers can quickly locate and dispense with problems with form?** For the most part. The graphical components of each method are clearly defined syntactically. None of the methods provides specific syntactic guidelines for primitive specifications. Each project or organization must define standards for conformance. The methods require only data/control flow balance.

2. **Does the method provide specific guidelines for reviewing the** representations? *Hatley-Pirbhai* provides some minimal guidelines for conducting walkthroughs and inspections of the representations. The other methods do not specifically address the issue.

Analysis

1. **Does the method provide techniques and a clear set of rules for static analysis of its representations?** All of the methods are supportive of similar static analysis of syntax and balancing rules.

2. **Is the syntax of the representations well defined, allowing the purchase or development of automated tools to perform static analysis?** The graphical portion of each method is well-defined syntactically. *Harel*'s definitions are substantially more formal and complete than the others. *Hatley-Pirbhai* provides an excellent summary of notational components required for complete automated support of their method [Hatley88].

3. **Does the method support the animation or simulation of its representations to allow dynamic analysis early in the development cycle?** *Harel* has clearly invested more effort in defining the required formalism to support dynamic analysis than the other methods. Automated support for dynamic analysis and related capabilities is more mature for *Harel* models than the others. *Ward-Mellor*, and by extension *ESML*, discusses semantics extensively in [Ward86]. *Hatley-Pirbhai* has not addressed the issue of formally defined semantics. The extent to which *Hatley-Pirbhai* models may be dynamically analyzed is unclear. For all of the methods, it should be noted that dynamic analysis tends to focus on the behavioral models (which are syntactically more formal), and not the primitive functional models.

Testing

1. **Can the specification be used to develop test scenarios that exercise the system under both normal and exceptional operating conditions?** Not easily. None of the methods provides a solid basis for most levels of testing. Possible exceptions are high-level functional testing and validation testing. The models tend to be too abstract and distant from the implementations to be useful for lower levels of testing. A more common approach is to follow requirements through trace tables to design documentation, which can then serve as the basis for lower level test scenarios.

2. **Does the method provide a technique for using the behavioral representation to generate behavioral tests?** Not specifically, although all of the behavioral model notations provide a sound basis for generating behavioral tests. Tabular notations are perhaps more straightforward in this regard.

C.f. Management Characteristics

Process

1. **Does the method provide planning techniques that lead to milestone definitions and project plans that are consistent with use of the method and the implementation language?** Not specifically, although the methods support commonly accepted life-cycle elements.

2. **Are representations clear and easy enough to understand to be used for design reviews?** Yes. The representations produced from all of the methods provide a good basis for technical design reviews. All reviewers must be trained in interpretation of the representations, however, and the availability of both domain expertise and methods expertise at the reviews provides maximum benefit from the effort.

3. **Can representations be used to comprise deliverables?** No, the representations from these methods do not constitute complete documentation by themselves, although they can form the technical core of deliverables. Supporting textual documentation, and appropriate figures and diagrams (e.g., block diagrams) are essential for a proper level of communication to the customer.

4. **Does the method help partition the system into manageable pieces that can be given out to individuals?** All of the methods are supportive of work package partitioning.

Cost

1. **Are there a number of automated tools on the market that support the method?** At present, *Ward-Mellor* and *Hatley-Pirbhai* enjoy the widest variety of support, with several vendors supporting some portion of one or both of the methods. Support for these methods is available on most major platforms, including most workstations and personal computers. From our observation, none of the presently available automation completely supports these methods. *Harel* is presently supported by only one vendor; however, this product supports the *Harel* method in its entirety, with extensive simulation, dynamic analysis, and prototyping capabilities. The variety of supported platforms is limited but expanding. At present, *ESML* is not specifically supported by automated tools, although limited support can be achieved through use of some of the *Ward-Mellor* and *Hatley-Pirbhai* tools. At least one vendor is developing an *ESML* toolset at this time, complete with animation capabilities.

2. **Does the method require a reasonable amount of training, that is proportionate to its power, and feasibility of use?** Successful users of these methods have acquired at least one week of training in method application, plus additional training in use of automated tools. Users have also stated consistently that follow-up consultation with a method expert is almost essential to success. Such an expert should be available for informal questions as well as formal reviews. We also observed that method and tool training by itself seems much less effective with those lacking sufficient education, training, or experience in software engineering concepts and issues in general.

3. **Does use of the method reduce product life-cycle costs by an amount that is worth the cost of adopting the method?** This question cannot be answered definitively without reliable metrics. No such metrics are available. Organizations that we have interviewed have stated that for the most part no savings were realized in front-end development (although a shift of effort from coding/integration/test to specification and design was frequently noted). Most interviewees described noticeable reductions in integration time and cited facilitated maintenance. Integration was improved primarily as a result of rigorous interface definition, while maintenance was improved as a result of "better software." We also note, however, that interviewees did not, in general, consider the method representations themselves to be beneficial to the maintenance process. Among the projects that we examined, we noted that developers felt that smaller projects gained less benefit from the use of the methods.

4. **Can the method be adapted for use: across a broad range of applications domains, with a variety of implementation languages, with conformance to a variety of standards?** All of these methods are general-purpose in nature and are applicable to a wide variety of application domains. We have noted some level of customization for each domain, as well as each particular project. The implication is that a high degree of flexibility in notation and technique is beneficial.

C.g. Other Issues

1. **What issues does the method not deal with?** None of the methods deals with software or hardware design. None of them deals in any depth with verification or maintenance issues. None of them provides a particularly smooth transition to design notations. None of the methods provides formal mechanisms for depiction of non-functional/behavioral/structural requirements.

2. **What are the negative consequences of using the method to solve a particular problem?** The primary technical negative consequence of using any of these methods is that maintenance of the resulting specifications is cumbersome, and occasionally detrimental to the development effort. In particular, where the requirements specifications diverge sharply from the design specifications, duplicated effort can be a problem.

Experienced Personnel

1. **Can experienced staff members use the method to capture and represent what they believe to be the key functional, behavioral, and structural characteristics of the system at all stages of development?** For each of the methods, maximum leverage is gained from experienced personnel. Although in concept the software design is not constrained by the partitioning strategy used in the functional/behavioral models, greater experience leads to more elegant, understandable specifications. Inexperienced users may become entrapped in indecision over minor semantics, trivial partioning issues, and differentiation between data and control. Experienced users

recognize more readily that a large percentage of such issues are not of significance, and decisions may be more or less arbitrary. Further, experienced personnel can effectively use the representations of the model to direct work packages to less experienced personnel, and as leverage to resolve integration issues.

2. **Can less experienced staff members use the rules and guidelines prescribed by the method to develop elaborations and refinements of the high-level representations?** The extent to which less experienced staff can effectively use the methods is limited. As a minimum, a certain level of training is required (at least one week) to enable all staff members to communicate at a consistent level. Once analysis is initiated, several weeks of trial and error may be required, with guidance from an experienced consultant, before an efficient level of confidence is gained. Once this level of effectiveness is achieved, these staff members can effectively elaborate abstract work packages to consistent detail and to uncover gaps and weaknesses in the work products of their peers.

Transformation Across Stages

1. **Can developers use the method to represent the system under development at all stages?** Not effectively. The target of these methods is requirements specification and, especially in the case of *Hatley-Pirbhai* and *Harel*, system design as well. We feel that these are the only developmental stages for which these techniques are appropriate. In particular, we believe that they are insufficient for deriving and capturing detailed software design, in that the notations cannot depict the wide variety of constructs of the solution space necessary for *efficient* implementation. Using appropriate transformation techniques, they may be used effectively with other software design techniques, such as variants on structured design and object-oriented design.

2. **Are developers, using the method, supported by a set of transformation rules or guidelines that allow them to transform the representations at one stage to those of the next?** Yes, within the limitations of the preceding question. *Hatley-Pirbhai* provides transformation guidelines between their functional/behavioral model and structural model. *Ward-Mellor* provides transformation guidelines between their "essential model" and "implementation model"; these could be applied to *ESML* as well. *Harel* does not provide specific transformation guidelines between his functional/behavioral models and structural model, although an approach similar to that of *Hatley-Pirbhai* might be used. None of the methods provides extensive guidelines in transformation to software design. Transformation techniques described for Structured Analysis to Structured Design may be used with minor modification, as described by *Ward-Mellor*. Similarly, a number of object-oriented transformation techniques have been proposed recently (see [Bulman88] and [Seidowitz87] for examples).

    - **Do the rules prescribe a transformation process that is relatively automatic?** Available transformation techniques (see above) are relatively automatic only for the derivation of "first cut" designs. Further

restructuring and refinement requires appropriate expertise in design and domain-specific issues.

- **Do the entities and structures created at one stage of development remain visible and intact at the next stage?** Not always. While the "first cut" design may retain the bulk of structures in the requirements specification, successive design modifications may result in extensive repartitioning of process and data structures.

- **Does a representation continue to serve a useful purpose after the representation for the next stage is completed?** Not always. When integrated with a structural modeling technique, there is a greater likelihood that the system/software requirements specification will continue to be useful throughout subsequent development and maintenance, but only if traceability is maintained and is fairly direct. Barring this, the requirements specification may become nearly useless, or even detrimental, to subsequent development.

Large-Scale Problems

1. **Can developers use the method to partition the problem into a set of smaller problems with well defined interfaces and integrate the results?** For the most part, the methods scale-up well, with the exception of some of the behavioral notations (particularly conventional state-transition diagrams).

    - **Does the method provide hierarchical representations that allow developers to work at a detailed level and easily remember the overall context?** In general, yes. The behavioral notations of *Hatley-Pirbhai*, *ESML*, and *Ward-Mellor* for the most part are not hierarchical by themselves. When slaved to the functional view, they obtain sufficient hierarchical features for scale-up. Because of their emphasis on state-transition diagrams, *Ward-Mellor* tends to be more difficult to scale up due to state/transition explosion. Partitioning these diagrams to reduce the problem is difficult and awkward. By contrast, tabular formats (such as those used by *Hatley-Pirbhai* and *ESML*) can be extended fairly arbitrarily. For many scenarios, *Harel*'s hierarchical statecharts are the best solution of all. Ideally, one should be able to choose any mix of the available representations that best meets the needs of a given situation.

    - **Are the rules used for partitioning and decomposing one view consistent with the rules for other views? For example, if the method supports data flow and control flow, are they dealt with at compatible levels?** All of the methods are internally consistent in terms of decomposition rules.

    - **Does the method endorse naming conventions that allow multiple developers to integrate their results and avoid naming clashes?** All of the methods employ a flat name space, rather than attempting to incorporate a notion of scope. This can be problematic for large-scale

systems. A solution that can be employed by automated tools is a scope-specific name space.

User Interface

1. Does the method provide representations that allow the user to visualize the user interface?

   - **Do the representations adequately represent what the user will see, content and format of displays?** No; however, some of the available automated tools include a front-end tool that can be used for modeling the user interface.

   - **Do they allow representation of user inputs, content and format?** No.

   - **Do they allow the user to visualize a dialogue with the system, the ability of the user to modify the displays and the mechanics of the interactions?** No.

2. **Does the method promote the partitioning of user interface processing from other processing?** In general, no. *Hatley-Pirbhai*'s structural modeling notations do differentiate between user interface and other components, but this is from an architectural viewpoint rather than processing. It is not clear that this is sufficient.

Method Documentation

1. **Does the developer of the method provide sufficient documentation on the syntactic and semantic conventions of the method?** *Harel* ([Harel87], [Harel88a], [Harel88b], [i-Logix87], [i-Logix89a]) focuses heavily on syntactic and semantic rules for the behavioral notations, and to a much lesser degree for the functional and structural components. *Hatley-Pirbhai* [Hatley87] provides substantial documentation of the functional and behavioral aspects, and sufficient (though not extensive) coverage of the structural component. *Ward-Mellor* [Ward85] provides adequate discussion of the syntax and semantics of the functional and behavioral components of the method. *ESML* [Bruyn88] is brief, but, used in conjunction with *Ward-Mellor* documentation, is manageable.

2. **Does the developer of the method provide practical assistance in the application of the notations and techniques to actual problems?** The *Hatley-Pirbhai* text is superior in its treatment of practical application. Clearly, it is written by practitioners for practitioners, with a wealth of guidelines and rules-of-thumb that should be applicable to a wide audience. As a practical reference work, the *Ward-Mellor* texts are somewhat less satisfying in style, format, and content, although many useful insights are provided. The *Harel* documentation consists of a distributed group of papers, manuals, and publications which do not constitute a satisfactory reference work for the practitioner. Derivation guidelines are particularly sparse. The *ESML* paper

provides a discussion only of the graphical language and does not delve into guidelines or techniques for derivation or examination, or hints for the practitioner. For such information, *ESML* refers users to [Ward85]. Insofar as many of the notations and techniques of these methods may be intermixed, the documentation for each method may be applied to the other methods.

3. **Does the developer provide good examples that can be used to clarify methodological issues?** *Hatley-Pirbhai* provides a few small-scale examples that are of some limited value. Fragmented examples from actual systems (avionics) can be found throughout and are valuable. *Ward-Mellor* also provides a few small-scale examples of limited value, but lacks significant realistic examples found in the *Hatley-Pirbhai* text. *Harel* relies primarily on very limited examples and mainly for behavioral notations. A more complete example of the *Harel* approach may be found in [i-Logix89b]. The *ESML* paper provides very few examples.

Appendix D: Comments From Methodologists

This appendix provides brief critiques of this report from methodologists representing the *ESML*, *Harel, Hatley-Pirbhai*, and *Ward-Mellor* methods.

D.a. ESML

The following information was submitted by Paul Carpenter, a member of the ESML Working Group.

> The Extended Systems Modeling Language (ESML) is a new modeling language based on the Ward-Mellor and Hatley Structured Methods. ESML is a graphics-based language that supports system and software modeling techniques such as: functional decomposition, data-driven, event-driven, object-oriented, and architecture modeling. ESML focuses on the analysis, design, and specification activities of the development life cycle, and the ESML working group is currently writing formal definitions for extended data flow diagrams, control specifications, information models, and architecture models.
>
> The ESML working group was formed after the 12th Structured Methods Conference in 1987, by a number of conference attendees interested in modeling language standards. The goal of the ESML working group is to develop extensions and modeling language standards for Structured Methods. The ESML working group consists of representatives from industry, CASE vendors, and independent consultants, who meet four times a year. ESML extended data flow diagrams and control specifications have been published in two journals and have been discussed at a number of CASE conferences. The mailing list of persons interested in ESML exceeds 300, and one open meeting has been held. Another open meeting will be scheduled in 1990.
>
> Current members of the ESML working group are:

Bill Bruyn	Index Technology Corporation
Paul Carpenter	Honeywell, Inc.
Derek Hatley	Smiths Industry
Randy Jensen	Hughes Aircraft
Paul Jorgensen	Research & Technology Institute of W. Michigan
Ted Liu	Athena Systems
Jess Thompson	Nastec
Steve Tockey	Boeing Commercial Airplane Company
Steven Weiss	Wayland Systems

Paul B. Carpenter Staff Engineer, Software Technology
Honeywell Inc.
Air Transport Systems Division
Sperry Commercial Flight Systems Group
21111 North 19th Avenue
Phoenix, Arizona 85027

D.b. Harel

The following comments were submitted by David Harel.

> We appreciate the opportunity to address some points that fall outside the scope of the report but may be of interest to its readers.
>
> The Harel method and its supporting toolset, Statemate, were developed as one; the method's strength is best appreciated in this context. As implemented in Statemate, the visual languages of the Harel method are used to create an executable model. Several of the report's categories for comparison take on new meaning when considered in this light.
>
> For example, examining a representation is quite a different thing when it can be done with automated support. Statemate provides step-by-step interactive execution, programmable batch execution, and exhaustive testing. This allows the user to perform extensive analysis to determine whether the system as specified will behave as intended.
>
> As another example, the report suggests that all the methods surveyed are weak in their ability to serve as a communication vehicle between developers and non-technical customers. However, the Statemate model can be automatically translated into prototype code, which can be used to drive a graphical representation of the system, such as a control panel. The panel can be manipulated by users or customers, providing an excellent means for communication.
>
> Similar points can be made about the report's mentions of verification, traceability, testing, and the resolution of ambiguous system descriptions.
>
> We would also like to emphasize a point made in the report that may not have been clear to all readers. Because all of the methods described belong to the family of real-time structured analysis, the derivation techniques developed for one method apply equally well to the others. We encourage users of our method to adopt the excellent guidelines developed and described by Ward-Mellor and Hatley-Pirbhai.

Professor David Harel
Department of Applied Mathematics and Computer Science
The Weizmann Institute of Science
Rehovot, Israel 76100

D.c. Hatley and Pirbhai

Derek Hatley and Imtiaz Pirbhai submitted the following comments:

> We appreciate the SEI's efforts to bring some objectivity to this hotly debated field, and especially their approach of biting off just a digestible chunk, rather than trying to chew on the whole methods arena. We feel that their coverage of our particular methods is fair as far as it goes, but only covers in depth those areas that are common with the other methods. Below we summarize our underlying methods philosophy, how our methods extend well beyond the above common areas, and the current CASE tool situation.
>
> Our background in practical systems development tells us that practitioners are not interested in methods for their own sake; they are interested in them only as tools to get a job done. A method whose use approaches the complexity of the system it seeks to model is not doing us any favors. Furthermore, no one can foresee all the needs of the myriad possible methods applications, so it is inappropriate to provide special notations for any particular subset of those needs. And finally, we build *systems*, not just software.
>
> Accordingly, we devised the requirements and architecture methods as simple and flexible means for supporting total systems development, rather than just software. The methods produce comprehensive systems models that are independent of any particular hardware or software technology or language; they simply provide a framework which can be adapted to the needs of the system at hand. They include a set of heuristics for gathering information and building the models, with a minimum number of symbols and constructs; their only constraints are those necessary to ensure consistency and traceability.
>
> We are pleased that our methods have been widely accepted, and that several CASE tool developers have chosen to support the requirements method, but we are sorry to report that none of them so far supports it completely or correctly. Our advice to CASE tool users is to evaluate a tool carefully against the definitions of the methods in which they are interested--the claims of the developer may not be borne out in practice. It is unfortunate too that no CASE tool yet supports the architecture method, even though many systems developers are using it. This method provides a vehicle for tools to truly work at the system level over the full systems development lifecycle, and potentially to integrate with Computer Aided Engineering (CAE) tools, a combination which we believe will eventually be essential.

Derek J. Hatley
Smiths Industries
Aerospace and Defense Systems
SLI Avionics Systems Corp.
4141 Eastern, SE
Grand Rapids, MI 49518-8727

Imtiaz Pirbhai
Systems Methods, Inc.
900 Summit Avenue East
#302
Seattle, WA 98102

D.d. Ward and Mellor

The following are comments submitted by Paul Ward and Stephen Mellor, respectively.

> I agree with nearly all your recommendations. However, I disagree with your inclusion of notational simplicity, but not well-defined semantics, in your set of general discriminating factors in the Executive Summary. Well-defined semantics is fundamental; its basis is not merely automated execution, but the availability to express unambiguously the nature of the proposed system. Simplicity is a valid figure of merit only from a solid semantic baseline. Calling a method "simple," when the method lacks the ability to express important distinctions, is not a compliment. Although I work with all four notations in my teaching/consulting practice, I prefer Harel and ESML because they combine modeling power and well-defined semantics.

Dr. Paul Ward Software Development Concepts
424 West End Avenue
Suite 11E
New York, NY 10024

> The report omits any discussion of the techniques used to derive an analysis; for example, functional decomposition vs. event-response. Conclusions are drawn without clear explication of the reasoning used. The report appears to be already out of date. The most popular method, according to the survey in the report, is Object-Oriented Analysis. OOA, as described in *An Object-Oriented Approach to Domain Analysis*, Software Engineering Notes, ACM Press, July 1989 (Shlaer and Mellor of Project Technology, Inc.), uses a simple, rigorously defined notation coupled with a strong formalism providing effective guidelines for systems analysis. Since the report concludes that there are few significant differences between the four methods surveyed, and OOA has become very popular over a short period of time, I would urge readers of the report to investigate OOA.

Stephen J. Mellor Project Technology, Inc.
2560 Ninth Street
Suite 214
Berkley, CA 94710

Appendix E: Glossary

animation: The continuous display of the current status of a simulation by visually highlighting the active elements in the specification.

CASE: Computer aided (or assisted) software (or system) engineering.

dynamic analysis: The computerized testing of the behavior of a system under development based on its specification. This analysis is done by searching for specific behavioral properties of the proposed system.

ESML: Extended Systems Modeling Language, a specification notation described in [Bruyn88].

executable specification: A description of the required behavior for a system under development that can be syntactically and semantically checked and run by a computer.

Harel: The method described in [Harel88a], [Harel88b], [i-Logix89a], and elsewhere, often called *statemate* after the name of the supporting toolset, or statecharts, after the name of the methods behavioral languages.

Hatley-Pirbhai: The method described in [Hatley87].

method: A systematic approach to solving a given problem. A method consists of some combination of notations and techniques. See the discussion in Section 2.5 of this report.

methodology: A collection of methods that together can be used to solve a problem (alternatively, the study of methods) [Webster85].

notation: Some combination of graphical, textual, or tabular representations used to describe the solution to the problem at hand.

process: A set of actions, tasks, and procedures that when performed or executed obtain a specific goal or objective.

real-time: Pertaining to the processing of data by a computer in connection with another process outside the computer according to time requirements imposed by the outside process [IEEE83].

RFP: Request for Proposal.

SEM: System Engineering Methodology; the method described in [Wallace87].

semi-formal: A notation or technique that constrains the engineer such that the progression of activities and the resulting products follow orderly and planned patterns; semi-formal methods may include components that are formal and components that are informal.

simulation: The process of a computer executing the specification of a system under development. This simulation shows (via animation or reports) how the system under development would operate if built according to the specification.

software design: The creative and mechanical process by which the solution to a specified software problem is attained. A software design is usually manifested in an allocation of requirements to interacting software modules and the definition of their interfaces. Compare to *system design*.

software engineering: The systematic approach to the development, operation, maintenance, and retirement of software [IEEE83].

software engineering process: The total set of software engineering activities needed to transform a user's requirements into software. The process of applying engineering principles to produce software.

software engineering process group (SEPG): A group of specialists concerned with the software engineering process used by an organization. The SEPG defines and documents the software process, establishes and defines process metrics, supports project data gathering, assists projects in analyzing data, and advises management on areas requiring further attention.

software process: A process performed to produce, support, maintain, and enhance software. Examples of a software process are a software development process, a software maintenance process, etc.

software requirements specification: The process and product of determining and defining the requirements for a software component of a system.

specification: The process and product of determining and defining all requirements for a given component in a system under development. In the context of a software component, the complete specification might include system requirements specification, system design, and software requirements specification.

state of the practice: The commonly accepted level of use of applied scientific or engineering knowledge at a particular time.

system design: The process by which a solution to a specified system problem is attained. A system design usually is manifested in allocations of requirements to software, hardware, and other components, and the logical and physical interfaces between those components are defined. Compare to *software design*.

system requirements specification: The process and product of determining and defining all requirements for a system under development at the most abstract level.

tool: A product, usually automated, that supports some engineering activity (such as a method).

toolset: A group of tools that support several engineering activities (such as a set of methods or a methodology).

Ward-Mellor: The method described in [Ward85].

Bibliography

[Alford77] Alford, M.
A Requirements Engineering Methodology for Real-Time Processing Requirements.
IEEE Transactions on Software Engineering SE-3(1):pp. 60-69, 1977.

[Bailin89] Bailin, S.
An Object-Oriented Requirements Specification.
Communications of the ACM 32(5):pp. 608-623, May, 1989.

[Bergland81] Bergland, G.D.
A Guided Tour of Program Design Methodologies.
Computer 14(10):pp. 13-37, October, 1981.

[Blank83] Blank, J. and Krijger, M.
Software Engineering: Methods and Techniques.
John Wiley and Sons, New York, 1983.

[Booch83b] Freeman, P., and Wasserman, A. (editors).
Object-Oriented Design; In Tutorial: Software Design Techniques.
IEEE Computer Society Press, Washington, DC, 1983.

[Brooks87] Brooks, F. P.
No Silver Bullet, Essence and Accidents of Software Engineering.
IEEE Computer 20(4), April, 1987.

[Bruyn88] Bruyn, W., Jenson, R., Keskar, D., Ward, P.
ESML: An Extended Systems Modeling Language.
ACM Software Engineering Notes 13(1):pp. 58-67, January, 1988.

[Buhr84] Buhr, R.J.A.
System Design with Ada.
Prentice-Hall, Inc., Englewood, Cliffs, NJ, 1984.

[Bulman88] Bulman, D.
Model-Based Object-Oriented Design for Ada.
Course Notes.
Pragmatics, Inc., Waikoloa, HA.

[CECOM89] Gerichten, L., et al.
Software Methodology Catalog, Second Edition.
Technical Report C01-091JB-0001-01, Teledyne Brown Engineering, Eatontown, New Jersey, March, 1989.
Prepared for U.S. Army Communications - Electronics Command (CECOM).

[Chen76] Chen, P.P.
The Entity-Relationship Model - Toward a Unified View of Data.
ACM Transactions on Database Systems 1(1):pp. 9-36, March, 1976.

[Cherry87] Cherry, G.
Introduction to PAL and Pamela II: Process Abstraction Language and Process Abstraction Method for Embedded Large Applications
P.O. Box 2429, Reston, VA 22090, 1987.

[Davis88] Davis, A.
A Comparison of Techniques for the Specification of External System Behavior.
Communications of the ACM 31(9):pp. 1098 - 1115, September, 1988.

[DeMarco78] DeMarco, T.
Structured Analysis and System Specification.
Yourdon, Inc., New York, 1978.

[DOD82] Ada Joint Program Office.
Ada Methodologies: Concepts and Requirements.
Technical Report, Department of Defense, November, 1982.

[DSB87] Brooks, F., et al.
Report of the Defense Science Board Task Force on Military Software.
Technical Report, DSB, Office of the Under Secretary of Defense for Acquisition, Washington, D.C., September, 1987.

[Fairley85] Fairley, R.
Software Engineering Concepts.
McGraw Hill, New York, 1985.

[Firth87] Firth, R., Mosley, V., Pethia, R., Roberts, L., Wood, W.
A Classification Scheme for Software Development Methods.
Technical Report CMU/SEI-87-TR-41, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November, 1987.

[Floyd86] Floyd, C.
A Comparative Evaluation of System Development Methods.
In Olle, T.W., Sol, H. G., and Verriijn-Stuart, A. A. (editors), *Information Systems Design Methodologies: Improving the Practice*. North-Holland, 1986.

[Gane79] Gane, C., and Sarson, T.
Structured Systems Analysis: Tools and Techniques.
Prentice-Hall, 1979.

[Griffiths78] Griffiths, S. N.
Design Methodologies - A Comparison.
Structured Analysis and Design II:pp. 133-166, 1978.

[Harel87] Harel, D.
Statecharts: A Visual Formalism for Complex Systems.
Science of Computer Programming 8:pp. 231-274, 1987.

[Harel88a] Harel, D.
On Visual Formalisms.
Communications of the ACM 31(5):pp. 514-530, May, 1988.

[Harel88b] Harel, D., Lachover, H., Naamad, N., Pnueli, A., Politi, M., Sherman, R., and Shtul-Trauring, A.
Statemate: A Working Environment for the Development of Complex Reactive Systems.
In *Proceedings of the 10th IEEE International Conference on Software Engineering*, pages 396-406. IEEE Computer Society, April, 1988.

[Harel89] Harel, D. and Rolph, S.
Modeling and Analyzing Complex Reactive Systems: The Statemate Approach
i-Logix, Inc., Burlington, MA, 1989.
Presented at AIAA Computers in Aerospace Conference, Monterey, CA.

[Hatley87] Hatley, D., and Pirbhai, I.
Strategies for Real-Time System Specification.
Dorset House, New York, 1987.

[Hatley88] Hatley, D.
CASE Tool Evaluation: A Real-Time Example.
In *Proceedings from CASES/ADE 88*, pages 67-87. Digital Consulting, Inc., Washington, DC, July, 1988.

[Heninger80] Heninger, K., et al.
Specifying Software Requirements for Complex Systems: New Techniques and Their Application.
IEEE Transactions on Software Engineering , June, 1980.

[Hoza89] Hoza, B., Smith, M., TocKey, S.
An Introduction to Object-Oriented Analysis.
In *Proceedings of STA5*, pages 312-330. Structured Techniques Association, Chicago, IL, May, 1989.

[Humphrey87] Humphrey, W.
Characterizing the Software Process: A Maturity Framework.
Technical Report CMU/SEI-87-TR-11, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1987.

[i-Logix87] i-Logix, Inc.
The Languages of Statemate
i-Logix, Inc., 22 Third Ave., Burlington, MA, 1987.

[i-Logix89a] i-Logix, Inc.
The Statemate Approach to Complex Systems
i-Logix, Inc., 22 Third Ave., Burlington, MA, 1989.

[i-Logix89b] i-Logix, Inc.
Statemate by Example.
i-Logix, Inc., Burlington, MA, 1989.

[IBM88] Locke, C., Vogel, D., Lucas, L.
Generic Avionics Software Specification.
Draft specification for Naval Weapons Center, China Lake, CA.
IBM systems Integration Division, Owego, NY.

[IDA87] Fife, D., et al.
Evaluation of Computer-Aided System Design Tools for SDI Battle Management/C3 Architecture Development.
Technical Report Draft Report 871204, Institute for Defense Analysis, December, 1987.

[IEEE83] The Institute of Electrical and Electronics Engineers, Inc.
IEEE Standard Glossary of Software Engineering Terminology.
Technical Report ANSI/IEEE Std. 729-1983, IEEE, February, 1983.

[Jackson83] Jackson, M.
System Design.
Prentice-Hall, Englewood Cliffs, NJ, 1983.

[Jenson88] Jenson, R.
Effective Use of CASE in an Organization.
In *Proceedings from CASES/ADE 88*, pages 89-113. Digital Consulting Inc., Washington, DC, July, 1988.

[Kelley87] Kelley, J.
A Comparison of Four Design Methods for Real-Time Systems.
Research paper supported by a NASA/ASEE fellowship at JPL, CA. 1987

[Lefkovitz82] Lefkovitz, D., and Hill, H.
The Applicability of Software Development Methodologies to Naval Embedded Computer Systems.
Technical Report, University of Pennsylvania , November, 1982.

[Mannino87] Mannino, P.
A Presentation and Comparison of Four Information Systems Development Methodologies.
ACM Software Engineering Notes 12(2):pp. 26-29, April, 1987.

[Peters77] Peters, L. and Tripp, L.
Comparing Software Design Methodologies.
Datamation 23(11):pp. 89-94, November, 1977.

[Ross77] Ross, D. T.
Structured Analysis (SA): A Language for Communicating Ideas.
IEEE Transactions Software Engineering SE-3:pp. 16-34, January, 1977.

[Shlaer88] Shlaer, S., and Mellor, S.
Modeling the World in Data.
Prentice-Hall, New York, 1988.

[Seidowitz87] Seidowitz, E., and Stark, M.
Towards a General Object-Oriented Software Development Methodology.
Ada Letters 7(4):pp. 64-67, July/August, 1987.

[SPS88] Comer, E., Donaldson, C., and Dyson, P.
Computer-Aided Systems/Software Engineerin (CASE) Evaluation for Time-Critical Applications.
Technical Report, Software Productivity Solutions, under Contract N62269-86-C-0415, Melbourne, FL, April, 1988.

[STARTS87] U.K. Department of Trade and Industry.
The STARTS Guide.
NCC Publications, National Computing Centre, Oxford Road, Manchester, UK, 1987.
Prepared by the industry with the support of the DTI, NEDO and NCCC.

[Wallace87] Wallace, R., Stockenberg, J., Charette, R.
A Unified Methodology for Developing Systems.
McGraw-Hill, New York, 1987.

[Ward85] Ward, P., and Mellor, S.
Structured Development for Real-Time Systems.
Yourdon Press, Prentice Hall, New York, 1985.

[Ward86] Ward, P.
The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing.
IEEE Transactions on Software Engineering 12(2):pp. 128-210, February, 1986.

[Ward89a] Ward, P.
Embedded Behavior Pattern Languages: A Contribution to a Taxonomy of CASE Languages.
The Journal of Systems and Software 9(2):pp. 109-128, February, 1989.

[Ward89b] Ward, P.
How to Integrate Object Orientation with Structured Analysis and Design.
IEEE Software :pp. 74-82, March, 1989.

[Webster85] Dictionary.
Webster's Ninth New Collegiate Dictionary.
Merriam-Webster, Inc., Springfield, MA, 1985.

[White87] White, S.
A Pragmatic Formal Method for Computer System Definition.
UMI Dissertation Information Service, Ann Arbor, MI, 1987.

[Wood88] Wood, W., Pethia, R., Gold, L., Firth, R.
A Guide to the Assessment of Software Development Methods.
Technical Report CMU/SEI-88-TR-8, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, April, 1988.

Table of Contents

List of Figures