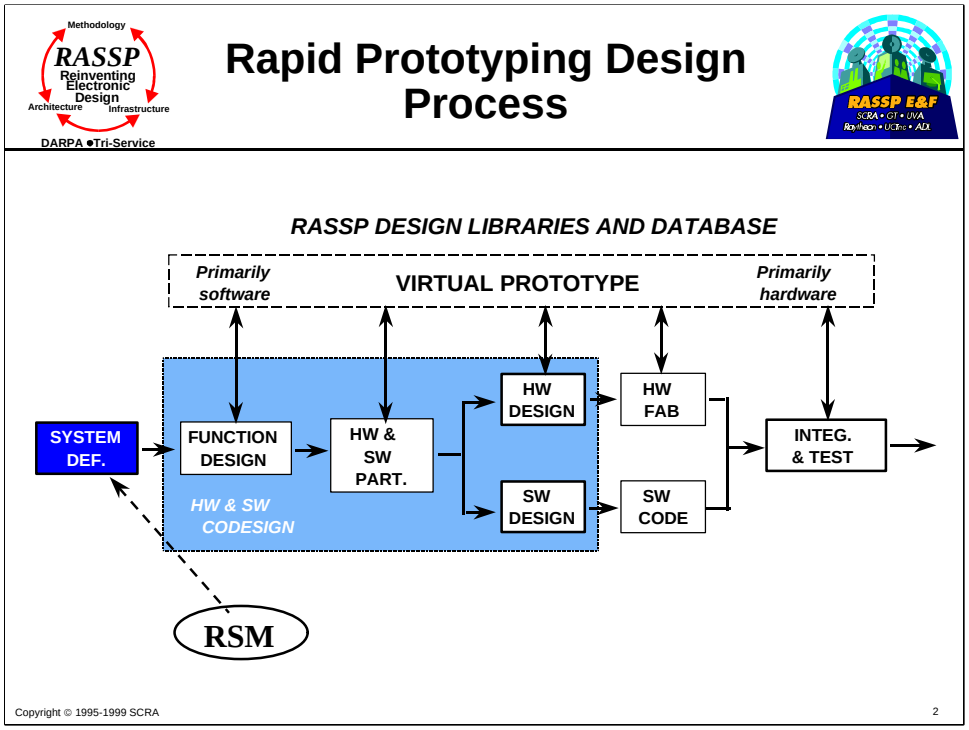
Requirements and Specification Modeling (RSM)

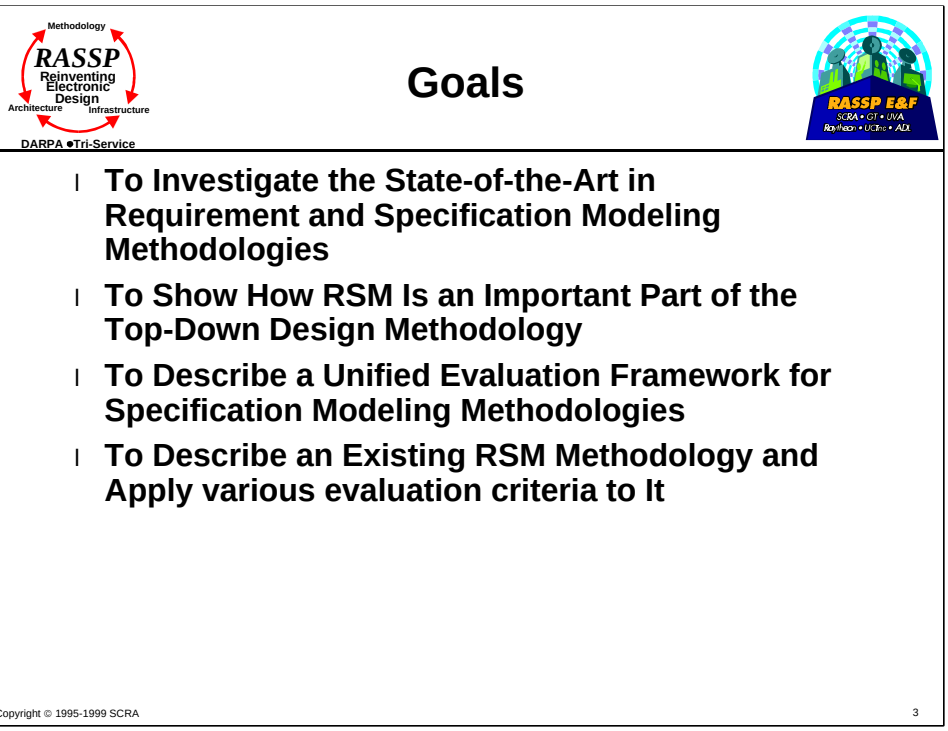
RASSP Education & Facilitation Program

Module 30

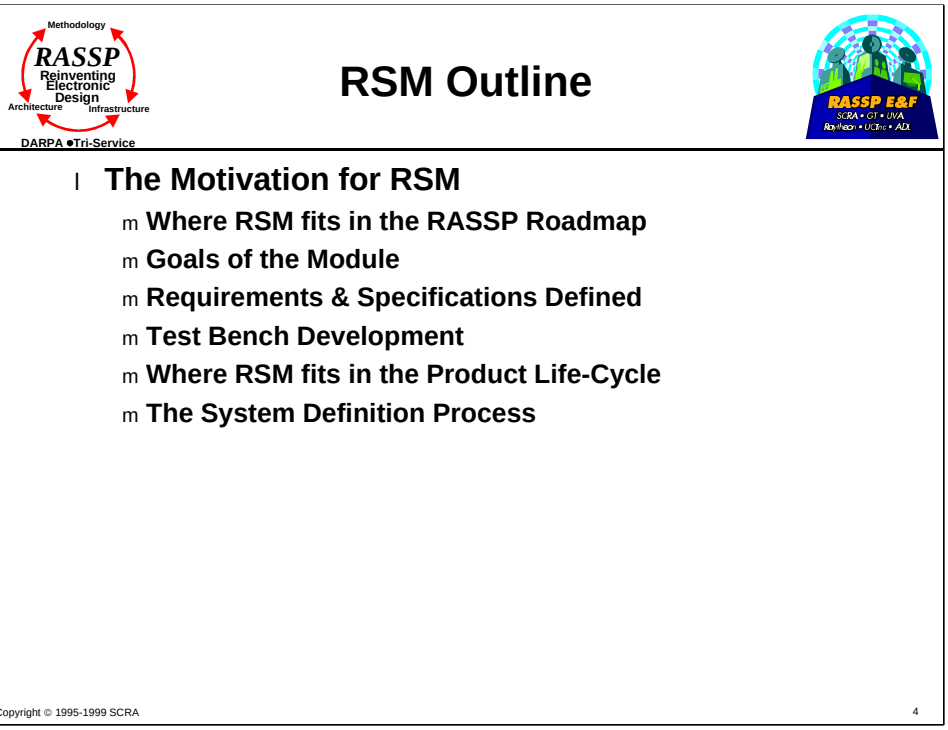
Version 3.00



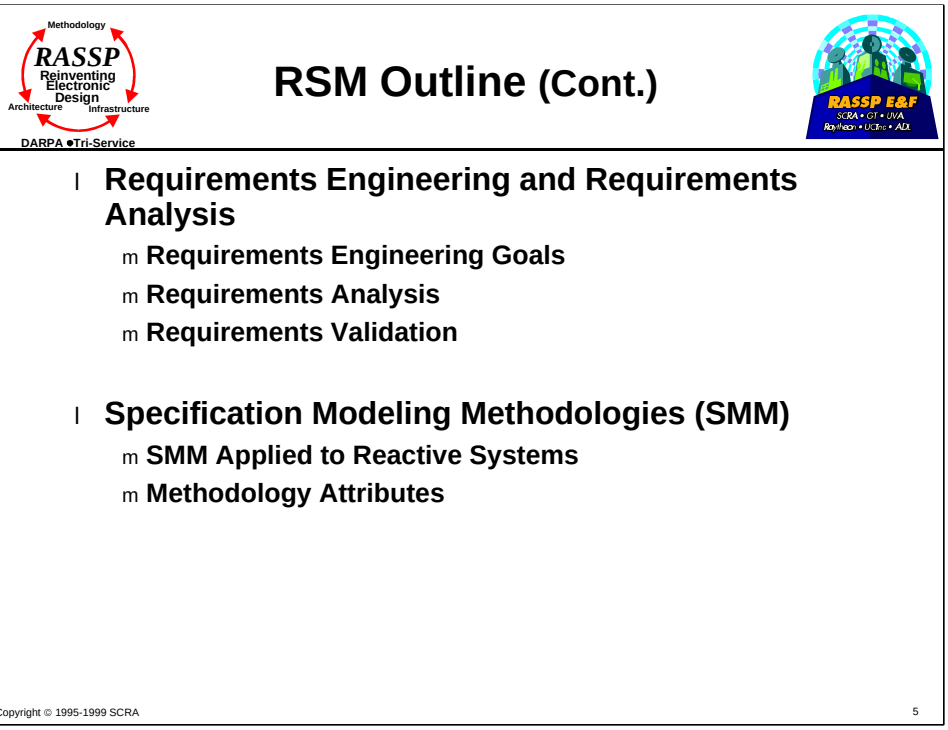
The Rapid Prototyping Design Process is applicable to all modules in the E&F program. This slide indicates where in the process Requirements and Specification Modeling (RSM) fits. Note that the dark blue region, "System Definition," is the primary home of RSM. However, the light blue area, Hardware/Software Codesign," which is the home of other modules in the E&F series, also has a strong affinity to RSM. Conformance between the light blue area and the RSM description is what must be tested continually as the design proceeds.



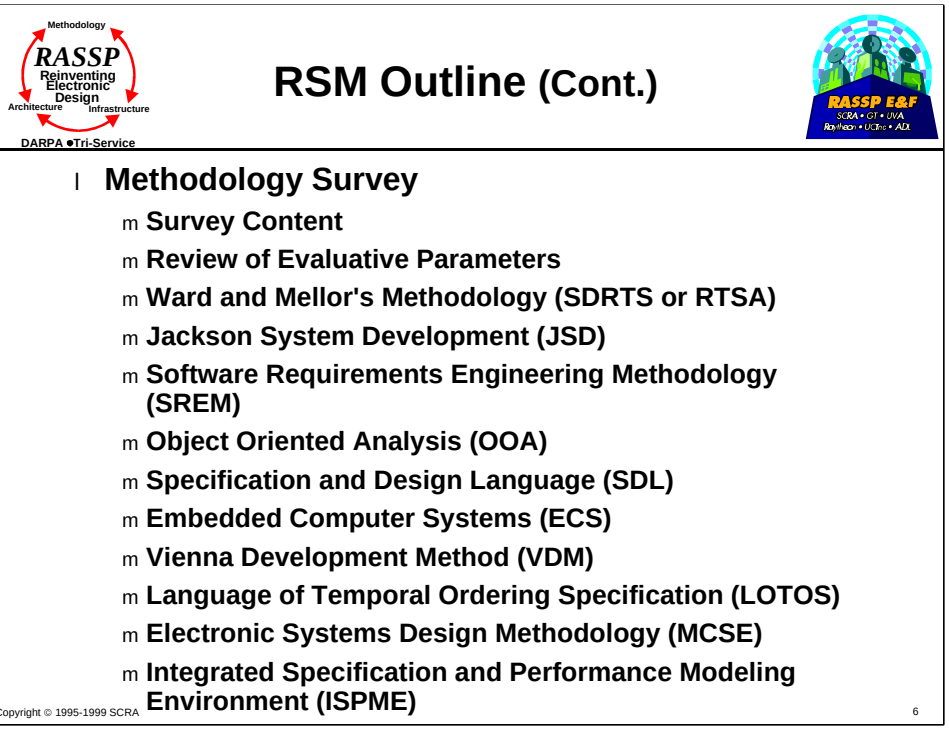
There are two major themes to this module: 1) RSM evaluation criteria and its application, and 2) RSM in industry and academia.



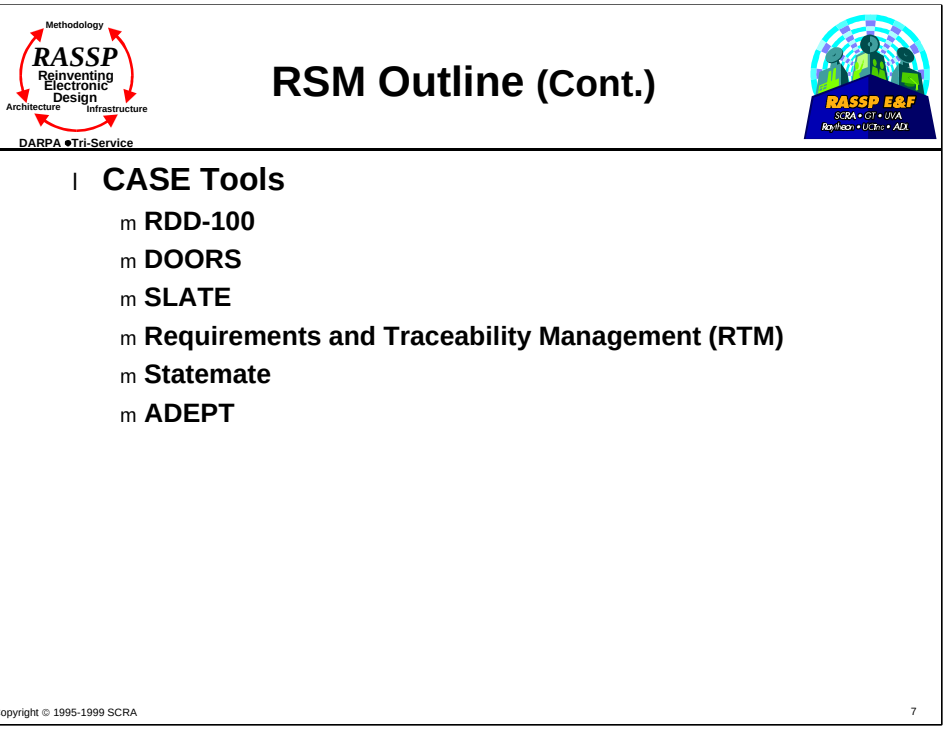
This module explains the role of RSM in the RASSP roadmap. Also of interest is another program, CEENS, that has contributed results to the development of executable requirements and specification using a representation called SimSpec that will also be used to briefly illustrate its role in capturing requirements and specifications.



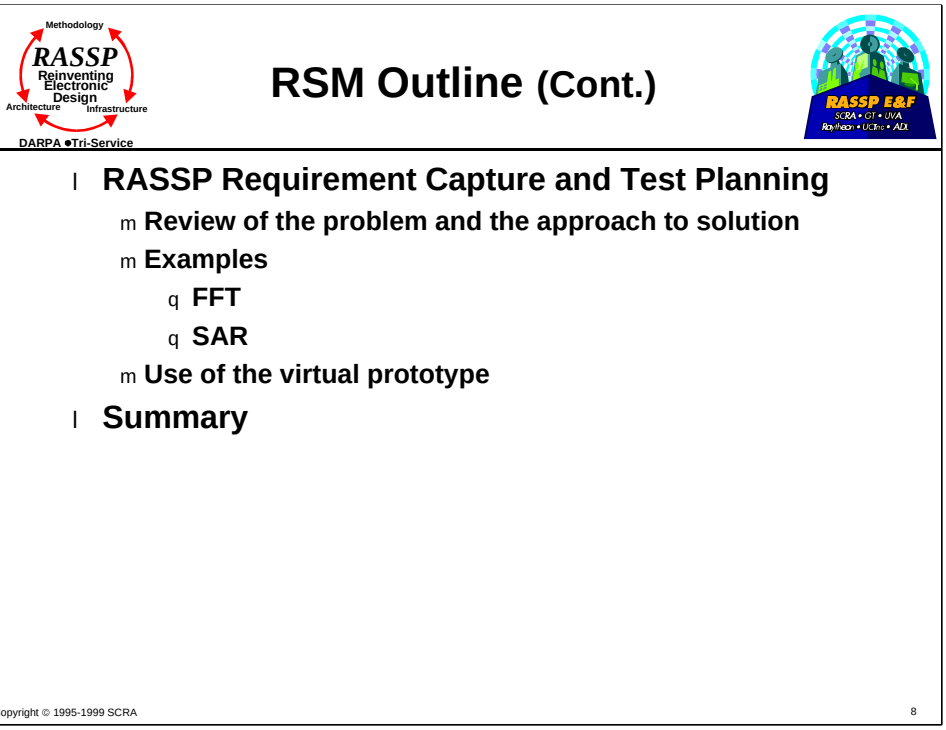
Requirements and Specifications each has a specific meaning. Requirements are usually obtained from the customer with respect to what a system should do. Specifications, on the other hand, describe how a system would implement the required functionality. More precise descriptions will be provided in the slides that follow.



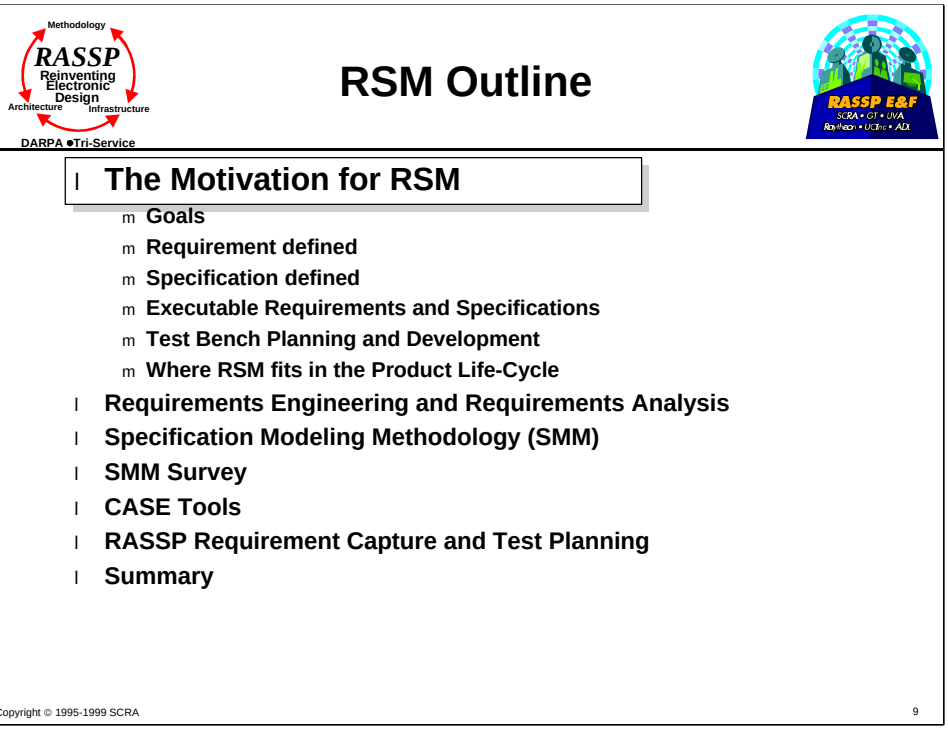
Considerable amount of recent literature exists in the area of system specification and modeling, and we attempt to survey a wide variety of proposed methodologies, and include a dozen or so prominent proposals as part of this module.



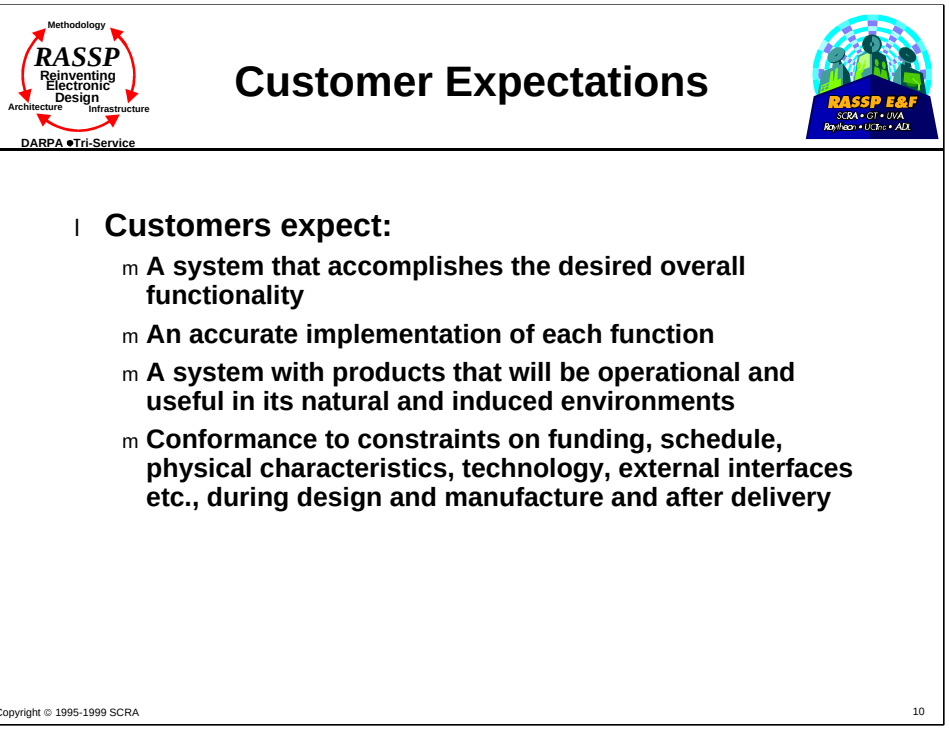
Tools that support the RSM methodology at various levels of abstraction are described in this module.



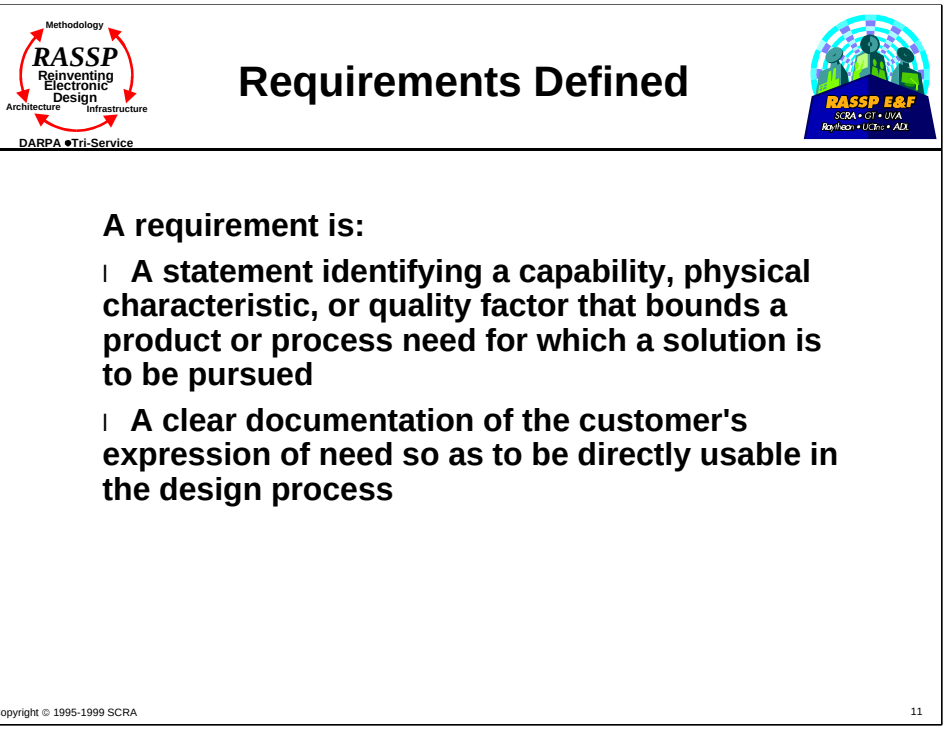
We describe RASSP contributions to the RSM effort through a series of detailed examples that illustrate the methodology proposed as part of that effort.



In this section of the module we describe the RSM process and define a number of terms and the methodology followed.



These define and quantify the customer expectations for the system.



Other definitions of requirements:

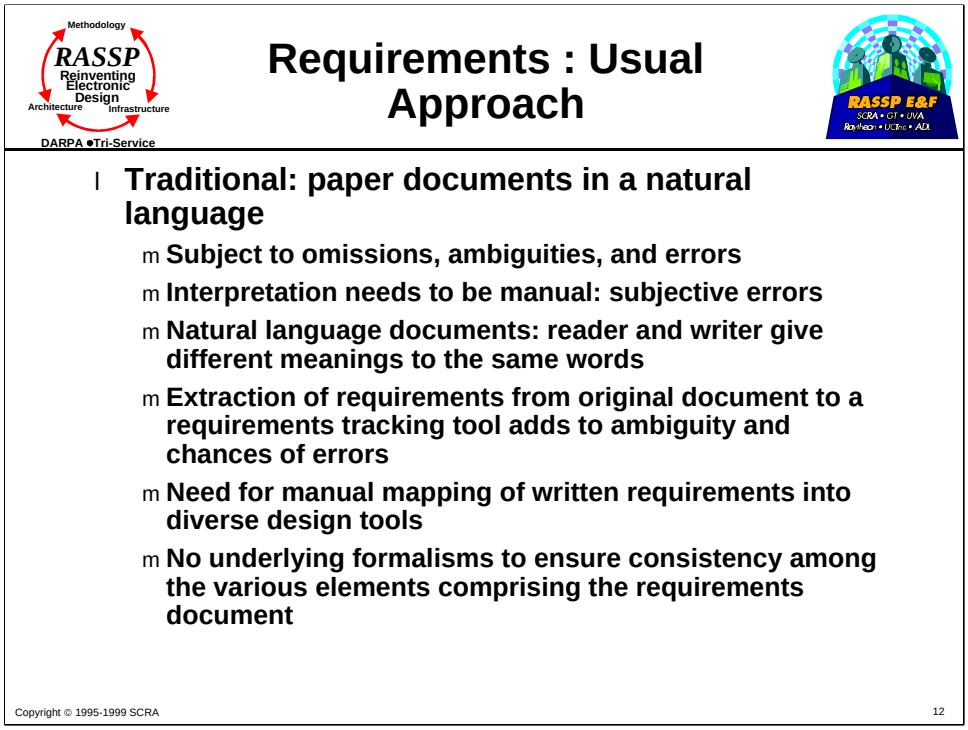
" An identification of a characteristic, physical or functional, that defines the need of a process or a product for which a solution will be attempted. It describes the nature of a request as an expression of a need."

" It is a condition or a capability that is:

-needed by a user to solve a problem or achieve an object.

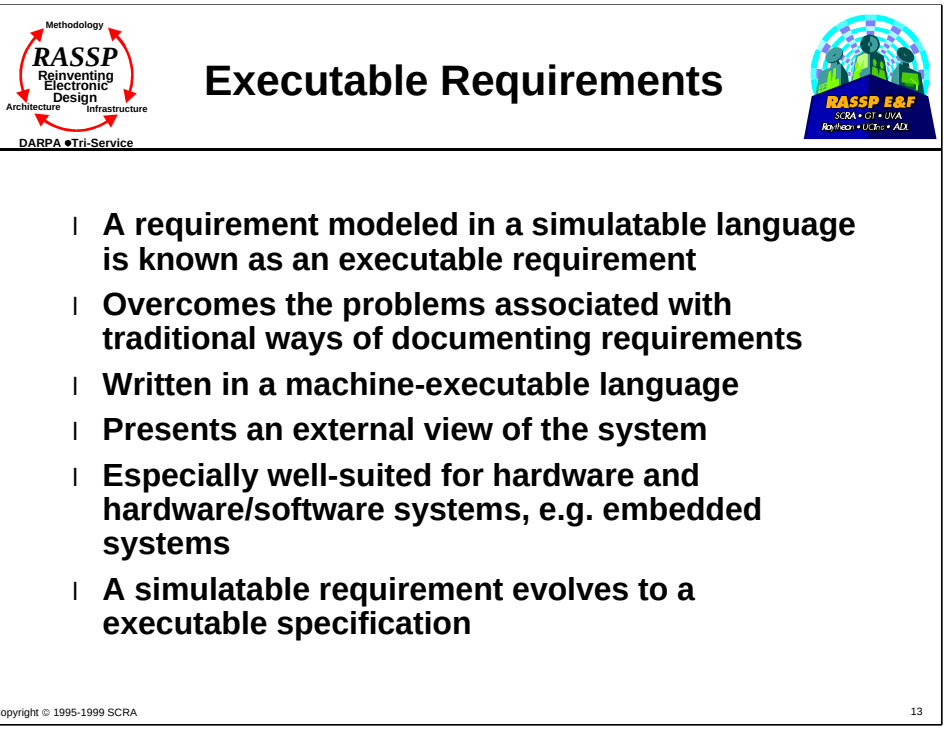
-required to be met or possessed by a system or system component to satisfy a contract, standard specification or other formally imposed documents.

-a documented representation of a condition or capability as in the above two."



Traditionally, the detailed form, function, cost and features desired for an electronic system are established in a set of requirements documents. Misinterpretation, omissions, and errors in these documents are often significant factors in slowing development of signal processing systems. A requirement which is written in a formally defined computer executable, rather than a natural, language provides an unambiguous description which can be tested for errors.Requirements for an embedded signal processor are conveyed in the form of paper documents which are subject to omissions, ambiguities, and errors on the part of the requirement generators (authors) and recipients (readers). In many instances, the written requirements are supplemented with supporting analysis and simulations, but this aggregate of requirements documentation must still be manually interpreted and judiciously applied to evaluate the adequacy of candidate hardware and software designs.During the course of design, it may be necessary to map written requirements into several different tools to assist in requirements tracking and allocation, performance analysis, simulation, and test development. Each mapping of requirements into a tool adds to the total design cost, and introduces opportunities for miscommunication and error. Executable requirements, when used in conjunction with a compatible language-based design methodology, reduce the need for manual mapping of written requirements into diverse design tools.

[Lincoln3]



An executable requirement must be written in a language which can be executed on a computer in order to test its syntactic correctness. It should present an external view of the desired system which, for an embedded signal processor, should include the following:

1. The desired signal transformation in each mode of operation.

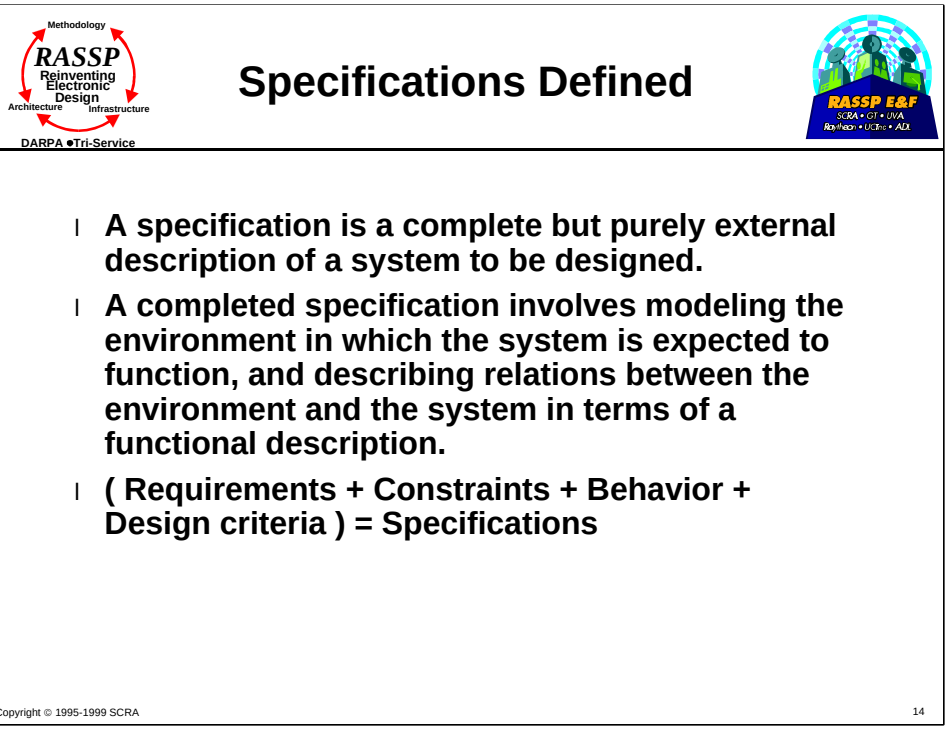
2. The desired response to commands and other control inputs.

3. Protocol and timing at the data and control ports.

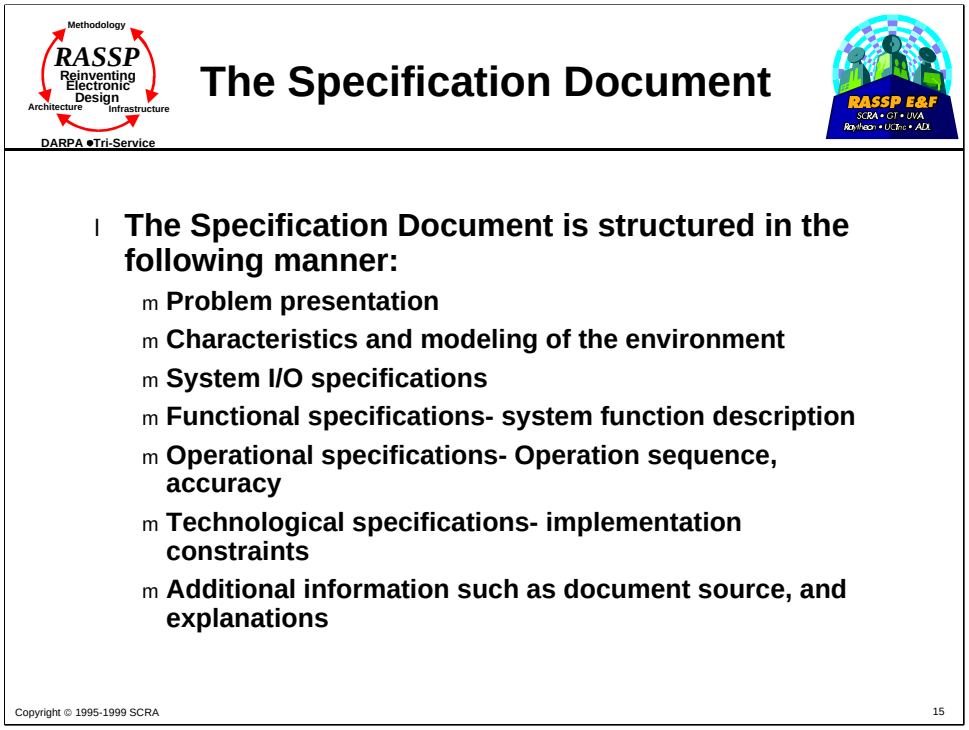
At some future time, an executable requirement may include other facets of the requirement such as physical and environmental constraints, testability goals, connector part numbers, etc. Executable requirements with this broad a scope are not feasible at this time, given the scope of available formal languages.

Many of the latter requirements, e.g., constraints may be captured through new enhancements to VHDL such as those proposed by the VSPEC effort in the RASSP Technology base program (University of Cincinnati).

[Lincoln3]



A specification is usually derived from the requirements, but includes additional detailed information on its function, timing, and design related behavior.



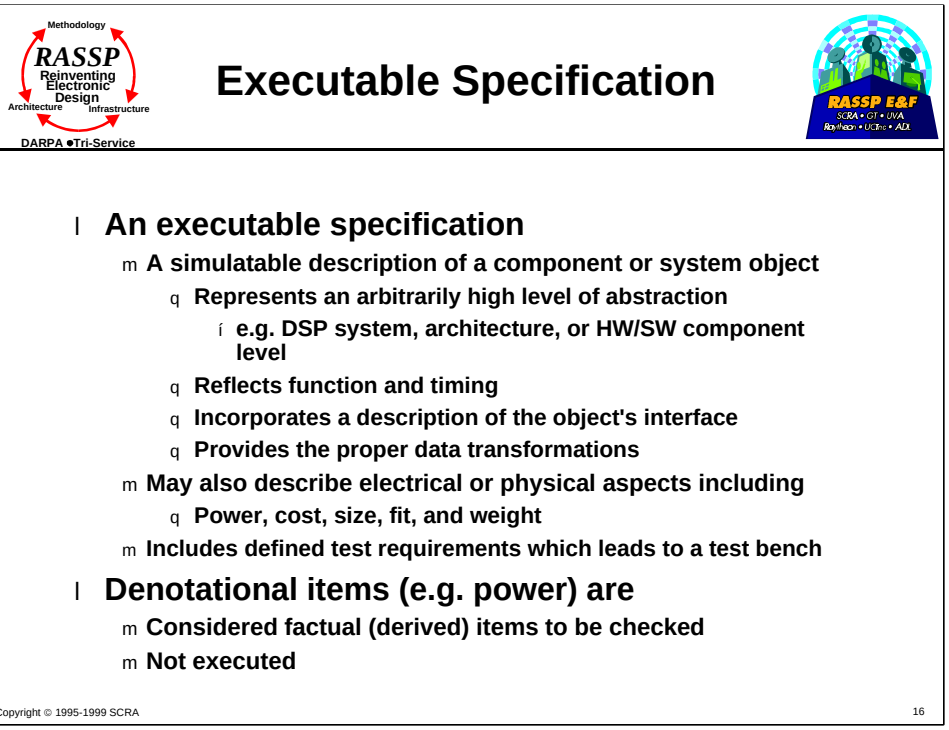
The cost of obtaining the (final) specification document can be reduced by using a suitable method and by specific analysis. The structure of the specification document must be in such a way that it addresses the following issues:

What questions should the document answer?

- Who are the readers and how will they use the document?
- What is the knowledge required to understand the document?

The document structure is a result of consideration of the above points.

The plan shows that the specifier gradually builds up the document based on his analyzing the environment, taking the output of the requirements phase as a chief input.

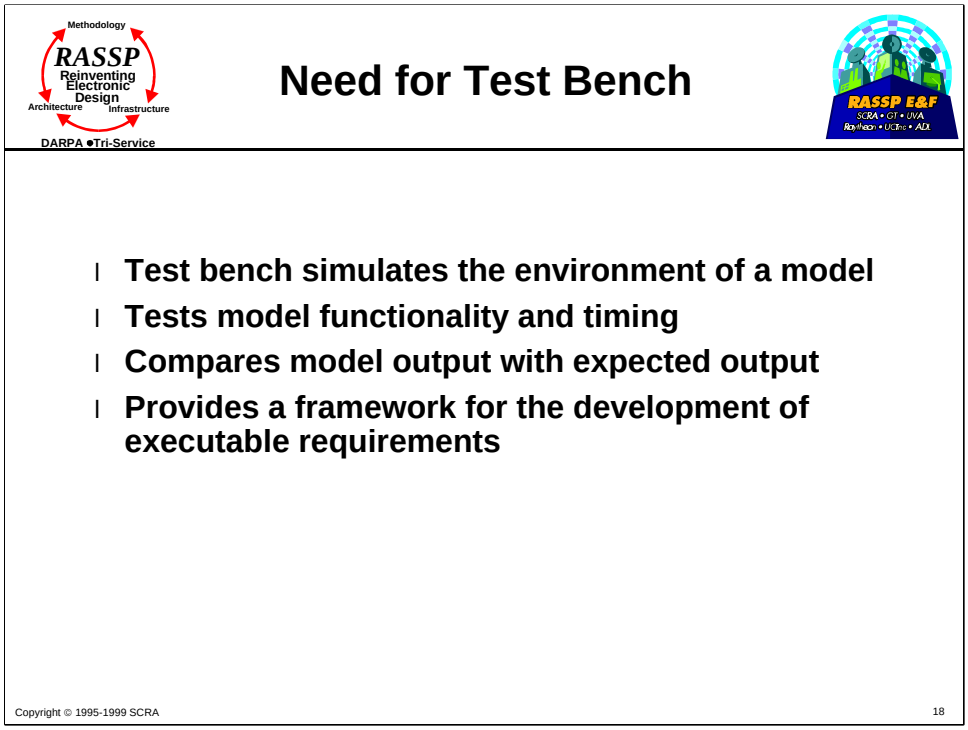


An executable specification is a behavioral description of a component or system object that reflects the particular function and timing of the intended design as seen from the object's interface when executed in a computer simulation. The executable specification may describe the electrical, behavioral, or physical aspects including the power, cost, size, fit, and weight of the intended design. Denotational items such as power are normally considered factual (derived) items to be checked but not executed. Executable specifications describe the behavior or function of an object at the highest level of abstraction that still provides the proper data transformations (correct data in yields correct data out; DEFINED bad data in has the SPECIFIED output results). Executable specifications may describe an object at an arbitrary abstraction level such as the DSP system, architecture, or hardware / software component level. In contrast to a virtual prototype, the executable specification need not describe a system's internal structure.

An executable specification is an electronic specification of the requirements for an electronic product. This electronic specification includes a formal requirements model, a simulatable model of the required product interfaces, function and timing as seen from the product's interface (includes hardware function and software function - which may not yet be defined/partitioned), simulatable interface models as required and the formal test benches to be used to assess the compliance of any implementation with this requirement specification. It also includes the captured product requirements which led to the requirements model and the defined test requirements which led to the test bench.

The Role of the Executable Requirement/Specification

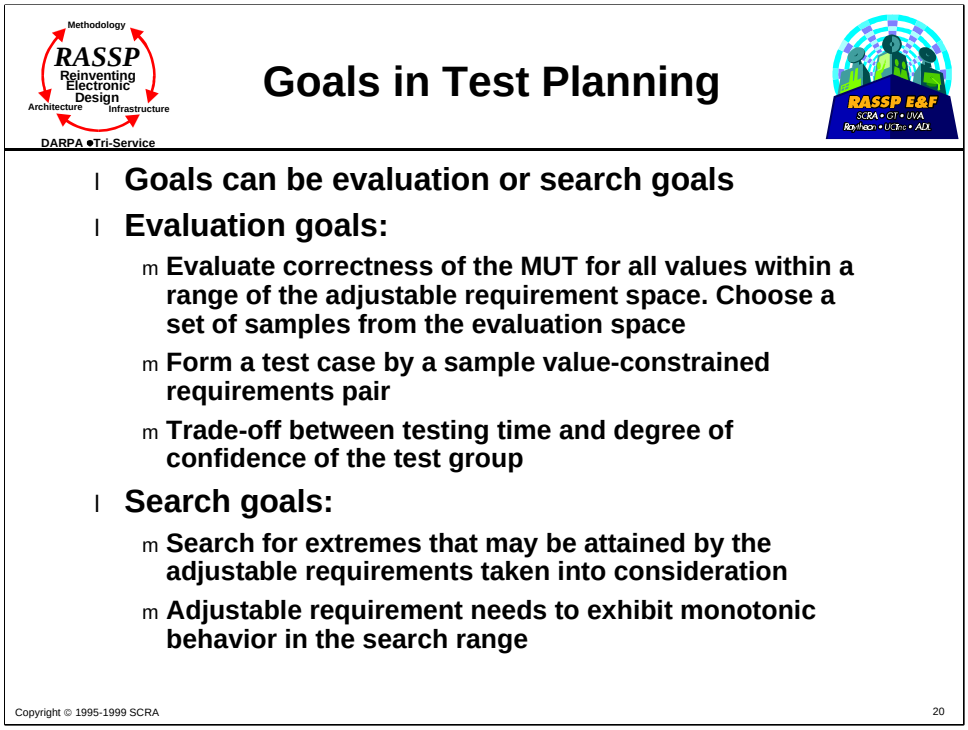
- **Executable Specification features:**
  - **A description of a system or component to reflect function, timing, and other aspects of the intended design**
  - **Operational features of the intended design**
  - **Abstract while retaining necessary detail**
- **Executable Specification contents:**
  - **Fiscal and non-functional constraints**
  - **Behavioral description**
  - **Test bench**
- **Executable Specification Applications:**
  - **At each level of design abstraction in a top-down design methodology**
  - **Decomposition of constraints into subsystems**
  - **Checking of systems and subsystems against testbenches**
  - **Deriving requirements for lower levels**



While the test bench concept originates in the context of testing a model of an integrated circuit, it is applicable to all levels of model abstraction. The environmental conditions can be expressed in input-data and command files and read by the test bench and applied to the processor across the interfaces. The test bench also can read precalculated 'known-good' image files and compare the processor output images to them. In some applications, the test bench, itself, might generate both the input and the comparison data. For systems in which the subsystem being modeled affects the environment, the test bench may be very reactive. Properly constructed, a test bench can be used to test and verify successively more detailed models of a design unit, whether the design unit corresponds to an integrated circuit, a board, or an entire system. The test bench concept, combined with the simulation capability of VHDL, provides a framework for the development of executable requirements.

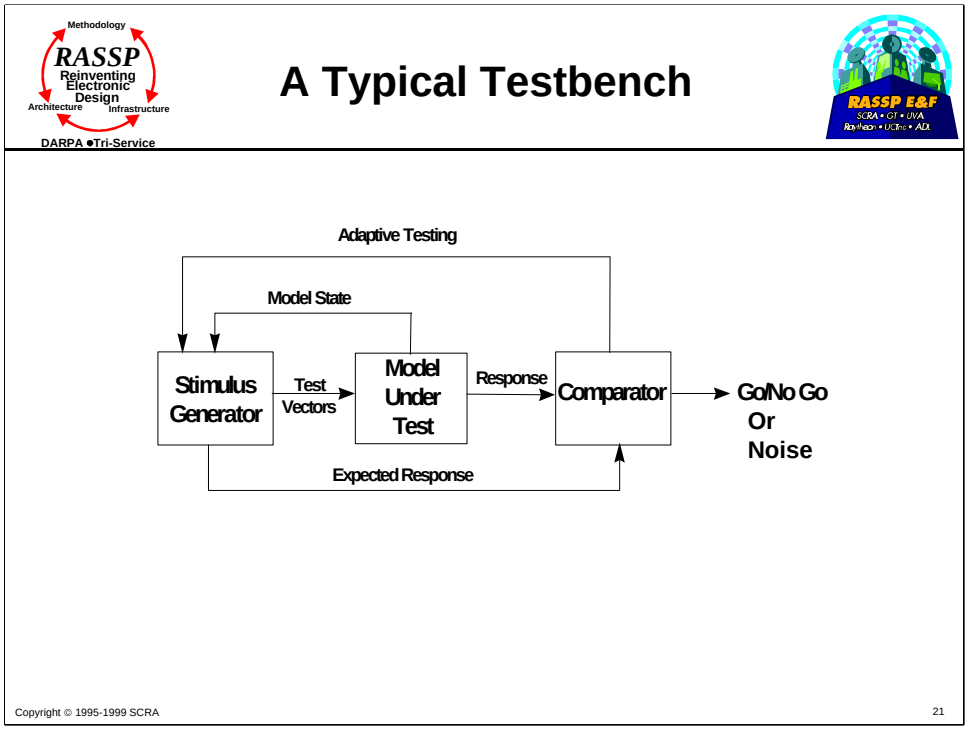
Test Planning

- A test bench is an executable VHDL model which instantiates the Model Under Test (MUT)
- The MUT output is compared against the expected response as generated by the test bench.
- Uses automatic test generation techniques
- Test planning is necessary for model validation

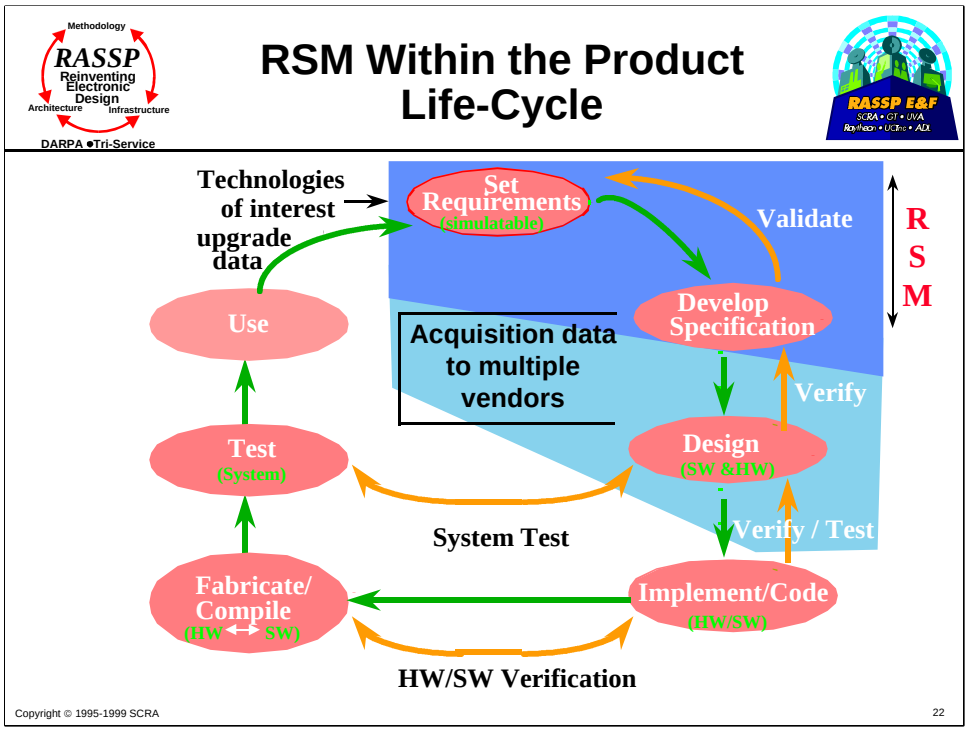


Based on the numerical characteristics, we classify primitive goals into evaluation goals and search goals. The objective of an evaluation goal is to evaluate the correctness of the MUT for all values within a range of the adjustable requirement space (called evaluation space), provided that the other requirements remain unchanged. Instead of testing all possible values, the test group chooses a set of samples from the evaluation space. A test case is formed by a sample value as well as the constrained requirements. Determining the number of test cases is a trade-off between testing time and degree of confidence gained after applying the test group. The more test cases the test group issues, the more time it takes to simulate, and the more evidence one can collect to draw conclusion on the primitive goal.

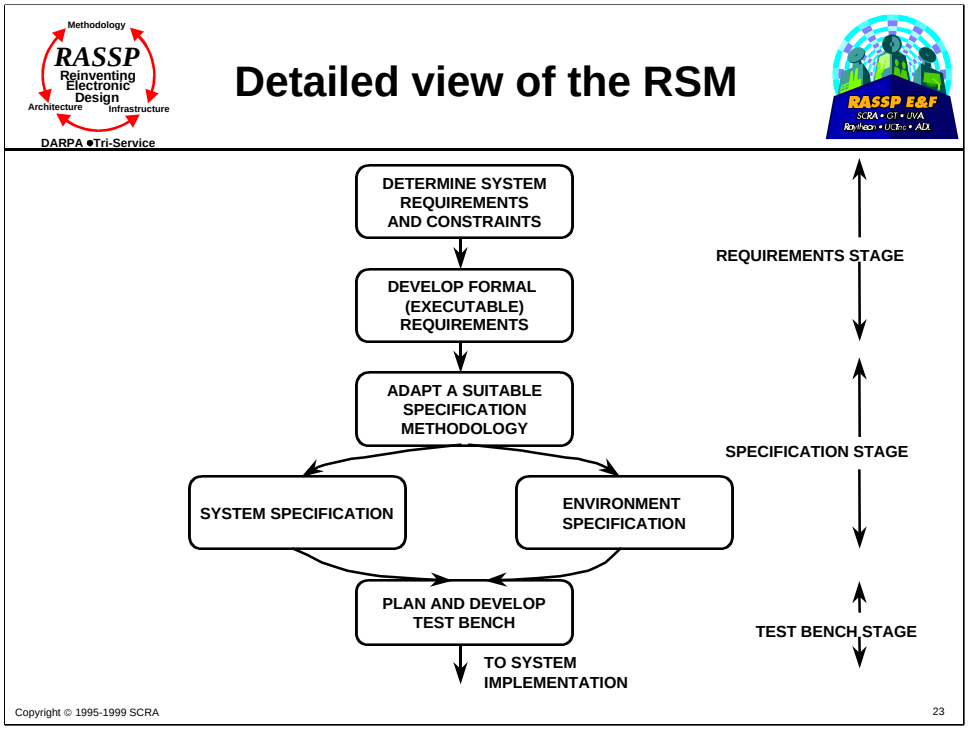
A primitive goal can also be a search goal. In this situation, the test group searches for the extreme value of the adjustable requirement with which the MUT can barely pass/fail the test, provided that the adjustable requirement has a monotonic effect on the MUT in the search range.

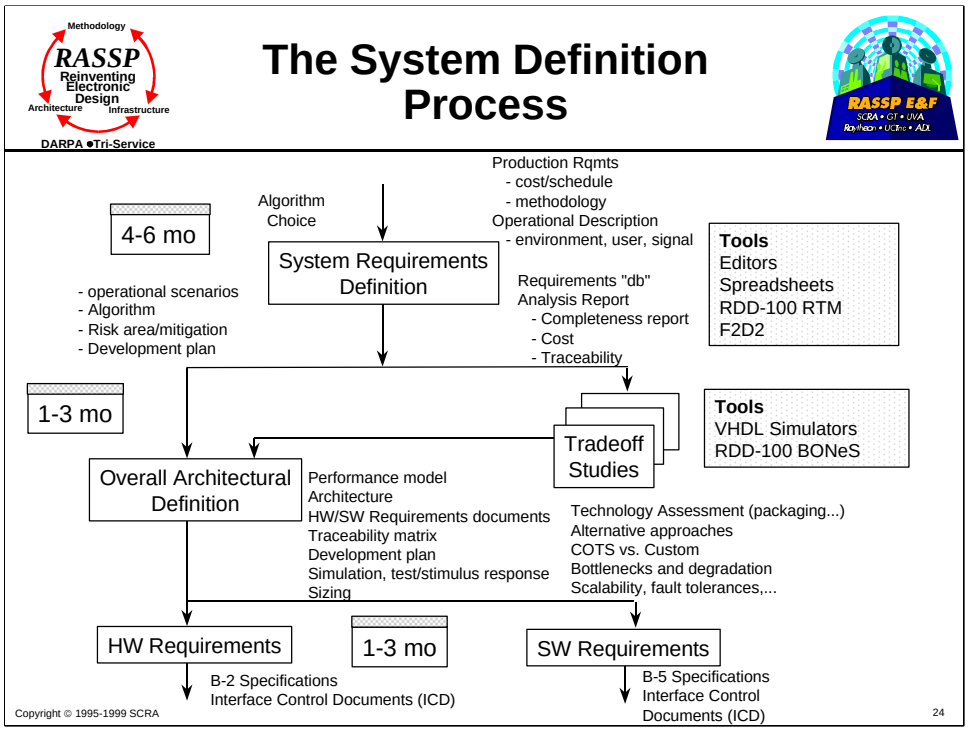


A test bench is an executable VHDL model which instantiates the model under test (MUT), drives the MUT with a set of test vectors, and compares its response with the expected response. The above figure illustrates a typical test bench, which contains a stimulus generator, a MUT, and a comparator. The stimulus generator generates the test vectors, inputs them to the MUT and provides the comparator with the expected response.

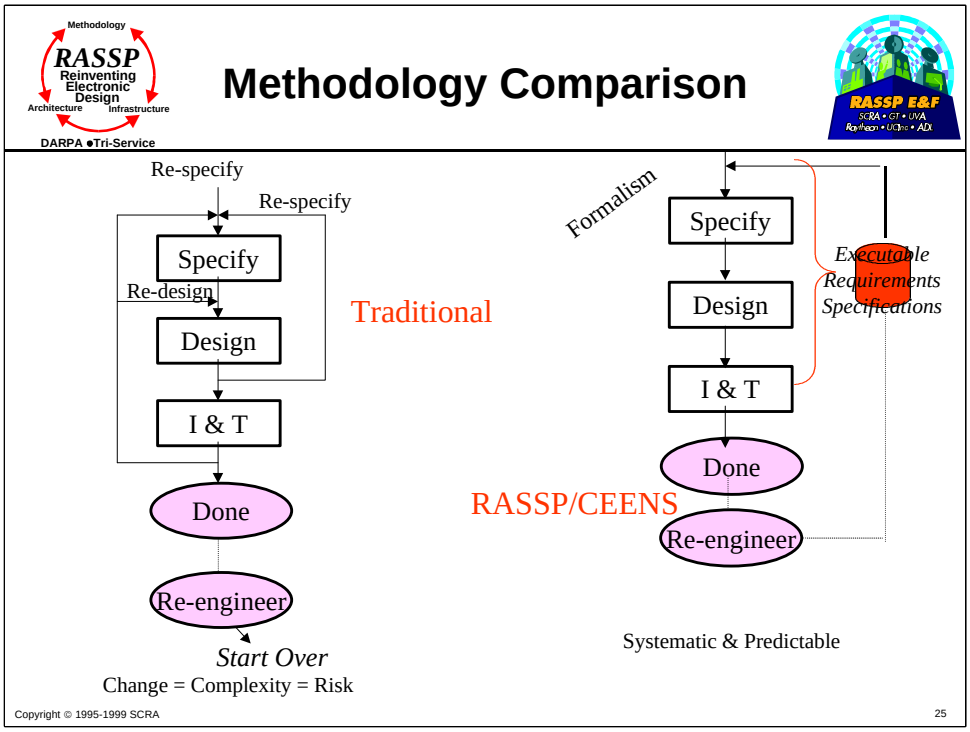


The figure indicates the total life-cycle of an electronics product. A design starts with the requirements that are to be met. Requirements generate concepts which are abstract descriptions. The concepts evolve to specifications and performance models which include the environment and the function of the design. The internal behavior of the software and hardware is next addressed. This is the step where a preliminary attempt is made to partition the design into hardware and software, although some of that decision may have taken place at a much higher level of abstraction, because of the desire to reuse hardware or software. Implementation of hardware and software is next performed. The designer iterates between the specification (which must remain constant) and the implementation to ensure that the implementation of the code and hardware captures the intent of the specification. The implementation must also comply with the design rules required by the fabrication process. The fabrication and compilation process further defines the product. Test is performed on the product in several stages. The chips are tested, mounted chips are tested, and so on. Software is tested on the hardware. The product then moves to the end user. While the product is being used, new requirements are often generated and the cycle begins again.



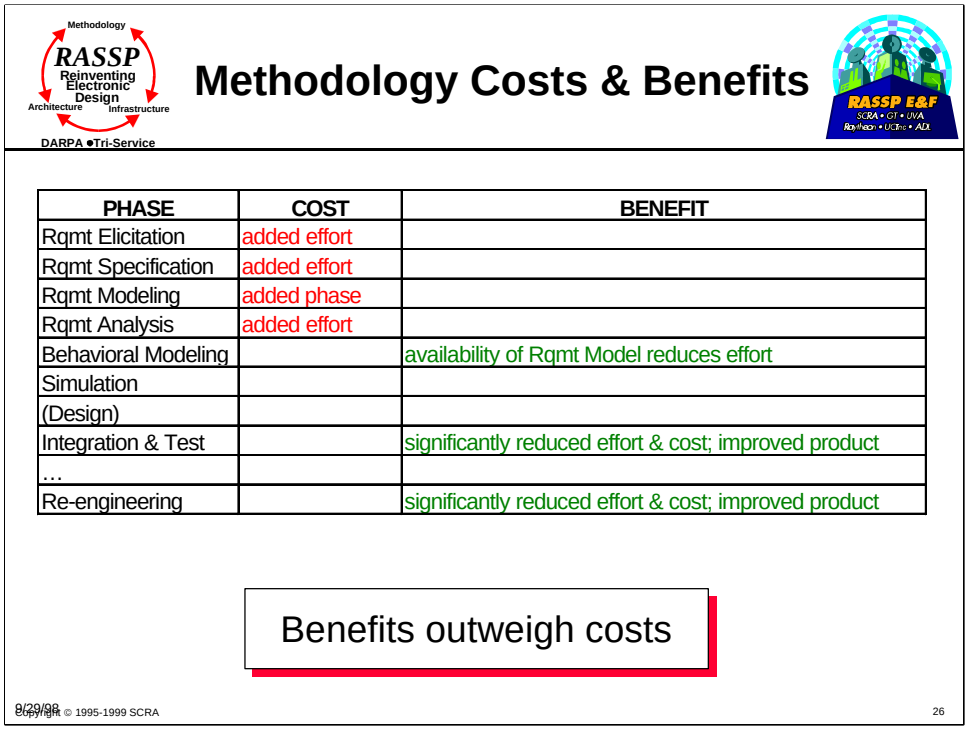


Some tools are being developed and refined to handle the system requirement flowdown. "Executable requirements and specifications" would put the requirements into a machine readable and executable form.



The new approach to RSM is compared with the older approach both in design of new systems, and in the re-design and re-engineering of older systems.

In the both cases, the executable requirements/specifications form the gold requirements/specifications of the design flow, and any changes are captured and stored in that format. In older approach, the actual implementation contained the requirements and specifications (translated to lower levels of abstraction) making it very difficult to upgrade or re-design the system, or validate lower levels of design to higher levels of abstraction.



The benefits and costs of the newer RSM approach is described in this slide. Note that there are up front costs associated with the proposed methodology, but the benefits are expected to outweigh costs.

These results were demonstrated in the RASSP and CEENS efforts.

RSM Outline

- The Motivation for RSM
- **Requirements Engineering and Requirements Analysis**
  - Requirements Engineering Goals
  - Requirements Analysis
  - Requirements Validation
- Specification Modeling Methodologies (SMM)
- SMM Survey
- CASE Tools
- RASSP Requirement Capture and Test Planning
- Summary

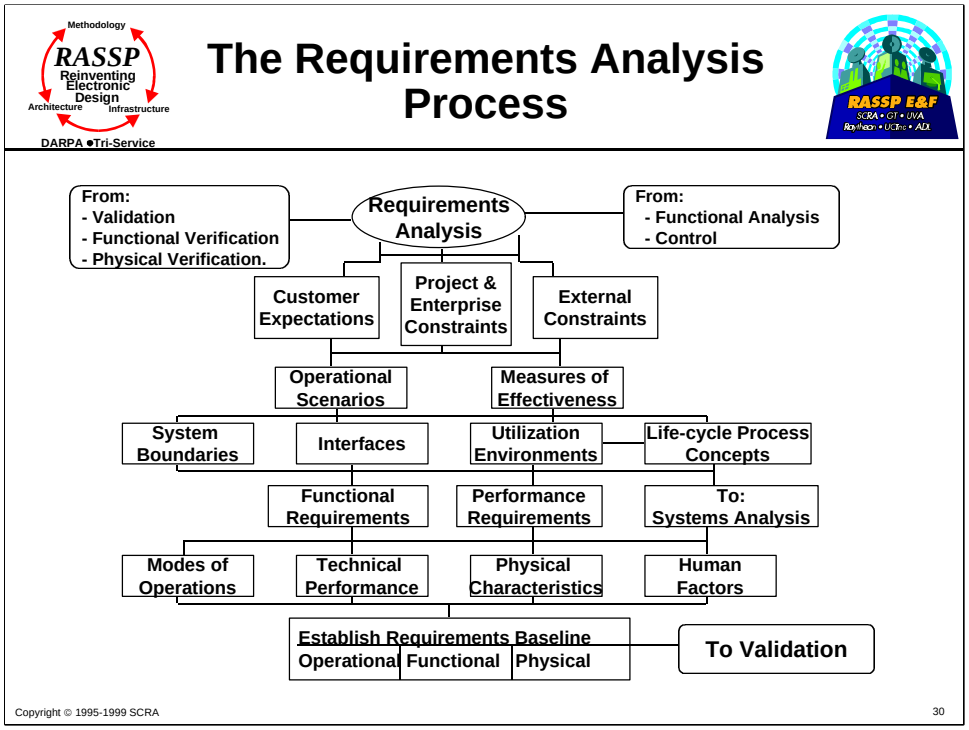
Requirements Engineering Goals

- **Requirements engineering is the description of**
  - **A proposed system's intended behavior and**
  - **Its associated constraints**
  - **By a disciplined application of proven principles, methods, tools and notations.**
- **The following are the chief aims of this activity:**
  - **Identification and documentation of customer and user needs**
  - **Documentation of external behavior and associated constraints**
  - **Ensuring consistency, completeness and feasibility of the required document by analysis and validation**
  - **Sensitivity to the evolution of needs**

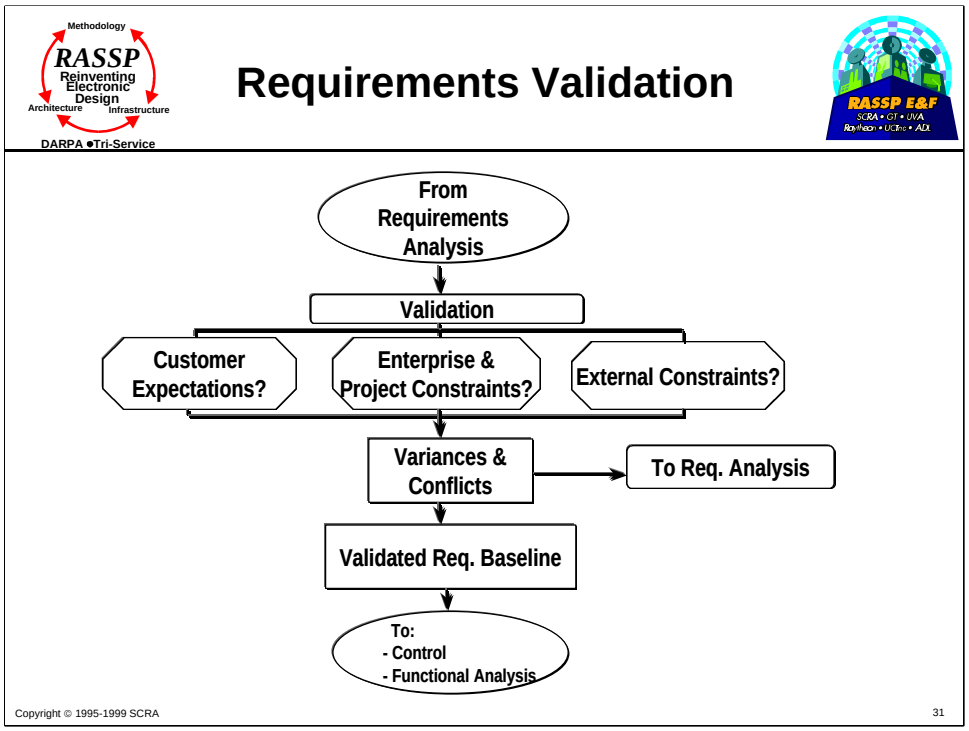
[Hsia]

Requirements Analysis

- **Requirements Analysis Models:**
  - **Functional Model: Analysis of the system by considering its functional aspects**
  - **Structural Model: Analysis of the components or objects which form the system**
  - **Mixed Level Model: Structural analysis and decomposition of both the system and the functionality of the system iteratively**
  - **Communication Model: Description of message exchange between the system components**
  - **Behavioral Model: Description of the internal behavior of the components based on FSMs**



[IEEE1220]



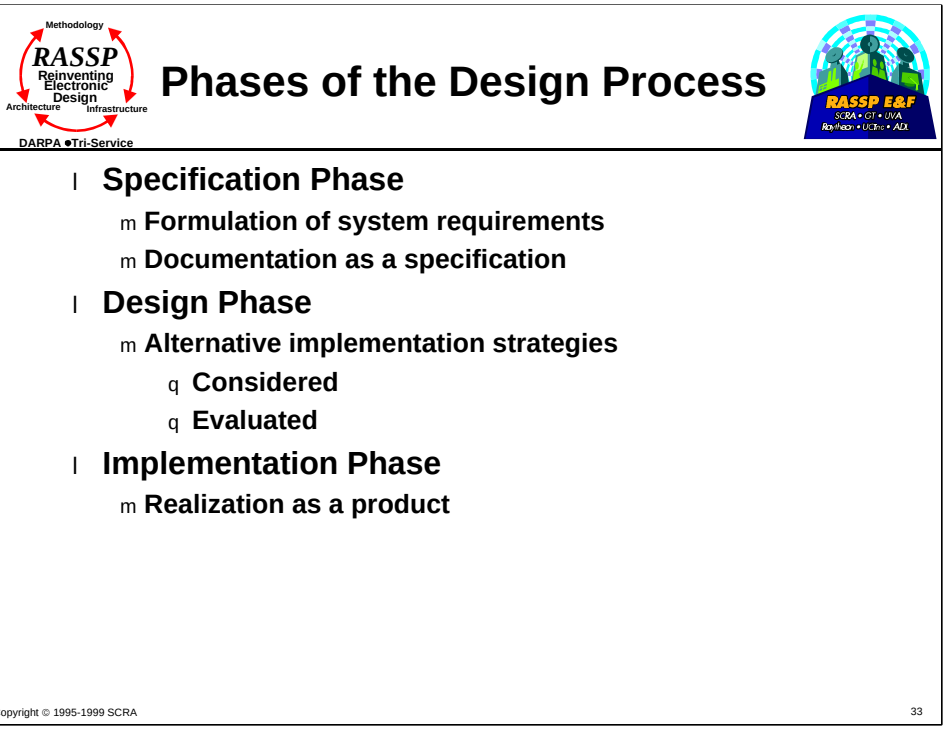
The validation process consists of two types of activities:

- Ensure that the identified customer expectations and project, enterprise, and external constraints are represented by the requirements baseline.

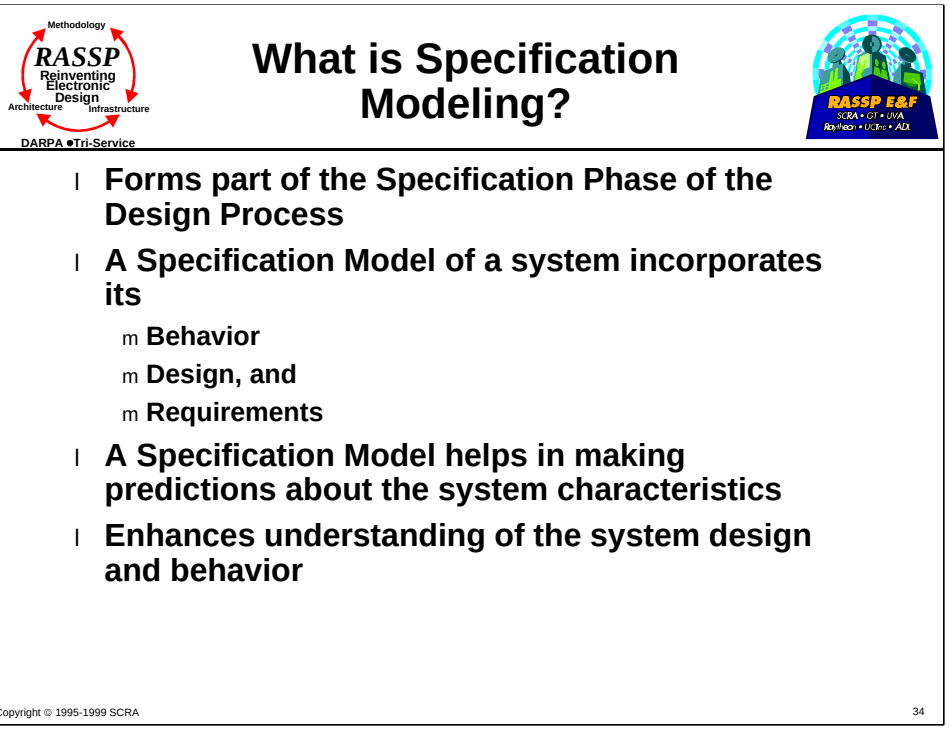
- Assess the requirements baseline to ensure adequate addressing of the entire gamut of possible system operations and system life cycle support concepts.

RSM Outline

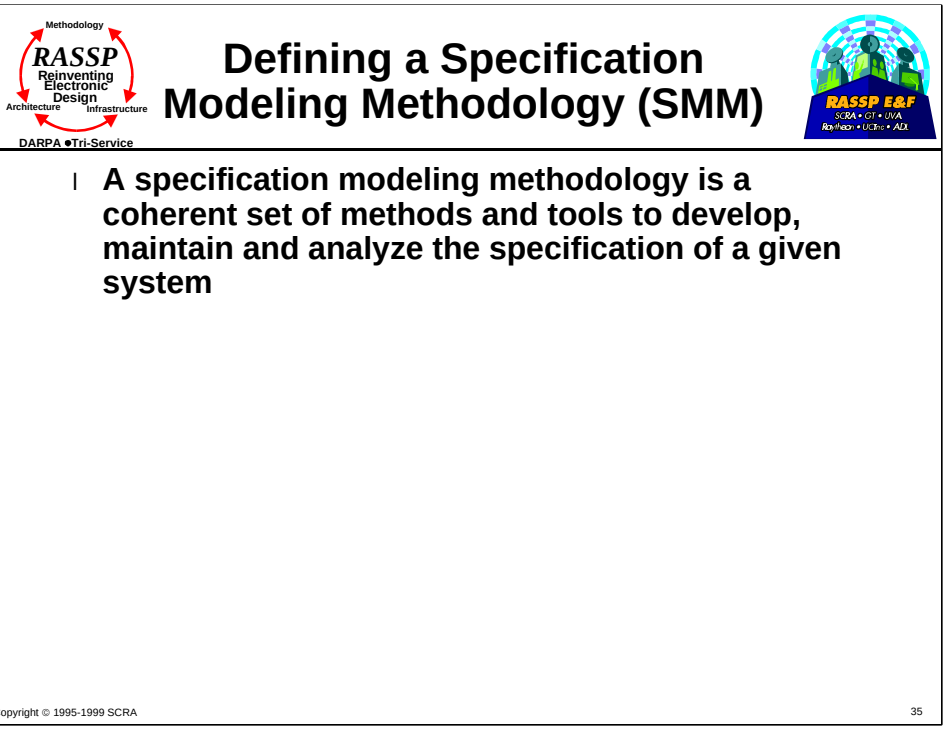
- The Motivation for RSM
- Requirements Engineering and Requirements Analysis
- **Specification Modeling Methodologies**
  - Definition and Application to reactive systems
  - Methodology Attributes
- SMM Survey
- CASE Tools
- RASSP Requirement Capture and Test Planning
- Summary



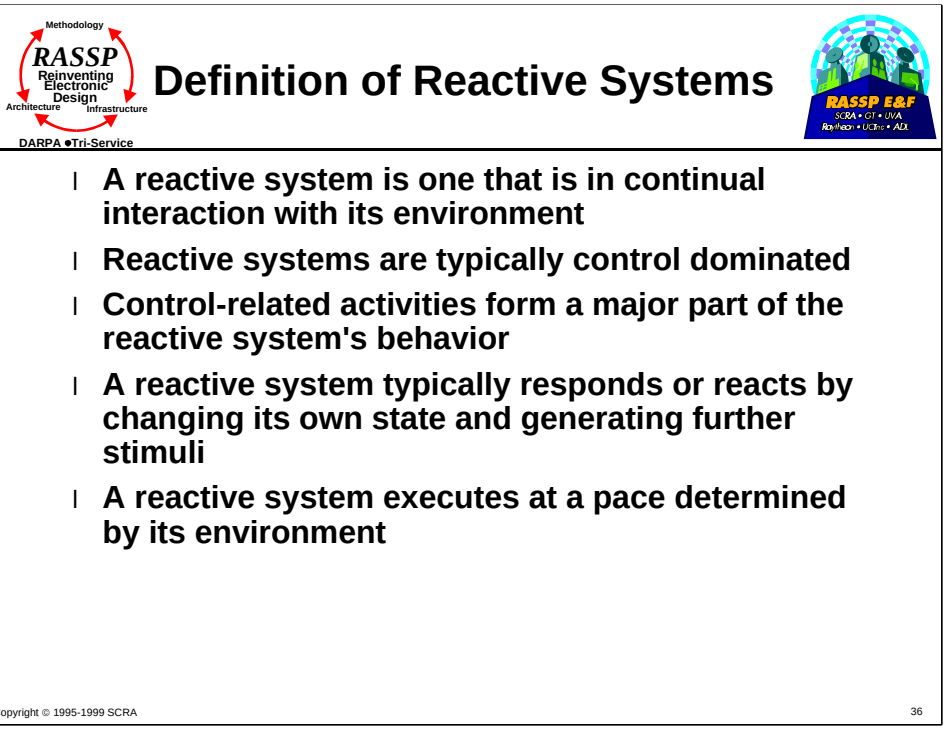
To appreciate the requirements of a specification modeling methodology, one must understand its role in the overall design process. The design process of a reactive system can be segmented into three major phases: the specification phase, the design phase, and the implementation phase. In the specification phase, the requirements of the system under design are formulated and documented as a specification. In the design phase, the possible implementation strategies are considered and evaluated. Finally, in the implementation phase of design, the specification is realized as a product. Note that even though the three phases may be initiated in the order we mention them, these phases often overlap. Overlapping of phases implies that during the design process, one phase may not be completed before the next phase is initiated. Another point to note is that in some methodologies, it can be difficult to distinguish the three phases from one another, especially when the difference in the levels of abstraction between the specification and implementation is small. In such cases, the specification is often a reflection of the implementation, and the process of developing specification may reflect the design phase.



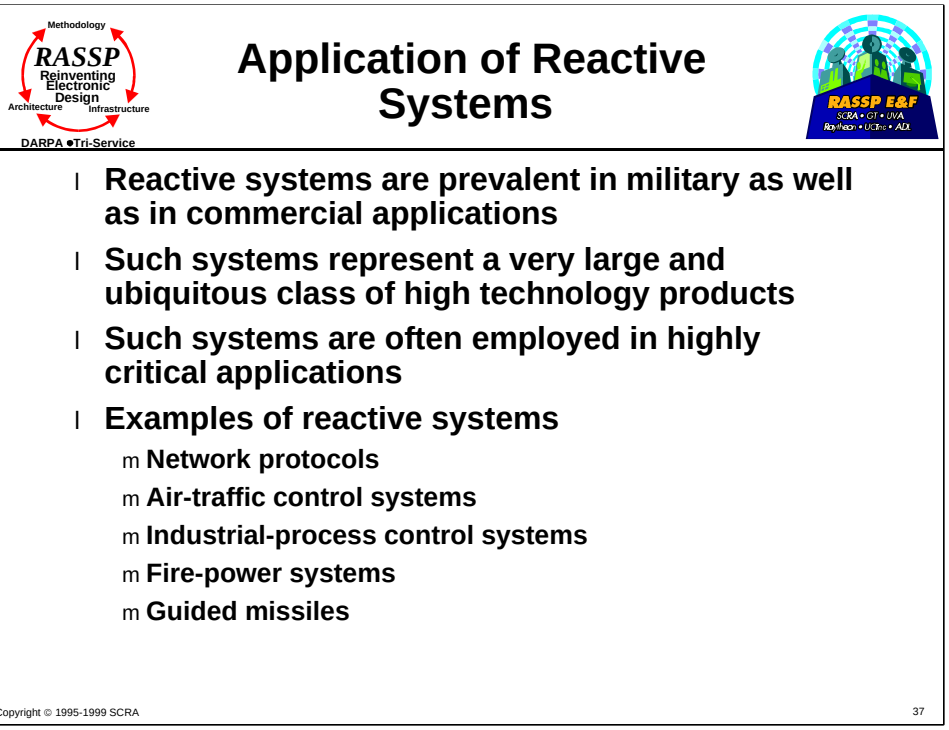
We are concerned with the specification phase, where the designer, or specifier, typically develops a model of the system called the specification model. A specification model, or simply, a specification is a document that prescribes the requirements, behavior, design, or other characteristics of a system or system components. By developing and analyzing the model, the specifier makes predictions and obtains a better understanding of the modeled aspects of the system.



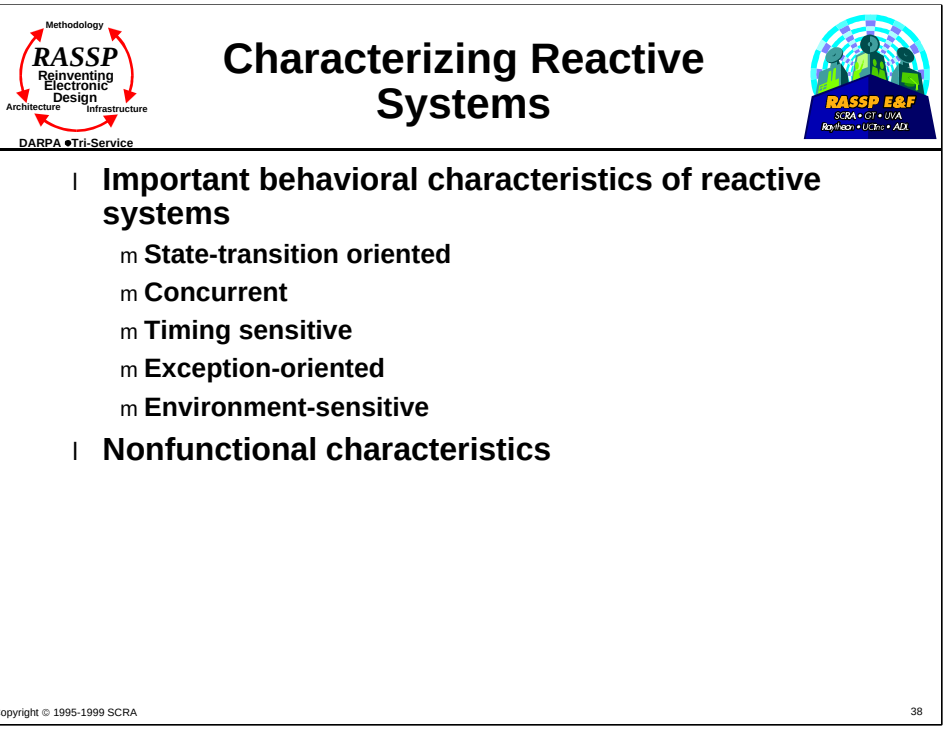
We now describe how a Specification Modeling Methodology is described, evaluated and implemented.



A reactive system is one that is in continual interaction with its environment and executes at a pace determined by that environment. This is to be contrasted to the type of system that merely transforms a set of inputs to a defined set of outputs. These are known as "transformational systems." A reactive system typically responds or reacts by changing its own state and generating further stimuli. Reactive systems are typically control dominated, in the sense that control-related activities form a major part of the reactive system's behavior.

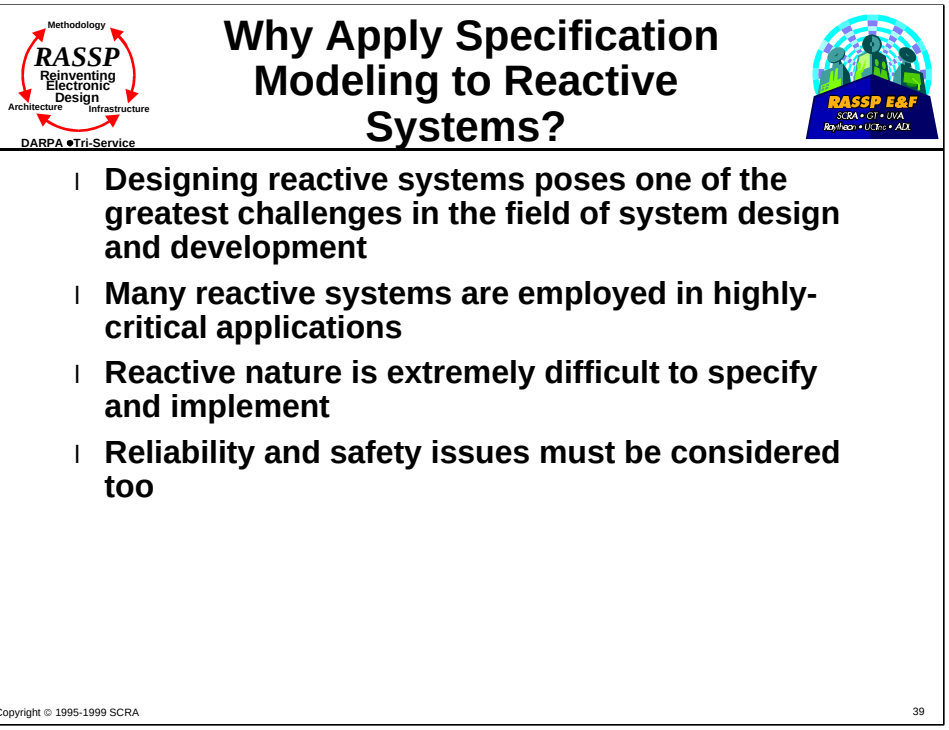


Reactive systems represent a very large and ubiquitous class of high technology products, a class of products that are prevalent in military as well as in commercial applications. Such products are often employed in highly-critical applications. Even controlling the level of water in a washing machine may be considered time-critical to the owner of the washing machine. Examples of reactive systems are network protocols, air-traffic control systems, industrial-process control systems, fire power systems, guided missiles, etc.

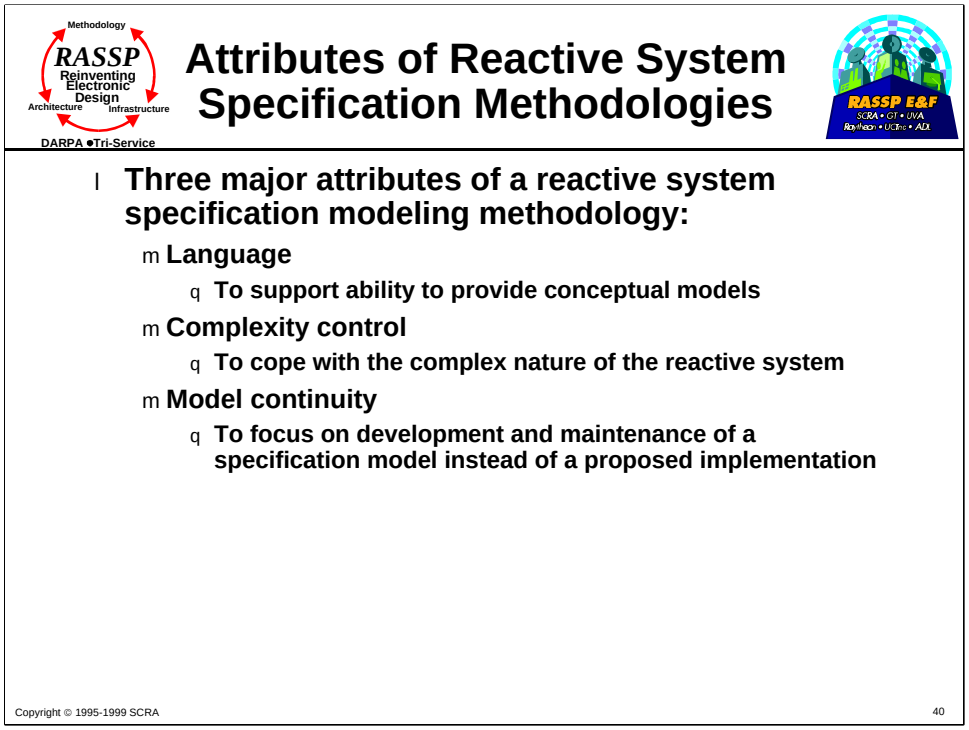


Reactive systems have typically been contrasted with transformational systems. The behavior of a transformational system can be adequately characterized by specifying the outputs of the system that result from a set of inputs to the system. In contrast, the behavior of a reactive system is characterized by the notion of reactive behavior . To describe reactive behavior, the relationship of the inputs and outputs over time should be specified. Typically, such descriptions involve reactive sequence of system states, generated and perceived events, actions, conditions and event flow, often involving timing constraints.

The expression of all the characteristics of a reactive system should be directly supported by the language of specification. Since a specification methodology is typically associated with a specific set of specification languages, the effectiveness of the methodology lies in how well its specification languages support the expression of the above characteristics. In addition to the language, the methodology plays an important role in developing the representation in a methodical rather than a haphazard manner.



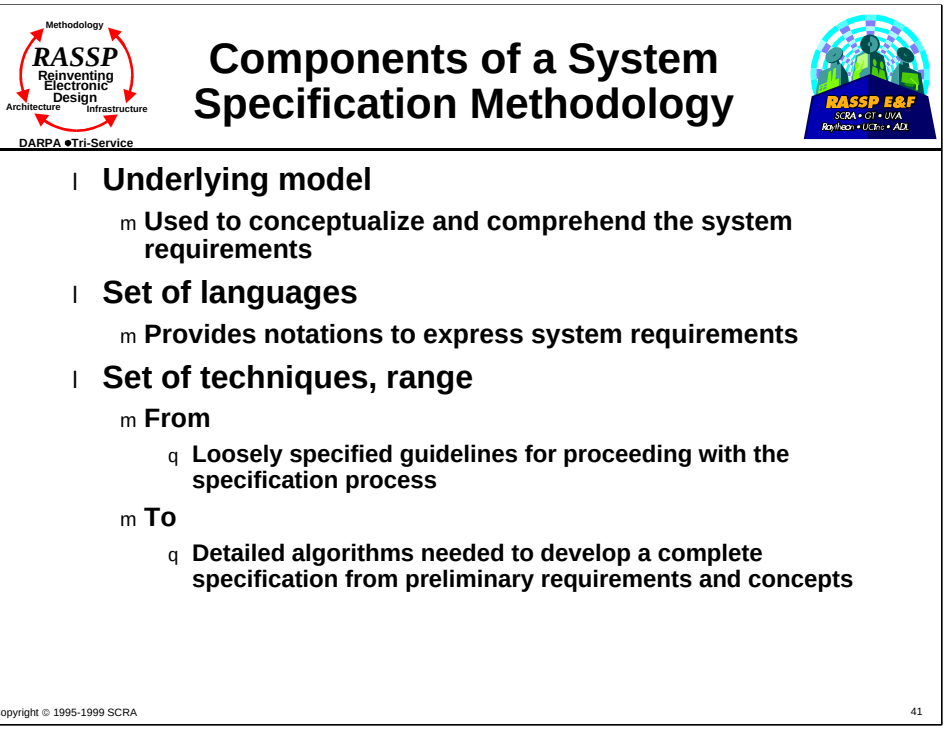
Due to their complex nature, reactive systems are extremely difficult to specify and implement. Many reactive systems are employed in highly-critical applications, making it crucial that one considers issues such as reliability and safety while designing such systems. Designing reactive systems is considered to be problematic, and poses one of the greatest challenges in the field of system design and development.



The language attribute represents the modeling languages that support the methodology. This attribute distinguishes reactive system specification modeling methodology from other specification methodologies, since the chosen languages should provide appropriate conceptual models and analysis techniques applicable to reactive systems.

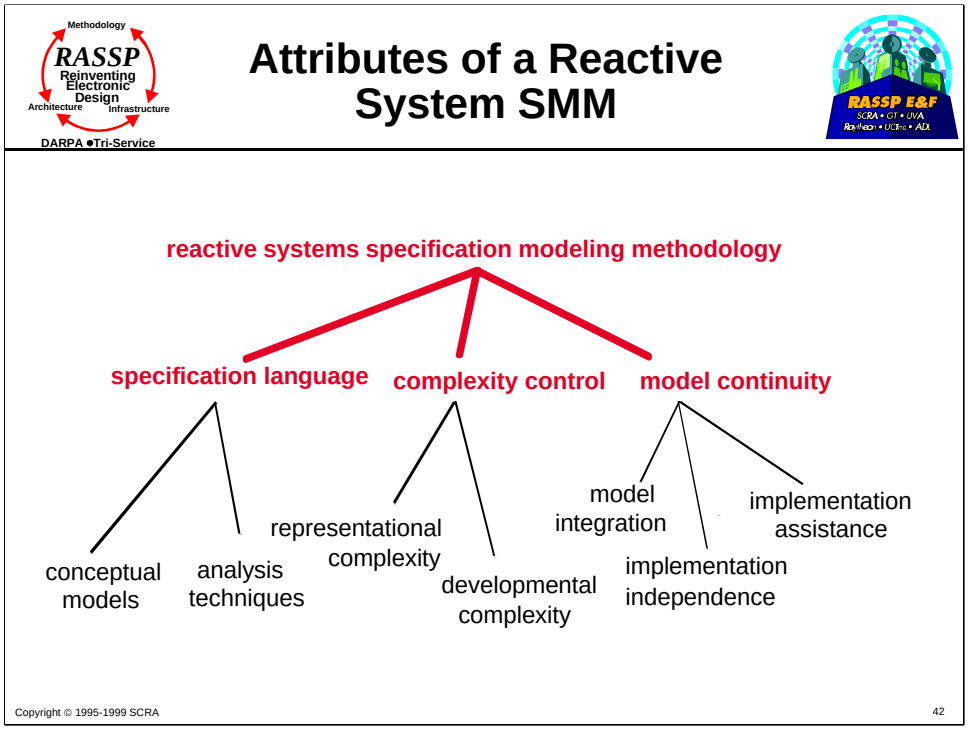
Complexity-control is necessary for any design methodology that is applicable to design problems of nontrivial complexity. There are two dimensions along which complexity control should be supported: representational complexity and developmental complexity.

Model-continuity distinguishes a reactive system specification modeling methodology as a specification methodology, instead of a design methodology. The specification modeling methodology should be more focused towards developing and maintaining a specification model instead of a proposed implementation. Support of model-continuity should be considered along three dimensions: integrated modeling, implementation independence, and implementation assistance.



A method, in the context of specification modeling, consists of three components. The first component is an underlying model which is used to conceptualize and comprehend the system requirements. The second component is a set of languages that provides notations to express the system requirements. Finally, the third component of a method is a set of techniques ranging from loosely specified guidance to detailed algorithms that is needed to develop a complete specification from preliminary concepts.

We focus mostly on the methods. The tools are important. However, the tools are primarily concerned with providing support for the methods. Therefore, the tools can be characterized by the method component of a methodology and are not considered separately.



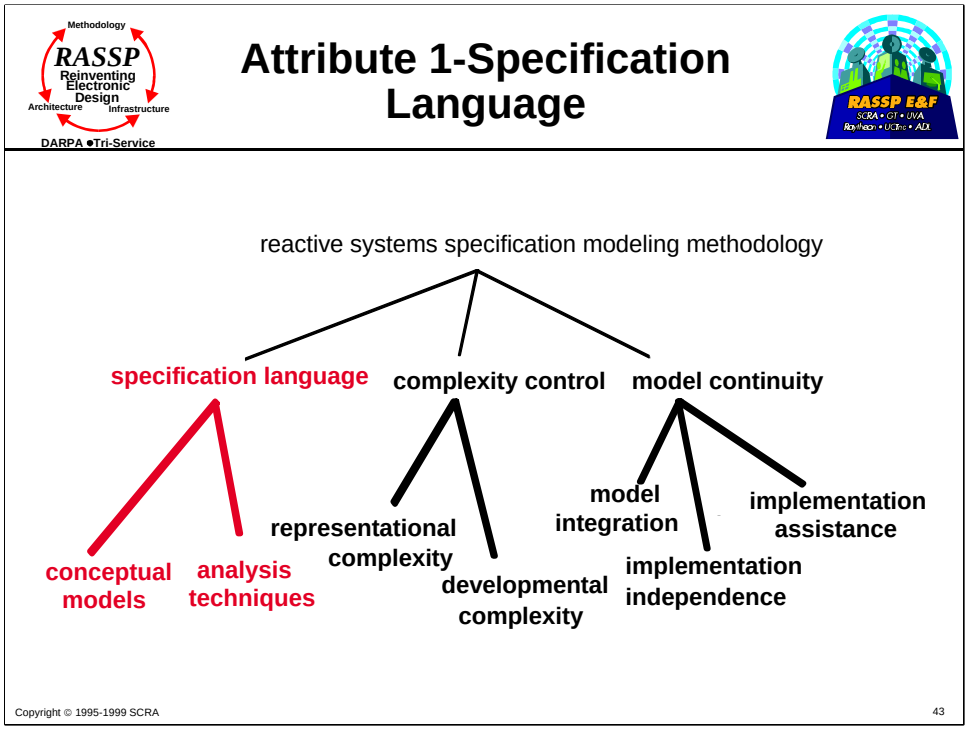
Based on the characteristics of a reactive system and the requirements of a specification-modeling methodology, we define the necessary attributes of a reactive-system specification modeling methodology. There are three major attributes of a reactive system RSM methodology: language attribute, complexity-control attribute, and model-continuity attribute.

The **language attribute** represents the modeling languages that support the methodology. This attribute distinguishes reactive system RSM methodology from other specification methodologies, since the chosen languages should provide appropriate conceptual models and analysis techniques applicable to reactive systems.

The second attribute represents the support available in the methodology for **complexity control**. Complexity-control is necessary for any methodology that is applicable to problems of nontrivial complexity.

The third attribute, support for **model-continuity**, distinguishes a reactive system RSM methodology as a specification methodology, instead of a design methodology. Specification modeling methodology must be focused on specification model development and maintenance instead of implementation.

[Sarkar95] and Sarkar under additional readings.



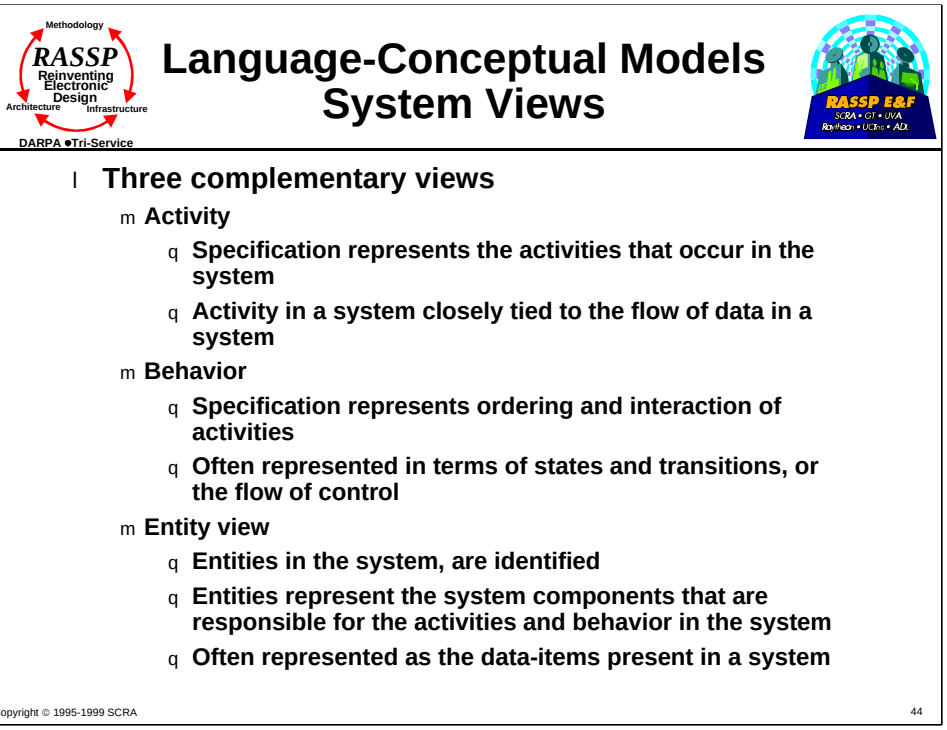
There are two dimensions of the **language** attribute. The conceptual-models dimension determines the available conceptual models for describing reactive systems. The analysis-technique dimension determines the kind of support available for checking the specification consistency.

There are two dimensions along which **complexity control** should be supported: representational complexity and developmental complexity. Representational complexity deals with the clarity of the developed specification, whereas developmental complexity provides support for developing the specification in an organized and productive manner.

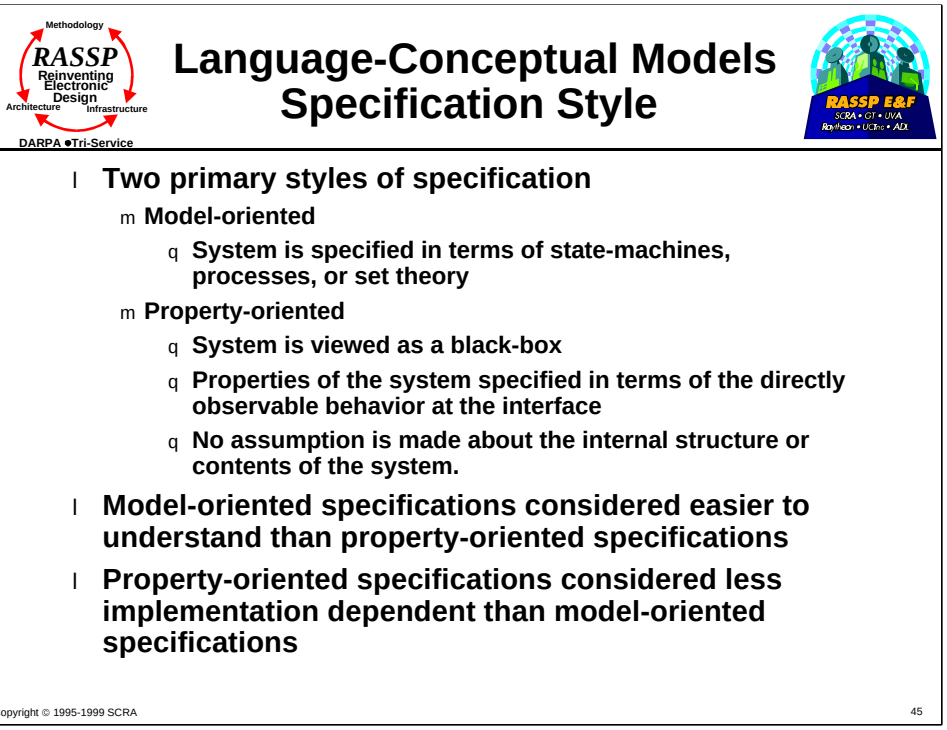
Model-continuity has three dimensions: integrated modeling, implementation independence, and implementation assistance. Support along these three dimensions ensures that usefulness of the specification model is maintained beyond the specification modeling stage of a design.

A reactive system SMM must strongly support the attributes stated. Each of these attributes is described in the following sections. The effectiveness of the methodology can be identified by evaluating the strengths of each of these components.

The following slides will discuss each of these attributes and their dimensions in greater detail.



There are three views that are complementary to each other: activity, behavior, and entity view. In the activity view, the specification represents the activities that occur in the system. Activity in a system is closely tied to the flow of data in a system. In the behavior view, the specification represents the ordering and interaction of these activities. Behavior in a system is often represented in terms of states and transitions, or the flow of control. Finally, in the entity view, the entities in the system, are identified. These entities represent the system components that are responsible for the activities and behavior in the system. Entities in a system are often represented as the data-items present in a system.

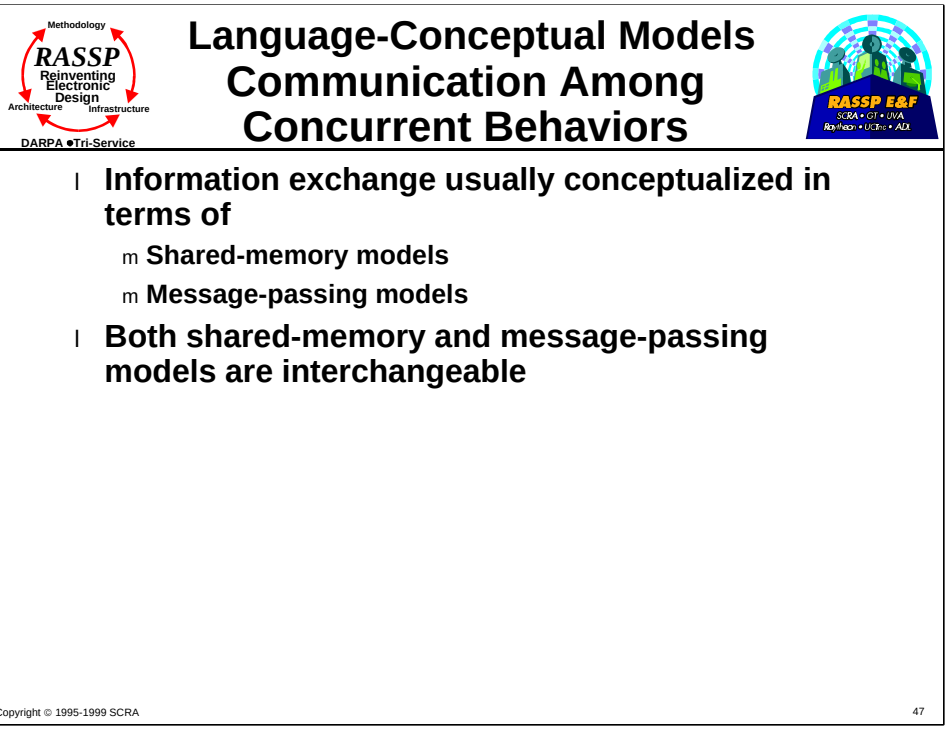


There are two primary styles of specification: model-oriented and property-oriented. In a model-oriented specification, the system is specified in terms of a familiar structure such as state-machines, processes, or set theory. In a property oriented specification, the system is viewed as a black-box, and the properties of the system are specified in terms of the directly observable behavior at the interface of the black-box. Generally speaking, model-oriented specifications are considered easier to understand than their property-oriented counterparts. On the other hand, property-oriented specifications are considered less implementation dependent since no assumption is made about the internal structure or contents of the system.

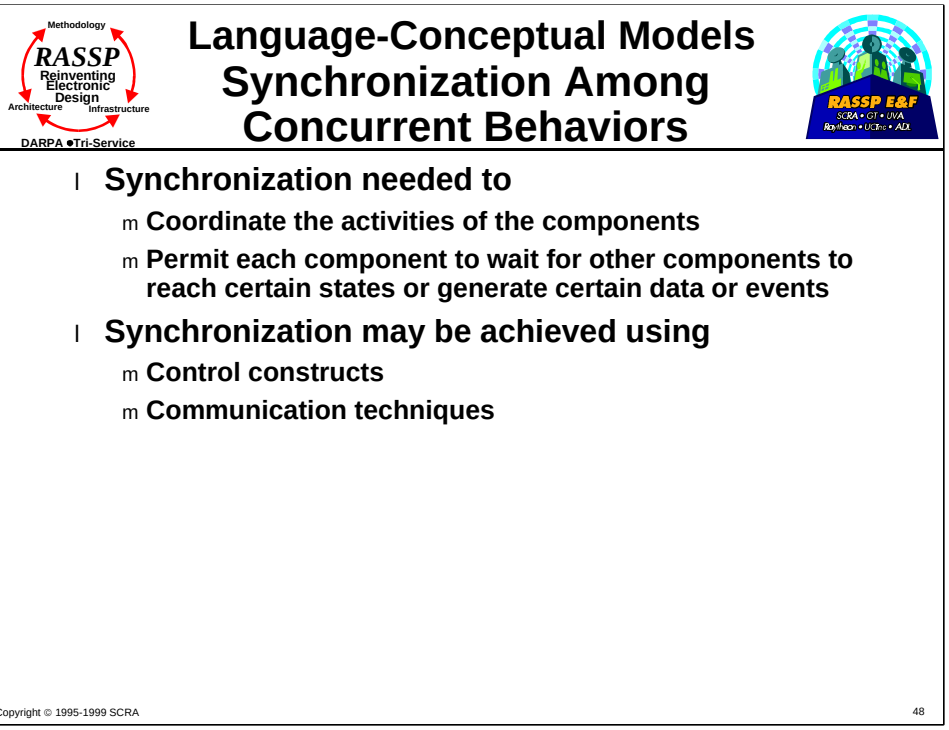
Language-Conceptual Models Concurrency

- **Concurrent behaviors cooperate with each other to achieve the desired functionality**
- **Concurrency is further characterized by the ability to express communication and synchronization among concurrent behaviors.**

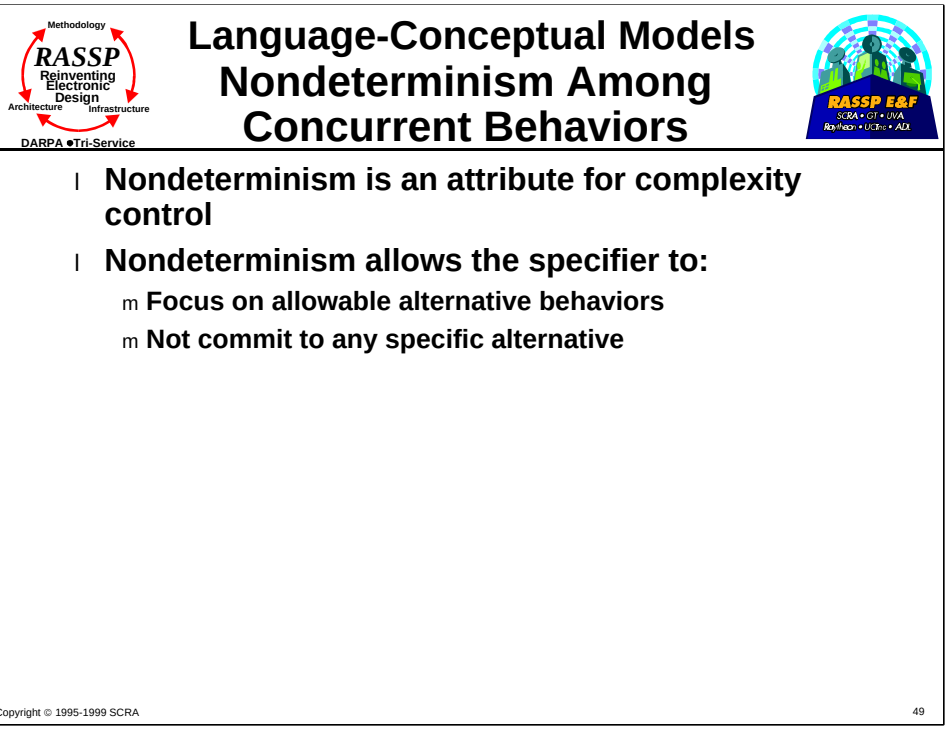
Reactive systems generally consist of concurrent behaviors that cooperate with each other to achieve the desired functionality. Concurrency is further characterized by the ability to express communication and synchronization among concurrent behaviors.



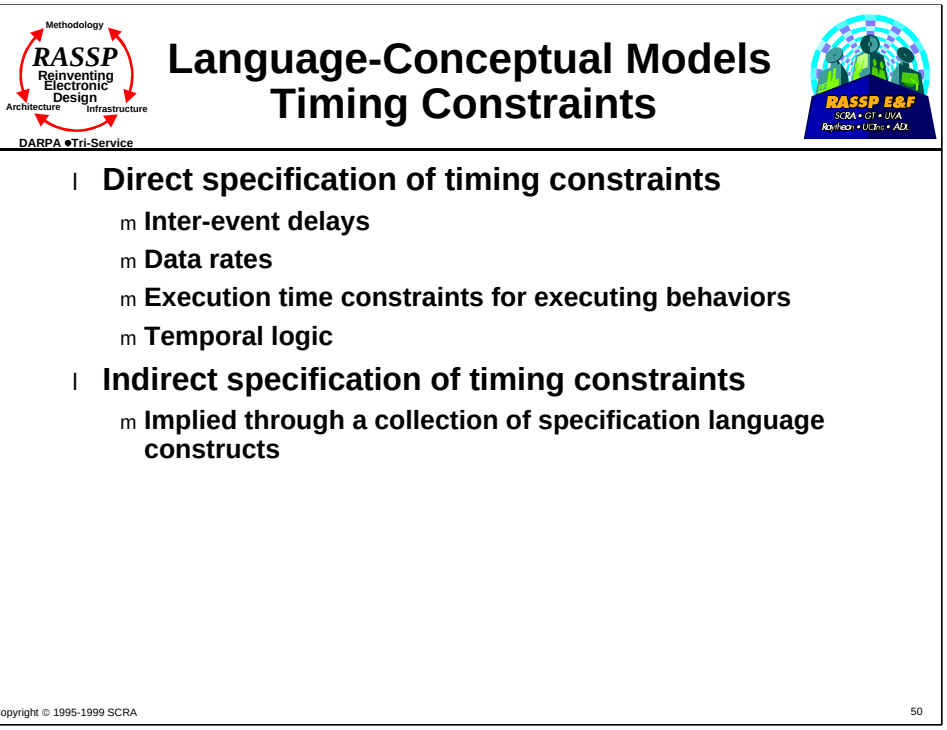
Communication between concurrently acting portions of a system is usually conceptualized in terms of shared-memory or message-passing models. In the shared-memory model, a shared medium is used to communicate information. The communication is initiated by the sender process writing into a shared location, where it is available immediately for all receiver processes to read. In the message-passing model, a virtual channel is used, with "send" and "receive" primitives used for data transfer across that channel. Both shared-memory and message-passing models are interchangeable: each model can be expressed in terms of the other model.



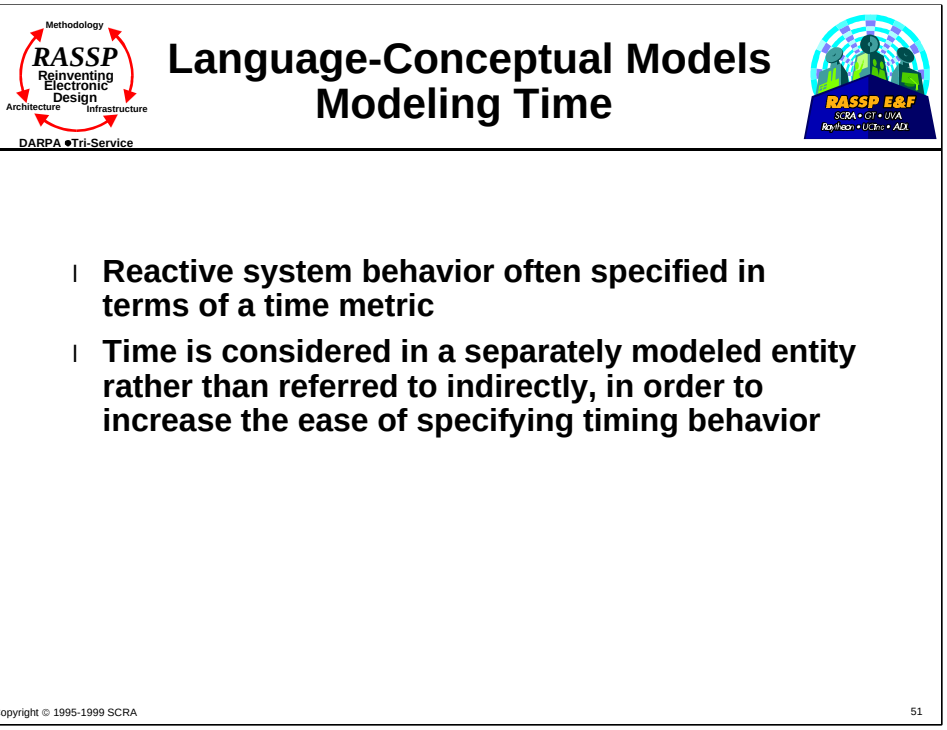
In addition to exchanging of data, the concurrent components of a system need to be synchronized with one another. Such synchronization is needed since the concurrent components often need to coordinate their activities, and have to wait for other components to reach certain states or generate certain data or events. Synchronization may be achieved using control constructs or communication techniques. Examples of control constructs are fork-join primitives, initialization techniques, barrier synchronization etc. Examples of using communication techniques for synchronization are global-event broadcasting, message passing, global status detection etc.



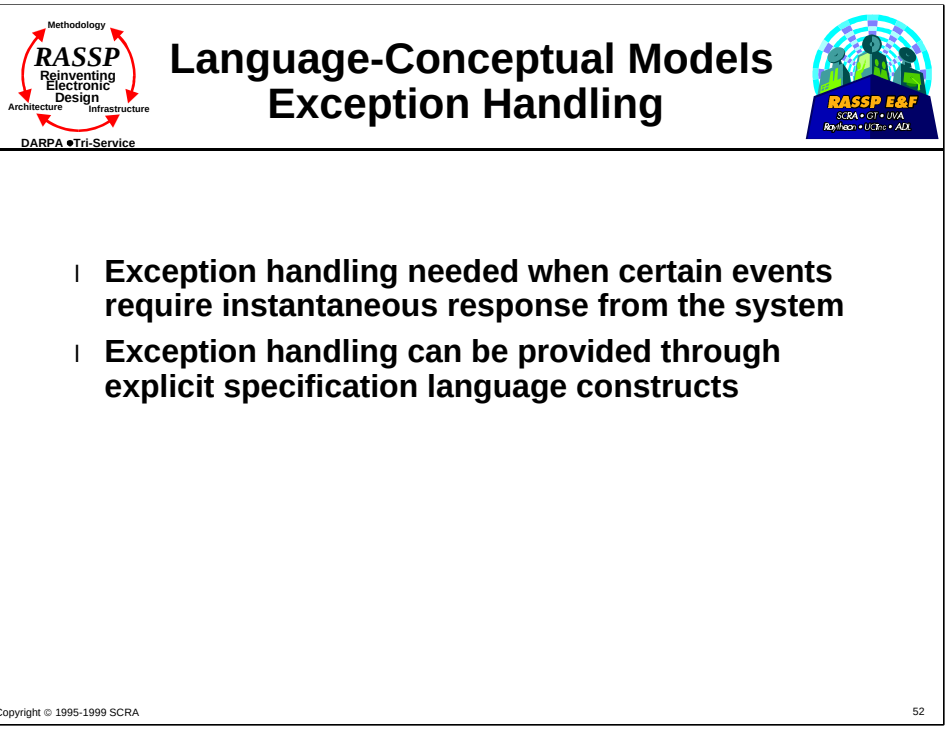
In addition to communication and synchronization, a specification language often supports expression of nondeterminism among concurrent behaviors . We consider nondeterminism an attribute for complexity control rather than a reactive system characteristic, since it allows the specifier to focus on the allowable alternative behaviors without committing to any specific choice. This choice is determined at a later stage depending upon the implementation.



Timing constraints are an important part of any reactive system behavior and can be specified either directly or indirectly. Timing constraints can be specified directly as inter-event delays, data rates, or execution time constraints for executing behaviors. Supporting temporal logic can be seen as a direct specification technique. Indirect specification of timing constraints occurs when the actual constraint is implied through a collection of specification language constructs. This approach is followed in Statecharts where timing constraints are specified by a combination of states, transitions, and time-outs.

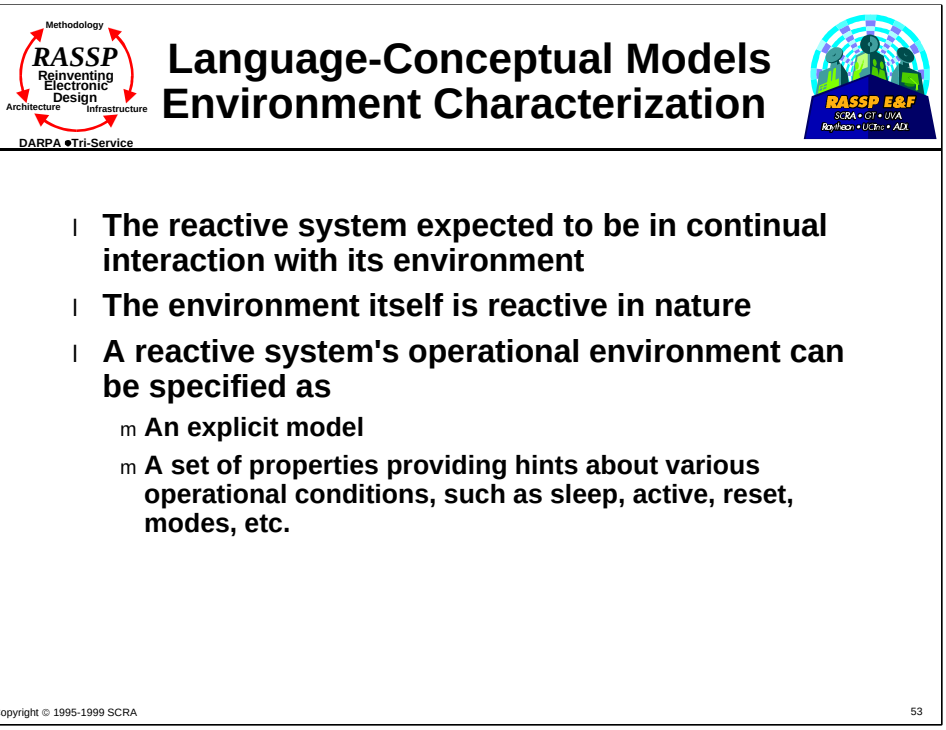


Reactive system behavior is often specified in terms of their time metric (e.g. wait for 5 minutes to warm up electromechanical equipment). Consideration of time in a separately modeled entity increases the ease with which the specifier can specify such timing behavior, instead of referring to time indirectly.

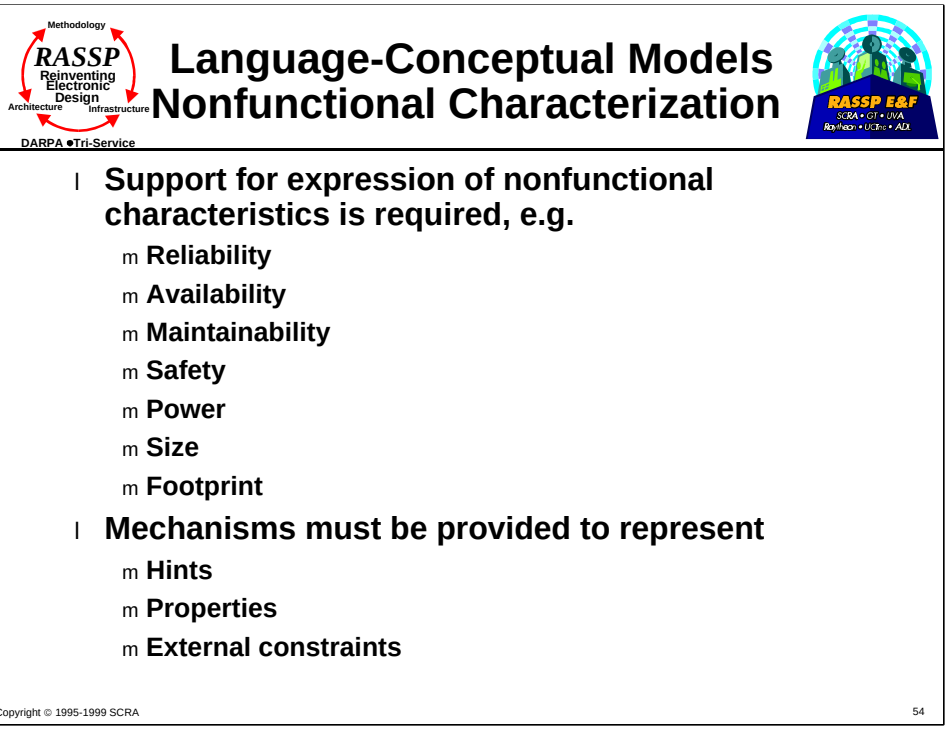


Certain events may require instantaneous response from the system. This requires the system to typically terminate the current mode of operation and transition into a new mode. In some cases, the system is required to resume in the original state at which the interrupt occurred. Interrupt handling is one such case.

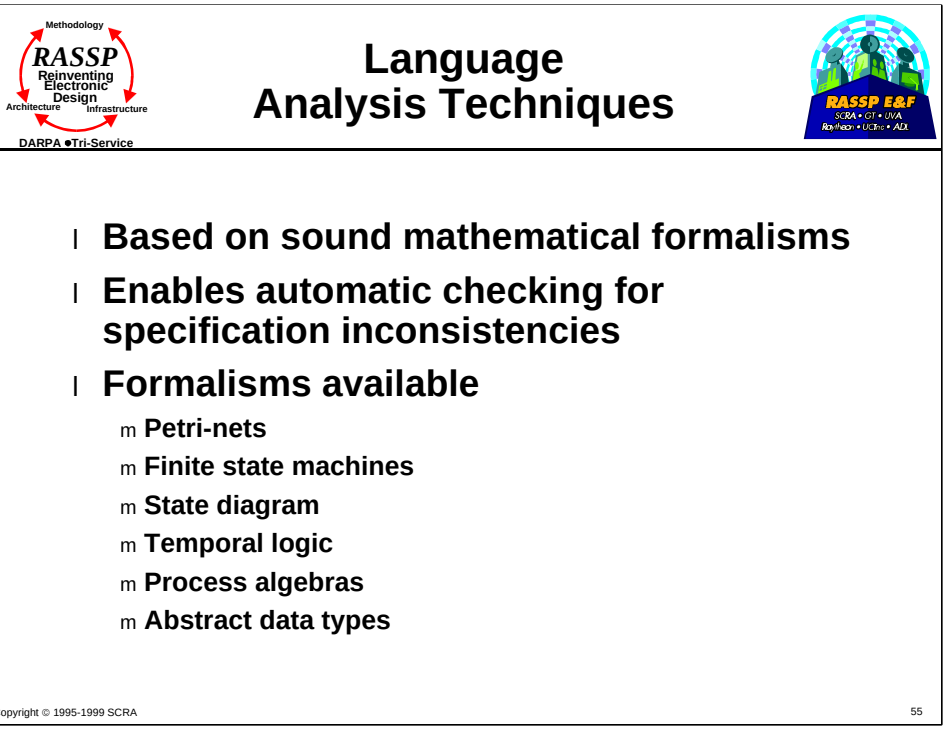
Exception handling can be provided through explicit specification language constructs. Examples are text-oriented exception handling mechanisms in Ada programming language or graphics-oriented mechanisms such as reactive transitions and history operators in Statecharts.



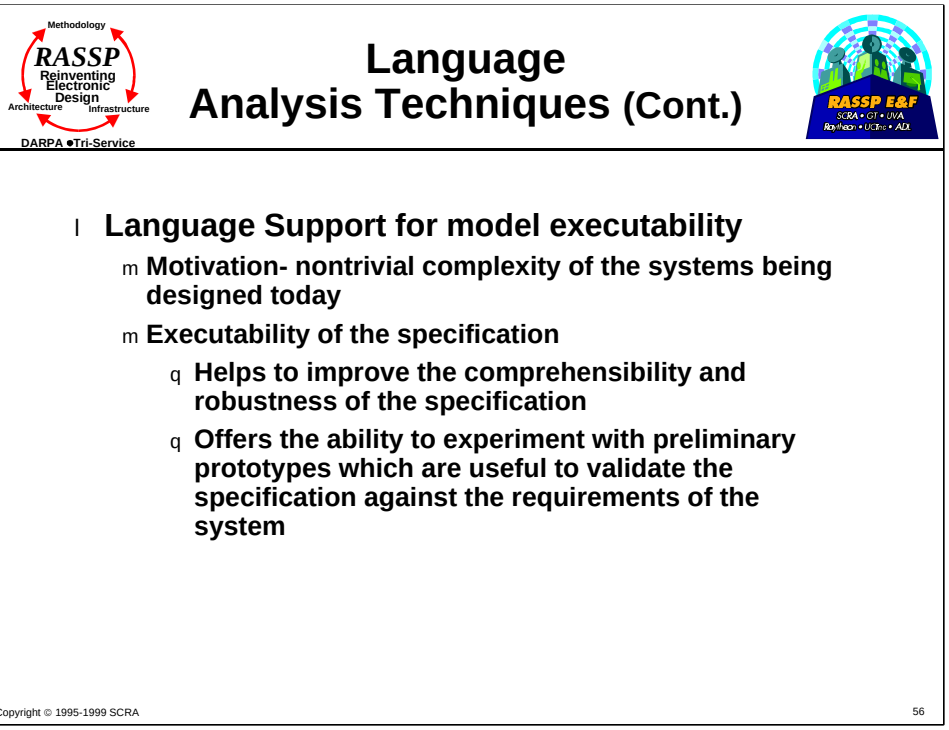
Since the reactive system is expected to be in continual interaction with its environment, it is important to be able to characterize the environment. Since the environment itself is reactive in nature, one may choose to model it using the same specification language. The operational environment of a reactive system can therefore be specified as an explicit model or as a set of properties. When specified as a separate model, the environment is seen as a separate entity that interacts with the model of the system under design. For property oriented characterization, the system's operational environment can be specified via hints about various operational conditions such as inter-arrival of events (expected frequencies, timings), expected work-loads, etc.



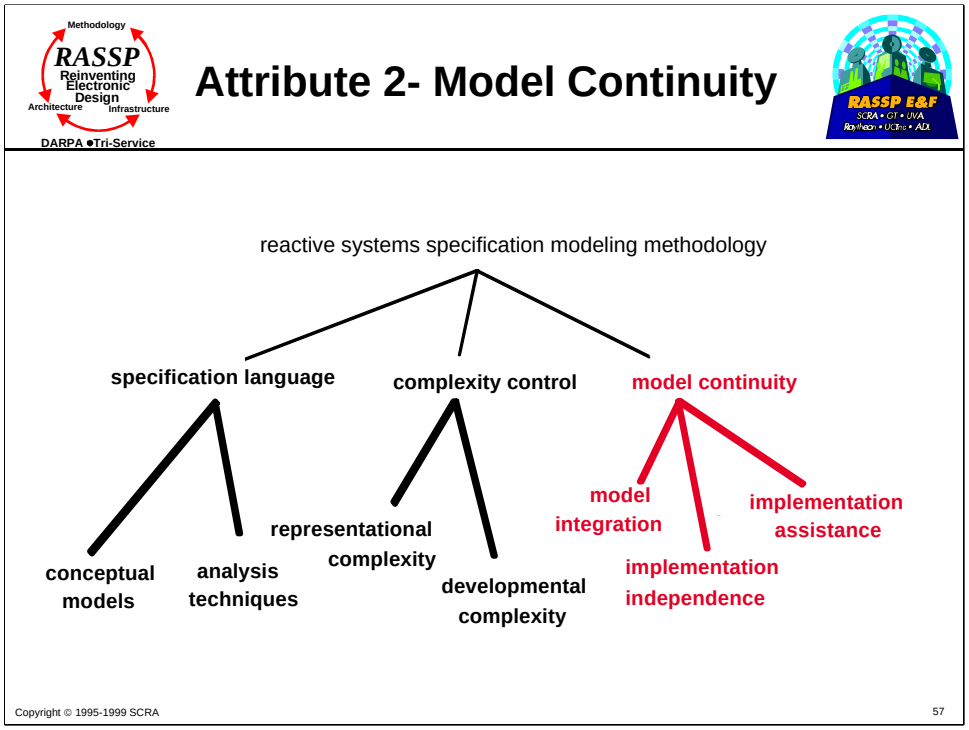
In addition to providing conceptual models for specifying the functionality of the system, a specification methodology for a reactive system should also provide support for expressing nonfunctional characteristics such as maintainability, safety, availability etc. Mechanisms to represent hints, properties, or external constraints that a system should follow should be provided.



We examine whether the language itself has a sound mathematical basis. Having a sound mathematical basis for the specification language enables one to automatically check for inconsistencies in the specification itself. Given the advances in formal techniques and the ever increasing number of safety critical reactive systems being designed, we consider that support for formal analysis of the specification is important. We examine if the specification language has a strong mathematical formalism as its basis. A number of formalisms are available: Petri-nets, finite state machines, state diagram, temporal logic, process algebras, abstract data types, etc. A specification methodology may offer several of these formalisms as a choice for analyzing its specification models. In this module, we observe whether the specification-modeling methodology chose a language which has a formal basis.



Given the typical nontrivial complexity of the systems being designed today, executability of the specification is a big help in improving the comprehensibility and robustness of the specification. Executability also offers the ability to experiment with preliminary prototypes of the system under design which is useful to validate the specification against the requirements of the system.



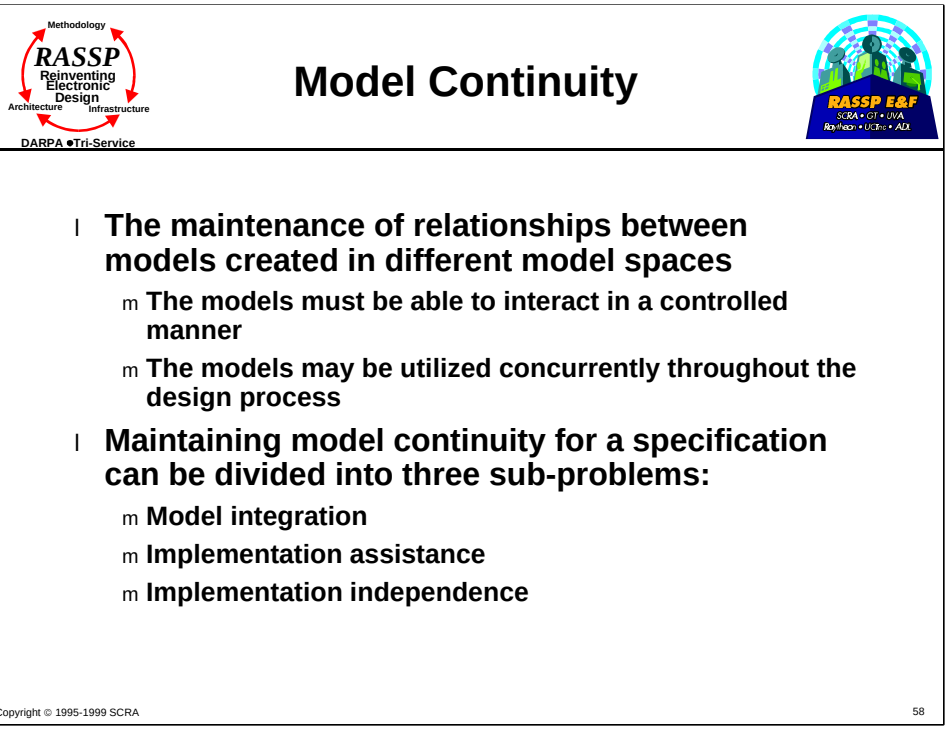
There are two dimensions of the **language** attribute. The conceptual models dimension determines the available conceptual models for describing reactive systems. The analysis-technique dimension determines the kind of support available for checking the specification consistency.

There are two dimensions along which **complexity control** should be supported: representational complexity and developmental complexity. Representational complexity deals with the clarity of the developed specification, whereas developmental complexity provides support for developing the specification in an organized and productive manner.

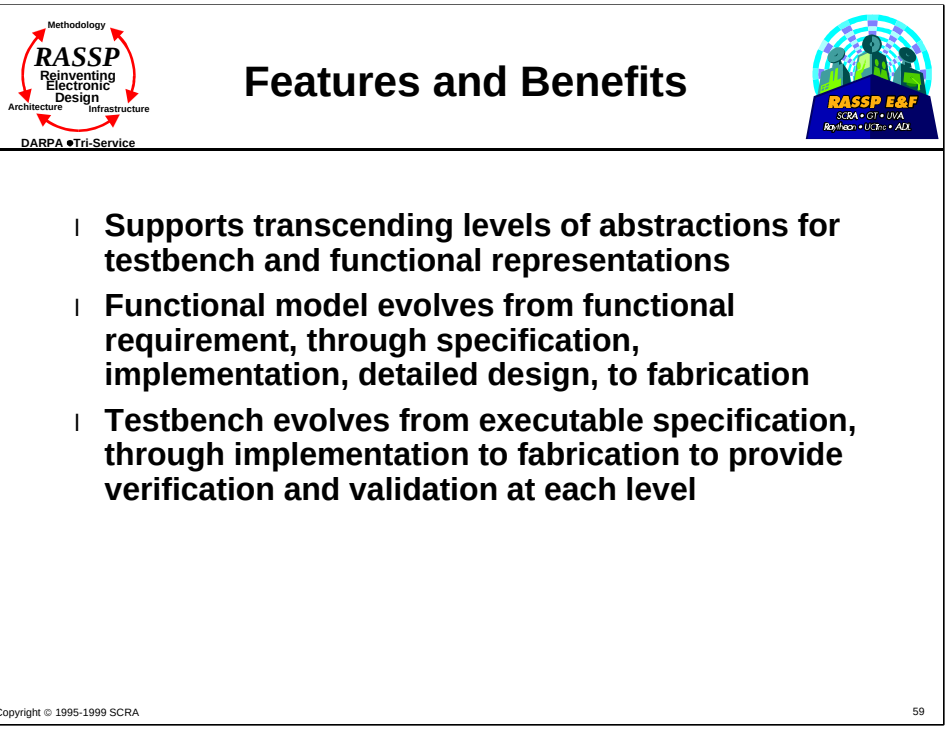
Model continuity has three dimensions: integrated modeling, implementation independence, and implementation assistance. Support along these three dimensions ensures that usefulness of the specification model is maintained beyond the specification modeling stage of a design.

A reactive system SMM must strongly support the attributes stated. Each of these attributes is described in the following sections. The effectiveness of the methodology can be identified by evaluating the strengths of each of these components.

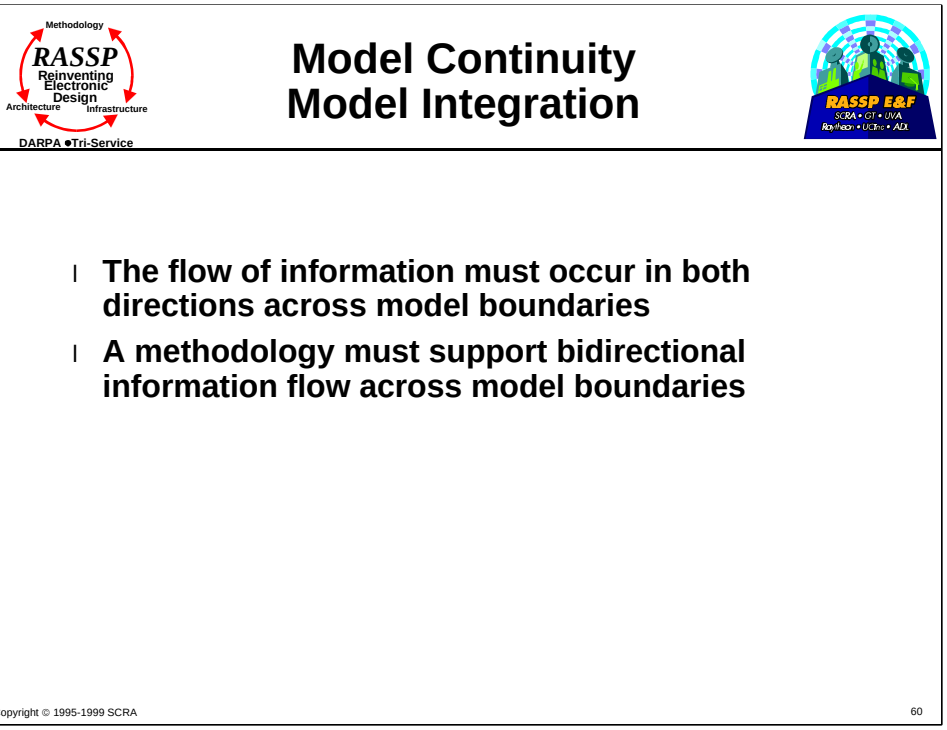
The following slides will discuss each of these attributes and their dimensions in greater detail.



Model continuity can be defined as the maintenance of relationships between models created in different model spaces such that the models can interact in a controlled manner and may be utilized concurrently throughout the design process. The problem of maintaining model continuity for a specification can be divided into the following three subproblems: model integration, implementation assistance, and implementation independence. Model integration addresses the challenge of making the specification model compatible with models developed during the design and implementation stages. Implementation assistance increases the usefulness of the specification by helping during the design and implementation stages. Implementation independence increases the useful life of the specification by not committing it to a particular design/implementation choice, thus avoiding restriction of creativity during the design process.



Design continuity allows a top-down design process, with increasing refinement and verification capability at each step.



By integrated modeling, we imply that the flow of information occurs in both directions across the model boundaries. This flow of information can occur during either integrated simulation or integrated analysis of both models. A methodology must support bidirectional information flow across model boundaries.

Model Continuity
Model Integration
Conformance

- **The methodology should provide support for checking conformance between the models at various levels of abstraction or in different domains**
- **Conformance checking may be viewed along two dimensions:**
  - **Vertical - involves validating conformance between models representing different levels of abstraction**
  - **Horizontal - involves validating conformance between models representing different modeling domains**
- **The need may exist for checking both horizontally and vertically**

The conformance attribute identifies how a methodology addresses the first subproblem of model continuity, namely: checking conformance among models developed. The methodology should provide either a simulation-based support or an analysis-based support for checking conformance between the models (or both).

We categorize conformance checking along two dimensions: vertical and horizontal. Vertical-conformance checking involves validating conformance between models representing different levels of abstraction. Horizontal-conformance checking involves validating conformance between models representing different modeling domains. To be effective, the methodology must provide support for checking conformance along both dimensions.

For example, one should be able to check the conformance between an algorithmic-level model and a logic-level of a system. This is an example of vertical-conformance checking. As an example of horizontal-conformance checking, one should also be able to check the conformance between a functional level behavioral model involving register-transfers and state-sequencers and a structural model involving ALUs, MUXs, registers etc.

Model Continuity
Model Integration
Model Interaction

- **Requirements of performing integrated modeling across different levels of abstraction and modeling domains:**
  - **A high degree of model interaction**
  - **Information flow among the models**
- **Model interaction involves identification of how a methodology addresses the following:**
  - **Implementation assistance**
    - **Maintaining visibility of the specification model during the implementation phase**
  - **Implementation independence**
    - **Incorporating relevant details obtained from the implementation phase back into the specification model**

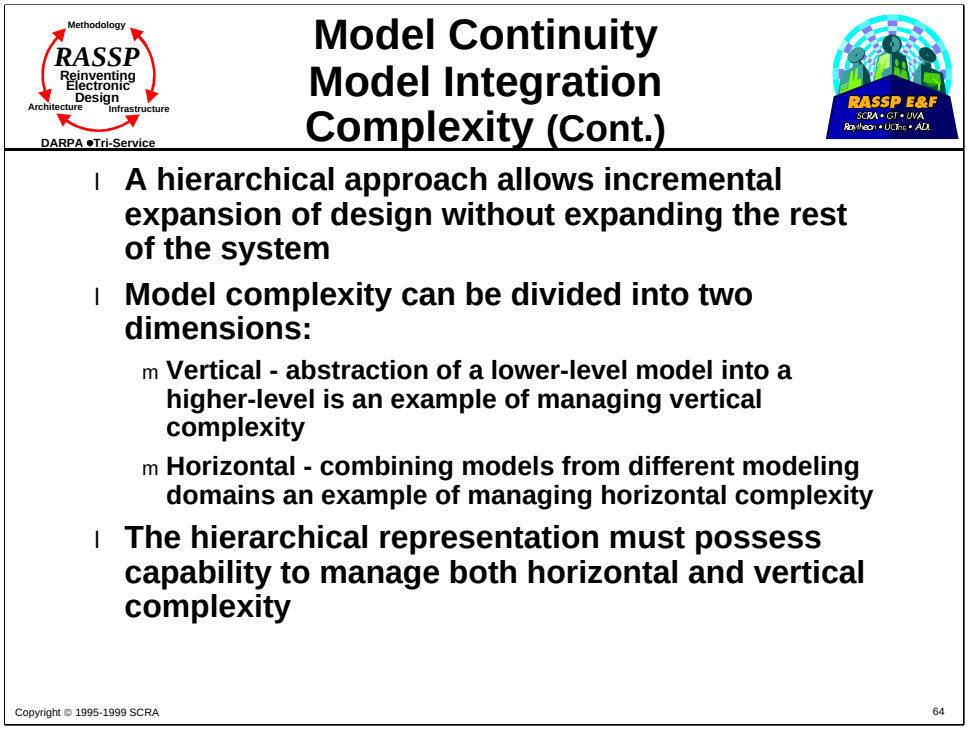
The interaction attribute identifies how a methodology addresses the second and third subproblems of model continuity, namely: maintaining visibility of the specification model during the implementation phase and incorporating relevant details obtained from the implementation phase back into the specification model. Supporting such a high degree of interaction and information flow among these models requires integrated modeling across different levels of abstraction and modeling domains.

Analogous to conformance checking, we categorize model interaction along two dimensions: vertical and horizontal. Vertical interaction occurs between models belonging to different levels of abstractions, whereas horizontal interaction occurs between models belonging to different domains of modeling. A methodology must provide mechanisms that support both vertical and horizontal model interactions.

Model Continuity
Model Integration
Complexity Control

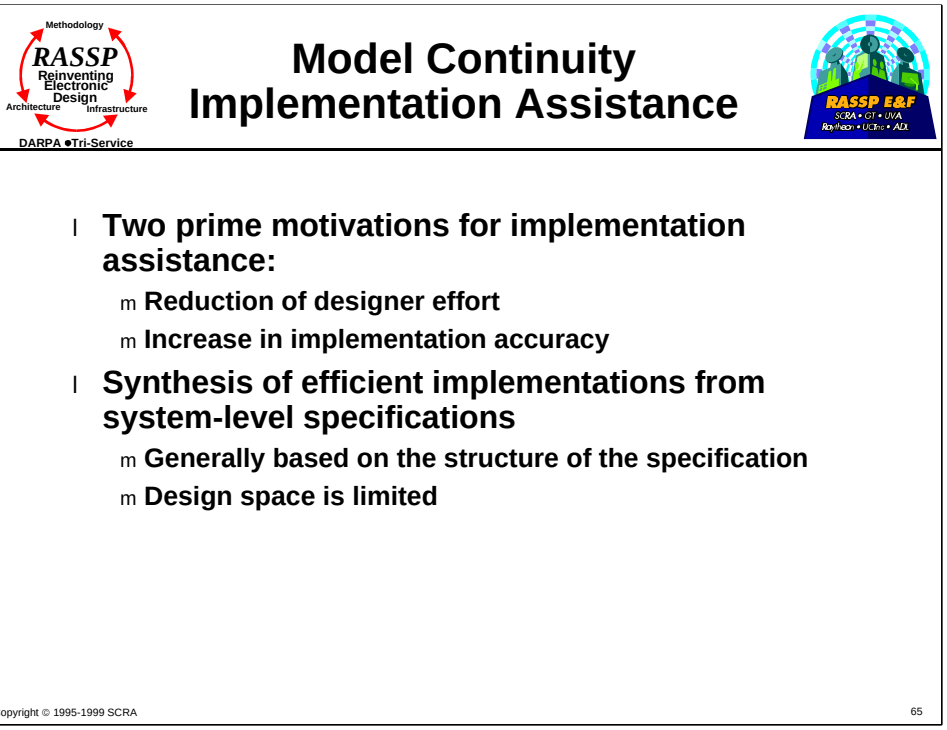
- **Required: a means to control complexity during the development and analysis of models throughout the design stages**
- **Support for this attribute is necessary for effective implementation of the conformance and interaction attributes**
- **Complexity control achieved by supporting a hierarchy of representations**

Complexity control is primarily achieved by supporting a hierarchy of representations. Support of hierarchy significantly reduces design time, as the designer is allowed to provide less detail in creating the original representation. For adding or synthesizing further information, he or she can then use automated or semi-automated design aids.

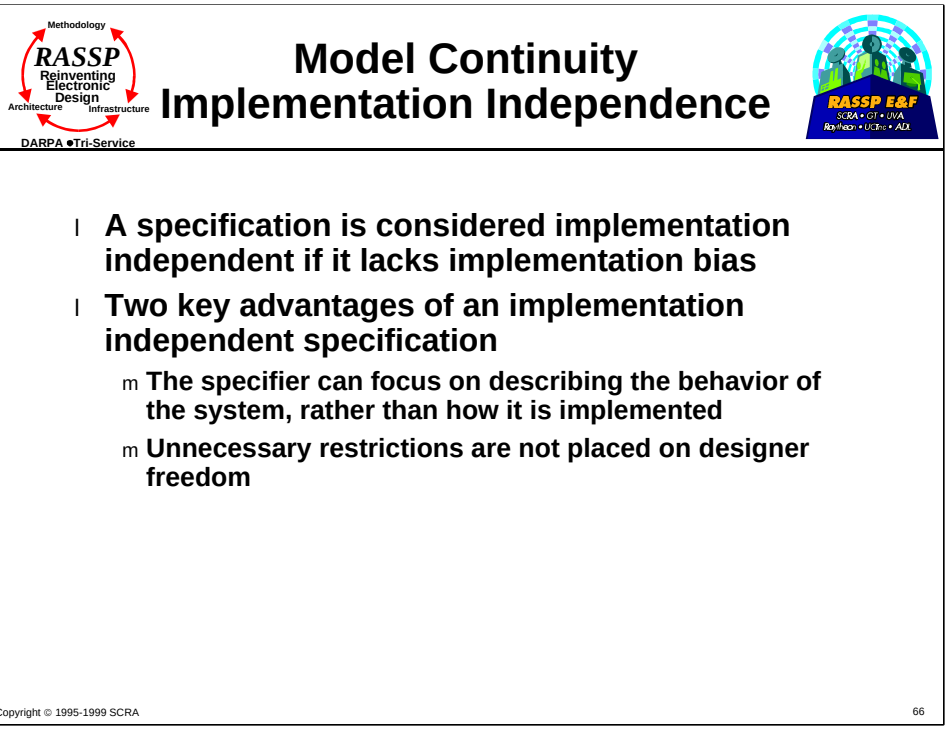


In addition to the addition or the synthesis of details, a hierarchical approach allows the designer to quickly identify what portion of the design should be expanded upon, without necessarily expanding the rest of the system. This incremental-expansion approach is of tremendous advantage when the expanded representation is radically different from the original representation. By enabling incremental modifications, a hierarchical representation improves designer comprehension of the effect of change on the original model.

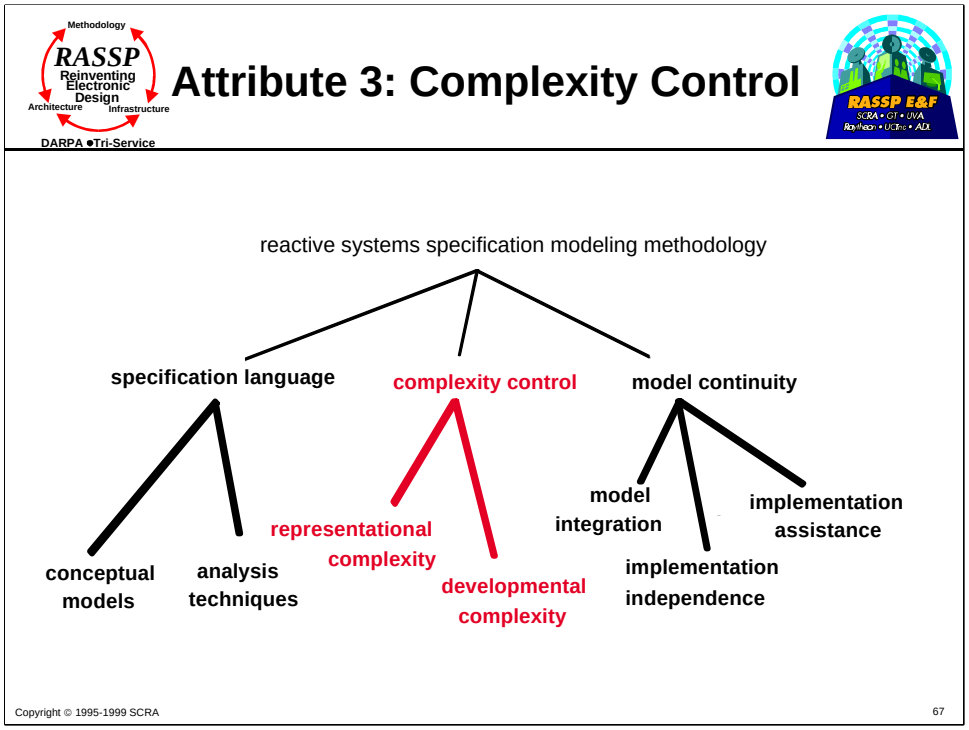
Similar to conformance checking and model interaction, model complexity can also be divided into two dimensions: vertical and horizontal. Abstraction of a lower-level model into a higher-level is an example of managing vertical complexity. Combination of models from different modeling domains into a unified representation is an example of managing horizontal complexity. The hierarchical representation must possess capability to manage both horizontal and vertical complexity.



The task of developing an implementation from a specification is reactive. As a result, there has been a significant research in the automated synthesis of implementations from specifications. There are two prime motivations for implementation assistance: reduction of designer effort and increase in implementation accuracy. By supporting automated/semi-automated techniques for the synthesis of a design/implementation, there is a significant reduction in required designer effort. Also, an automated technique avoids human errors that can be introduced otherwise during the manual design process. Synthesis of efficient implementations from system-level specifications is still immature. Another limitation of current synthesis techniques is that they are generally based on the structure of the specification, thus limiting design space and therefore producing less optimal solutions.



A specification has an implementation bias if it specifies externally unobservable properties of the system it specifies. A specification is therefore considered implementation independent if it lacks implementation bias. While evaluating a specification methodology, we examine how well it supports implementation independence. There are two key advantages of an implementation independent specification. First, it allows the specifier to focus on describing the behavior of the system, rather than how it is implemented. Second, it avoids placing unnecessary restrictions on designer freedom.



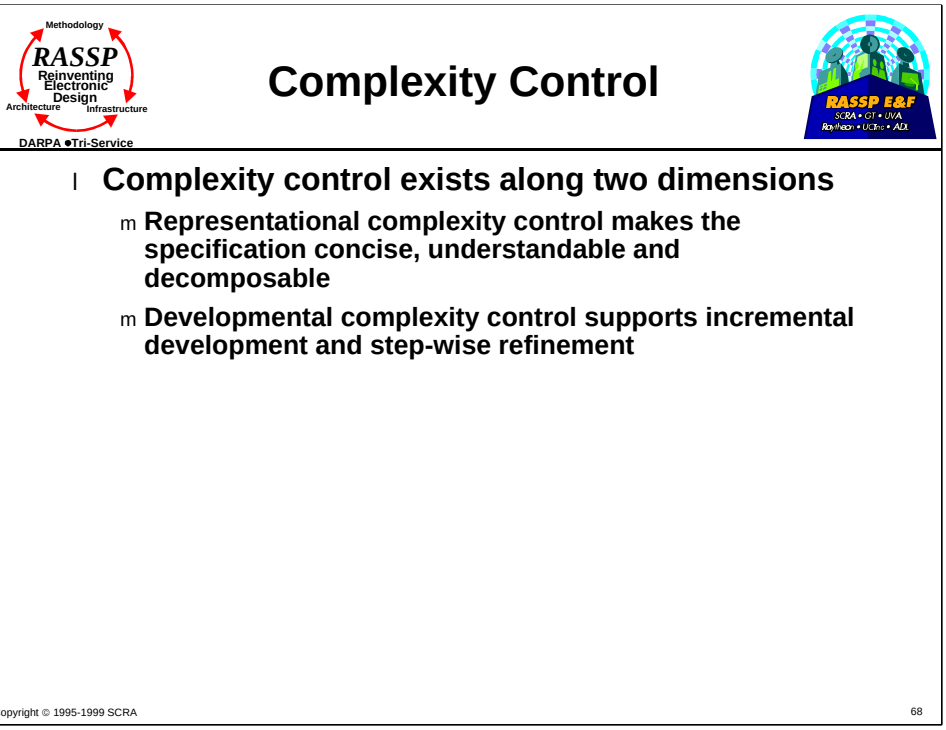
There are two dimensions of the **language** attribute. The conceptual-models dimension determines the available conceptual models for describing reactive systems. The analysis-technique dimension determines the kind of support available for checking the specification consistency.

There are two dimensions along which **complexity control** should be supported: representational complexity and developmental complexity. Representational complexity deals with the clarity of the developed specification, whereas developmental complexity provides support for developing the specification in an organized and productive manner.

Model-continuity has three dimensions: integrated modeling, implementation independence, and implementation assistance. Support along these three dimensions ensures that usefulness of the specification model is maintained beyond the specification modeling stage of a design.

A reactive system SMM must strongly support the attributes stated. Each of these attributes is described in the following sections. The effectiveness of the methodology can be identified by evaluating the strengths of each of these components.

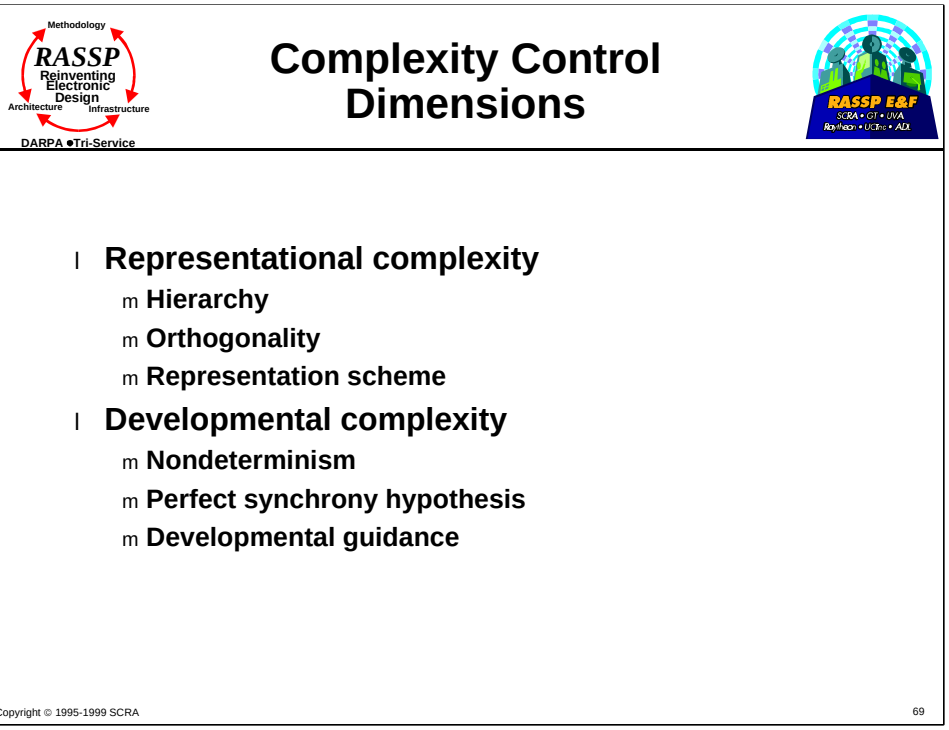
The following slides will discuss each of these attributes and their dimensions in greater detail.



Support for complexity control in a specification-modeling methodology can exist along two dimensions. The first dimension is the representational complexity, which makes the specification itself concise, understandable, and decomposable into simpler components. The second dimension is developmental complexity, which supports the development of the specification in an incremental, step-wise refined manner.

Support for representational complexity is usually dependent on the specification language chosen. We separate the complexity control aspect of a specification language from its support for expressing reactive system characteristics.

One of the main requirements of a design methodology is to be able to control the complexity of the design process. Support for complexity control in a specification-modeling methodology can exist along two dimensions. The first dimension is the representational complexity, which makes the specification itself concise, understandable, and decomposable into simpler components. The second dimension is developmental complexity, which supports the development of the specification in an incremental, step-wise refined manner.

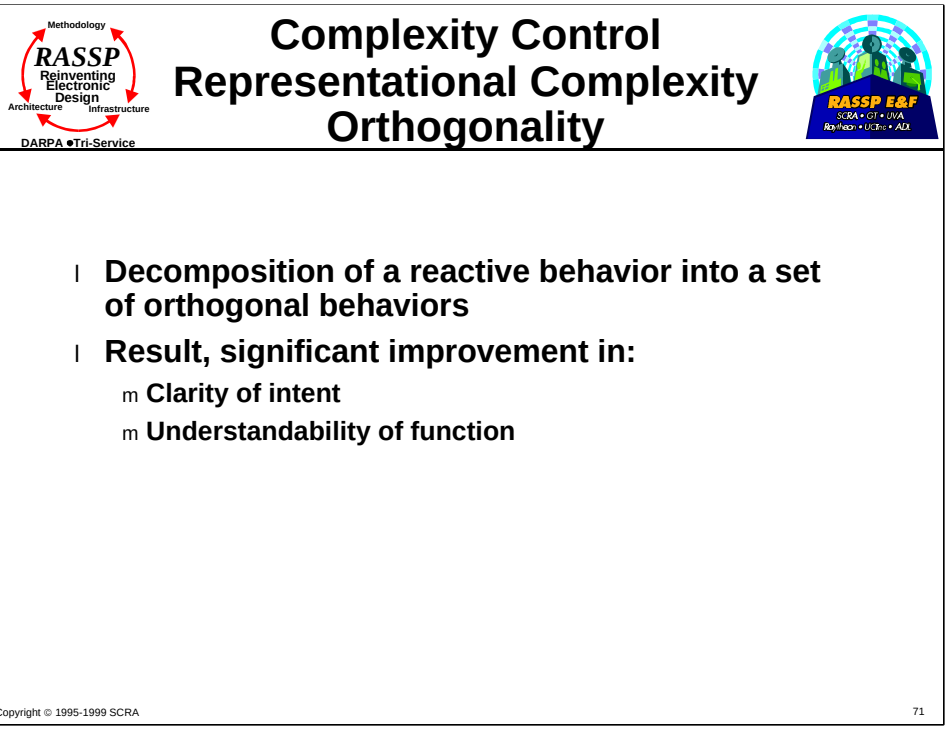


We will discuss each of the elements of the two dimensions of complexity control in the following charts.

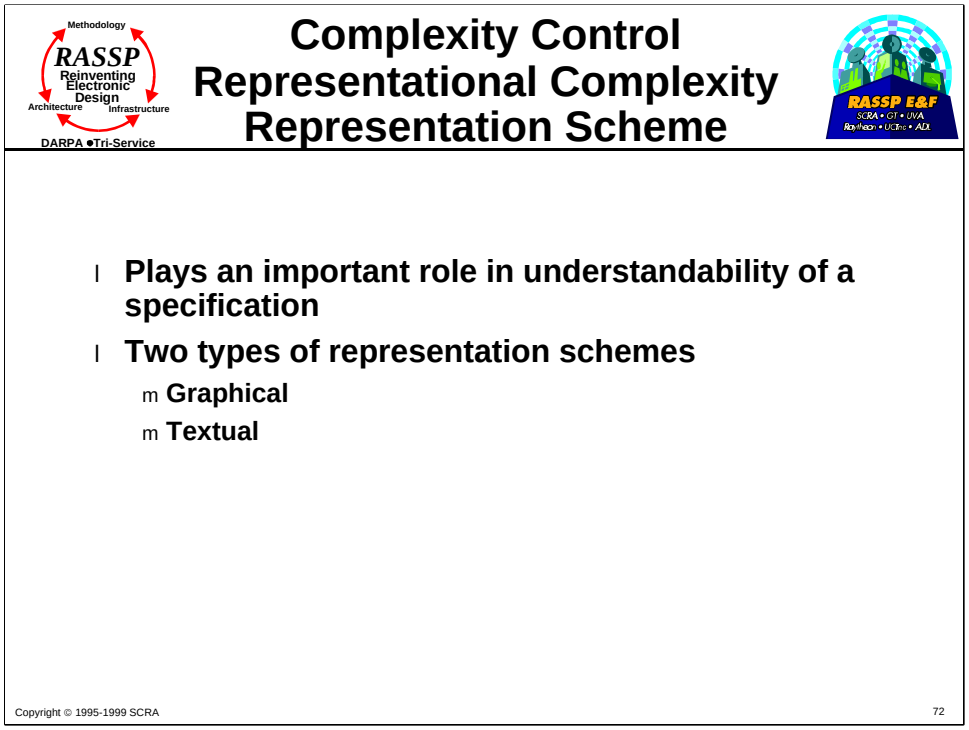
Complexity Control Representational Complexity Hierarchy

- **An important tool in controlling complexity**
- **Group similar elements together to create a new element**
  - **The new element represents the group of similar elements**
- **Result**
  - **Introduction of common behavior**
  - **Support for multiple levels of abstraction**

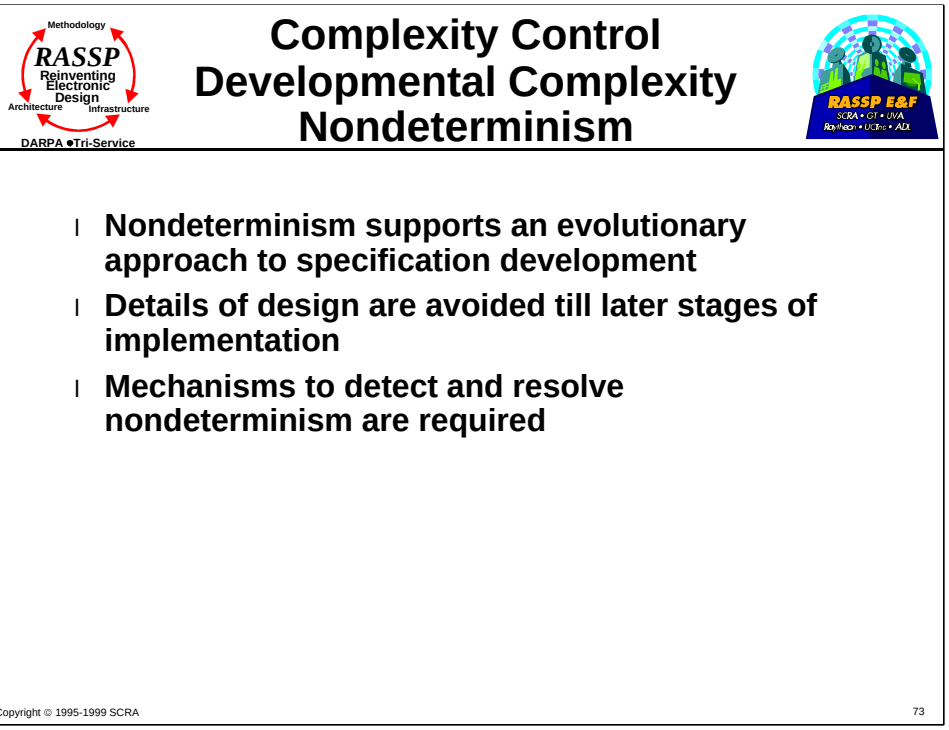
The notion of hierarchy is an important tool in controlling complexity. The basic idea in hierarchy is to group similar elements together and to create a new element that represents this group of similar elements. By introducing the common behavior in this way, multiple levels of abstractions can be supported.



A reactive behavior can often be decomposed into a set of orthogonal behaviors. By supporting such decomposition in the representation, significant improvement in clarity and understandability can be attained.

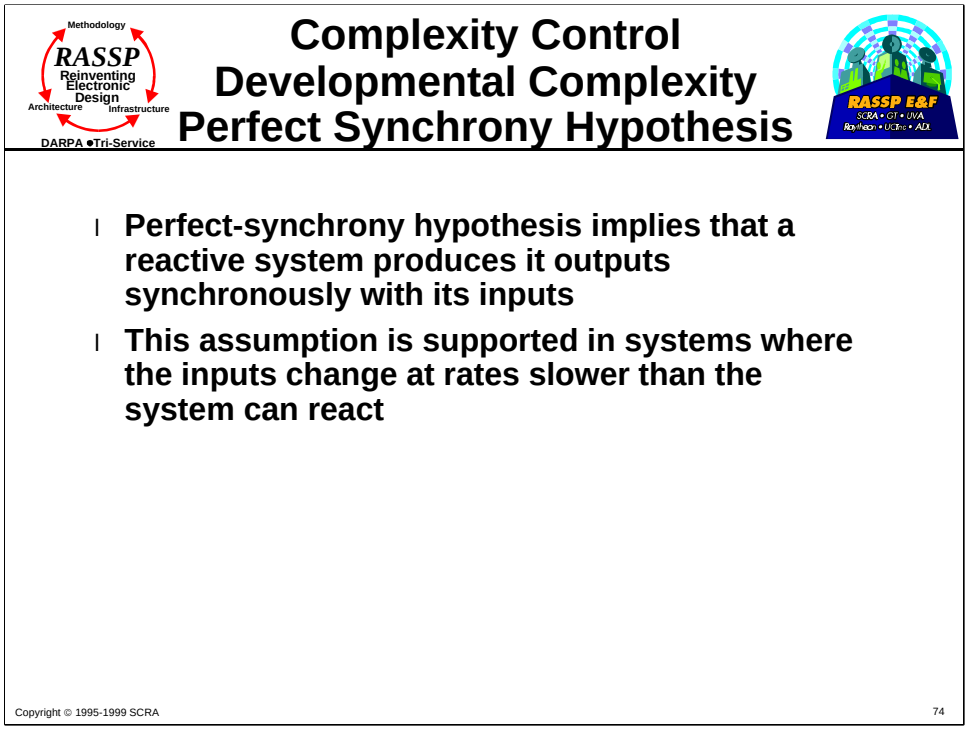


The representation scheme plays an important role in the understandability of a specification. We make the distinction between graphical and textual representation schemes. By graphical scheme, we imply visual formalisms where both syntactic and semantic interpretations are assigned to graphical entities. For example, Statecharts, Petri-nets etc. are such visual formalisms. In our opinion, graphical representation schemes are preferable to a textual ones since the former allow the specifier to visualize the system behavior more effectively, especially during execution of the specification. For many textual approaches, however, tools exist today that transform the textual approach into a graphical approach.



In addition to the representation of system behavior, a specification methodology must also support the evolution of the specification model from initial conceptualization of system requirements.

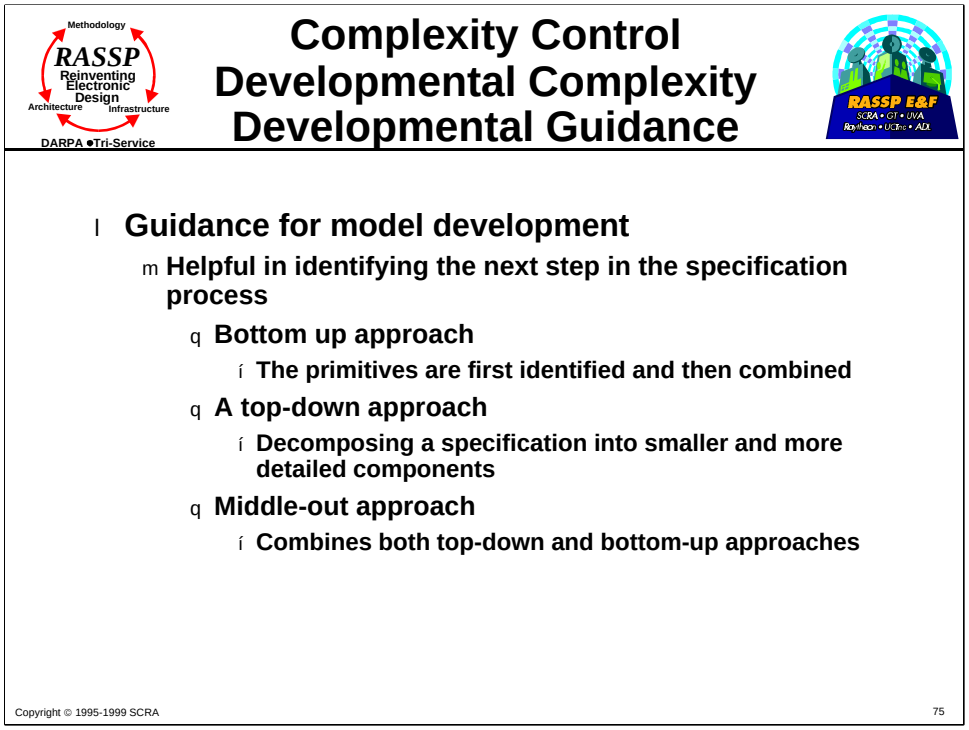
By incorporating nondeterminism in a controlled manner, the specification can leave details to the implementation and final stages. For example, in a typical producer-consumer type system, if requests for both element insertions and element deletions from a buffer arrive simultaneously, the specification may non-deterministically select either operation to be executed first. The commitment to an actual choice in such a scenario is deferred to later design stages. Nondeterminism thus supports an evolutionary approach to specification development. In addition to the incorporation of nondeterminism, a specification methodology should also provide mechanisms to detect and resolve nondeterminism. It is often difficult to detect nondeterminism in specifications of reactive systems, given the reactive interrelationships of the behaviors of the system components. If left undetected and consequently unresolved, nondeterminism is a potential source of ambiguity.



Perfect-synchrony hypothesis implies that a reactive system produces it outputs synchronously with its inputs. In other words, the outputs are produced instantaneously after the inputs occur. This assumption of perfect-synchrony is borne out in many cases, especially if the inputs change at rates slower than the system can react. For example, in a clocked system, if the clock cycle is long enough, the system gets a chance to stabilize its output values before the next clock event occurs.

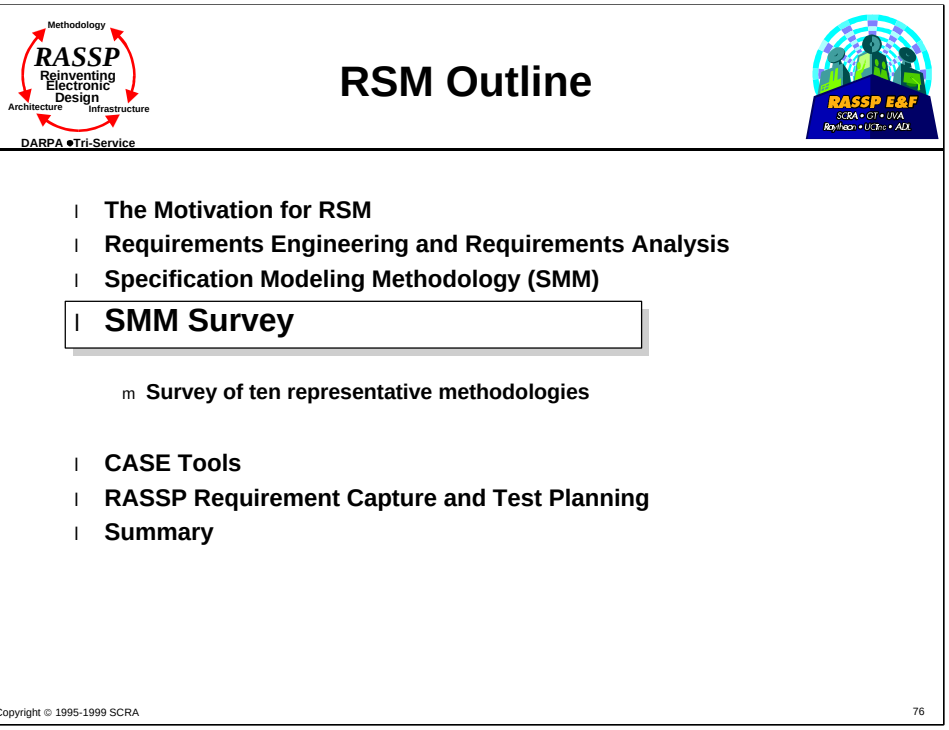
One of the main requirements of a design methodology is to be able to control the complexity of the design process. Support for complexity control in a specification-modeling methodology can exist along two dimensions. The first dimension is the representational complexity, which makes the specification itself concise, understandable, and decomposable into simpler components. The second dimension is developmental complexity, which supports the development of the specification in an incremental, step-wise refined manner.

Perfect-synchrony assumption makes the specification concise, composable with other specifications, and in general lend the specification to a number of elegant analysis techniques. However, the assumption of perfect synchrony may not be valid at all levels of abstractions especially if the reaction of the system is complicated. For example, if the reaction is a lengthy computation, clearly assumption of perfect synchrony will be violated, as the input may change before the computation is completed.

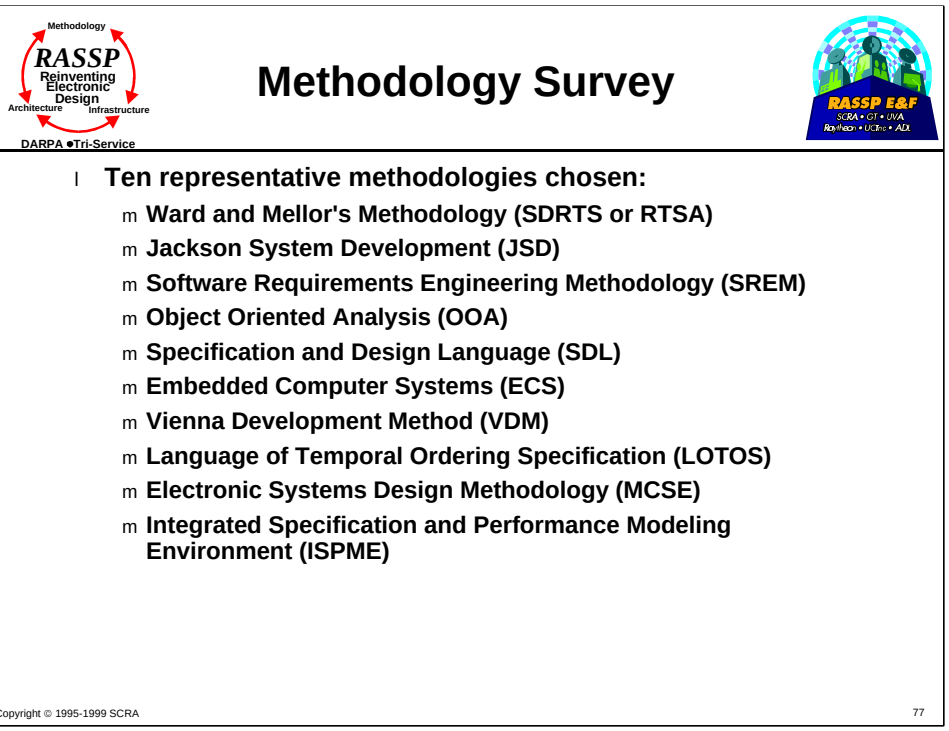


Guidance for model development is helpful in identifying the next step in the process of specifying a system. One may do it bottom up, where the primitives are first identified and then combined. Another guidance is in the form of a top-down approach, where a specification is decomposing into smaller and more detailed components. Yet another approach is called the middle-out approach, which combines both top-down and bottom-up approaches.

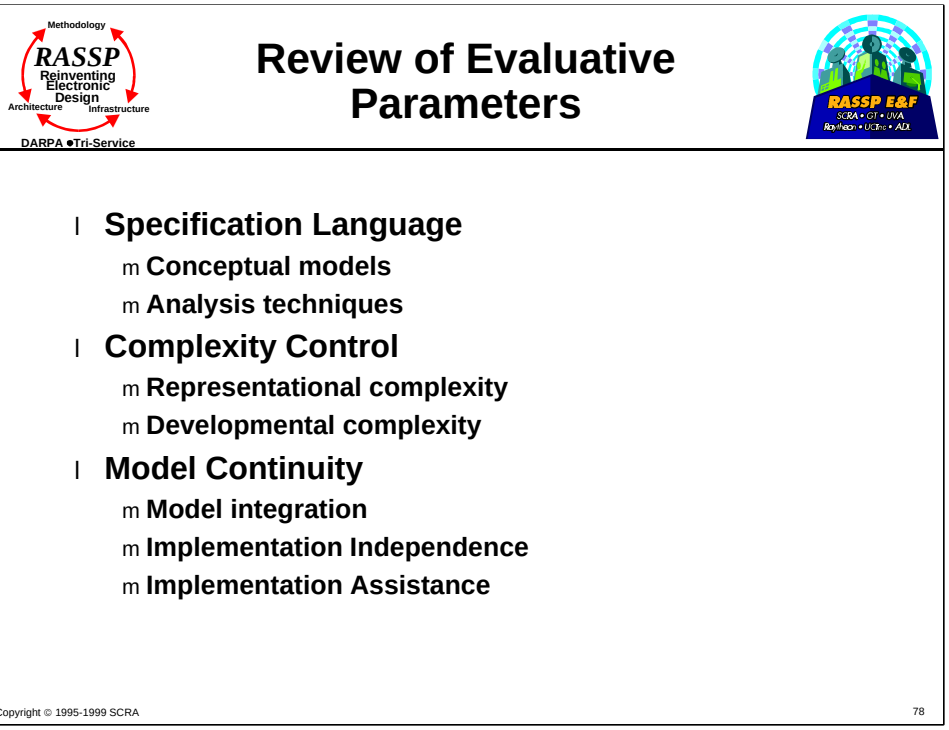
Typically, development style is a matter of choice and not strictly enforced. In some methodologies, however, this can be enforced. Enforcing the style may interfere with designer creativity.



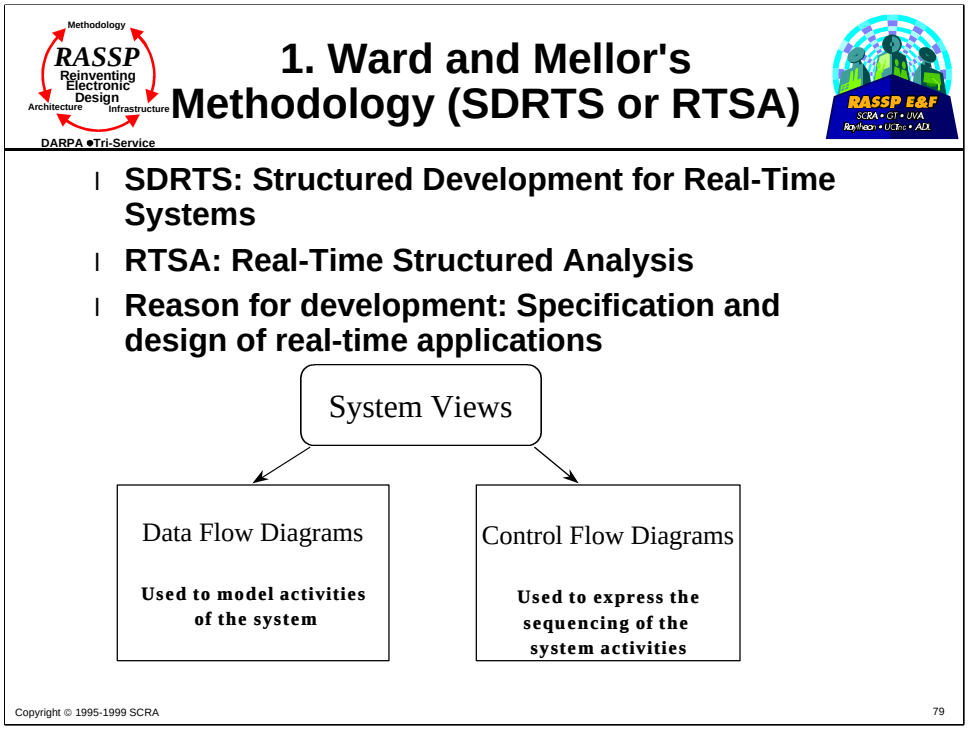
We now introduce several proposed methodologies in recent literature and provide a relative evaluation of their suitability in a RSM methodology.



Ten methodologies selected are shown above. References contain details about each of the methodologies, above and beyond what could be included within this module.

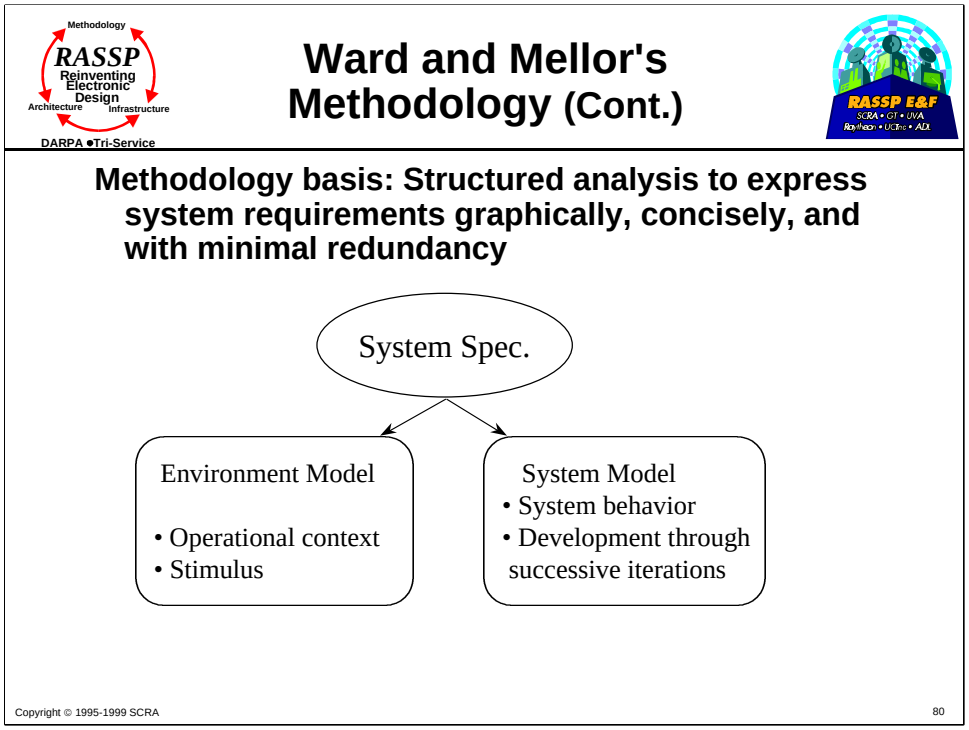


We review the main metrics and parameters that will assist us in evaluating the several proposed RSM methodologies.

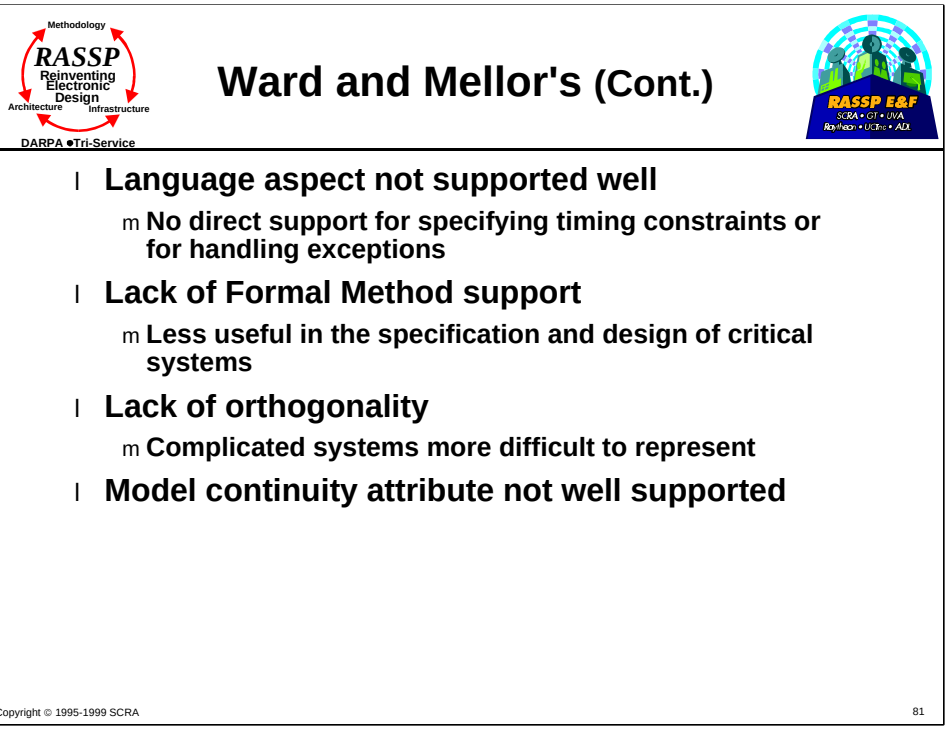


Ward and Mellor's methodology called *S*tructured *D*evelopment for *R*eal-*T*ime *S*ystems (SDRTS), was developed for the specification and design of real-time applications. Since the methodology is an extension of the Structured Analysis methodology for real-time systems, it is also called *R*eal-*T*ime *S*tructured *A*nalysis (RTSA).

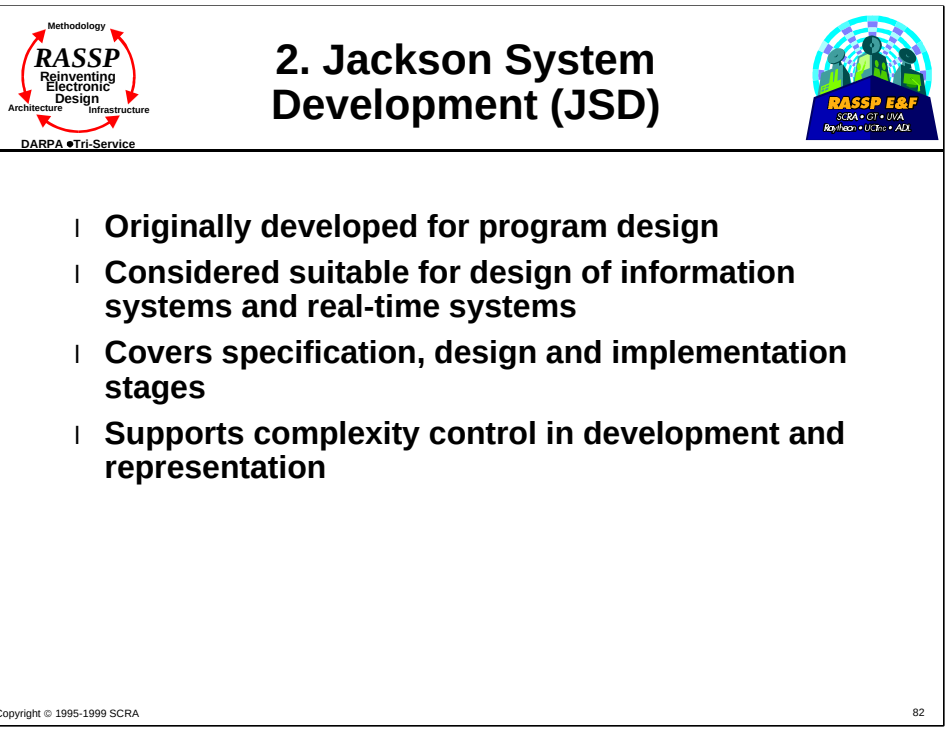
There are two system-views adopted by RTSA: Data-Flow Diagrams (DFD) and Control-Flow Diagrams (CFD). DFD is used to model the activities in the system, whereas the CFD is used to express the sequencing of these activities. The CFD itself is defined in terms of a finite-state model such as FSM or decision table.



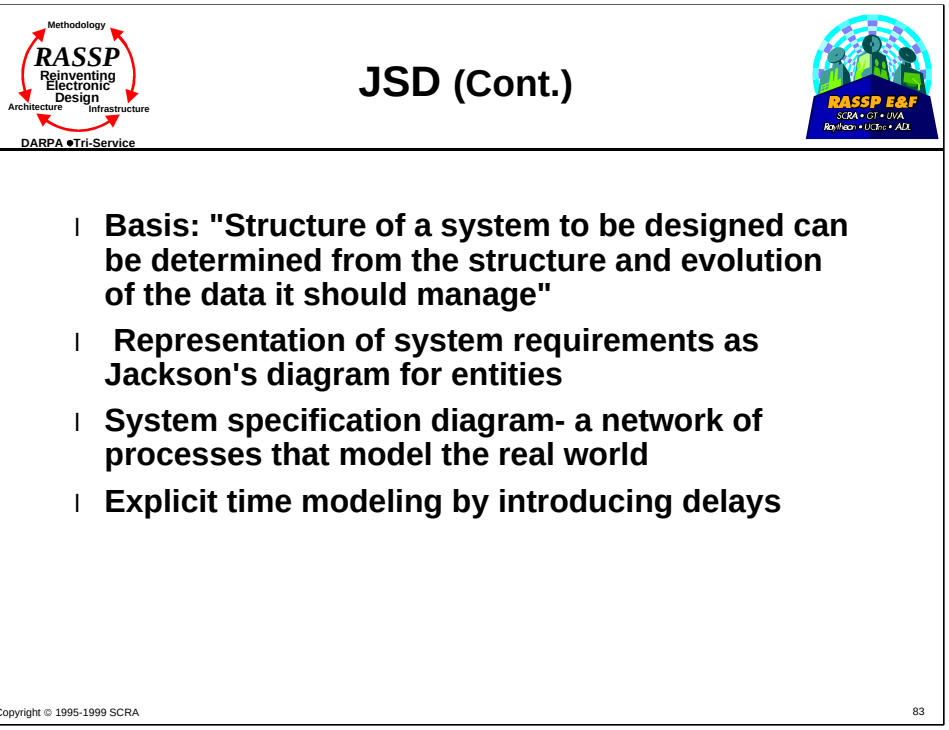
The methodology is based on structured analysis, one of the earlier approaches to express system requirements graphically, concisely, and minimally redundant manner. To develop a specification of the system, two models are created: the environment model and the system model. The environment model describes the operational context in which the system operates and the events to which the system reacts. The system model, which expresses the system's behavior, is then developed through successive refinements.



The language attribute is not supported well, since several aspects of reactive system behavior cannot be modeled conveniently. For example, there is no direct support for specifying timing constraints or handling exceptions elegantly. A lack of any formal method support makes the methodology less useful in specification and design of critical systems. Lack of orthogonality makes it harder to conveniently represent and understand the behavior of complicated systems. The model-continuity attribute is also not well supported.

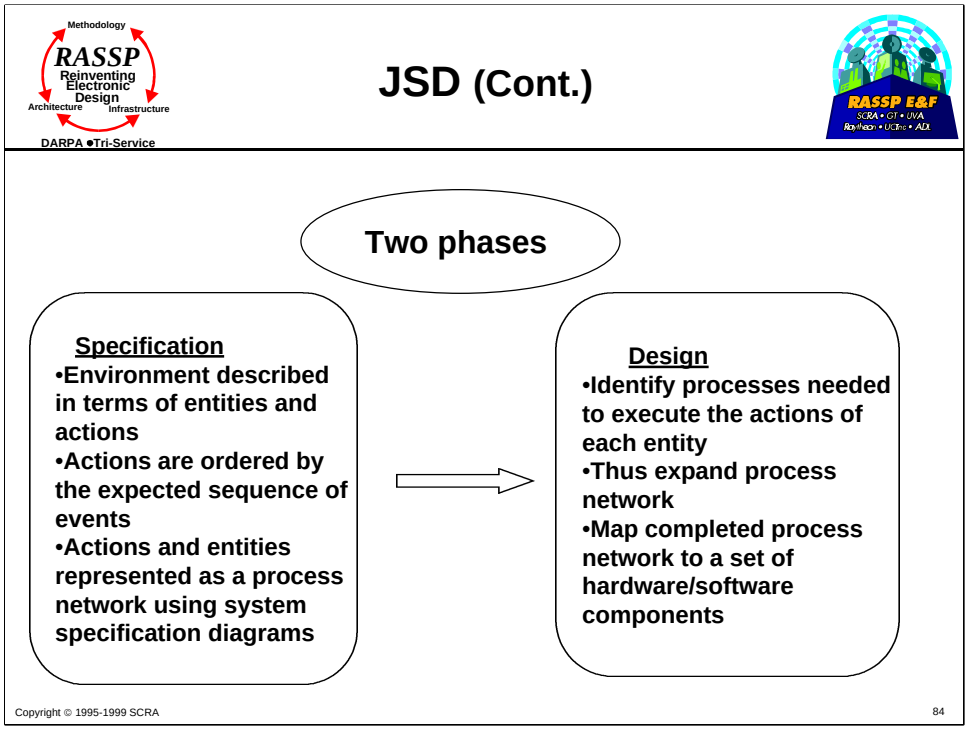


Jackson System Development methodology, originally developed for program design, is considered suitable for the design of information systems and real-time systems. In its current stage, the methodology covers specification, design and implementation stages.



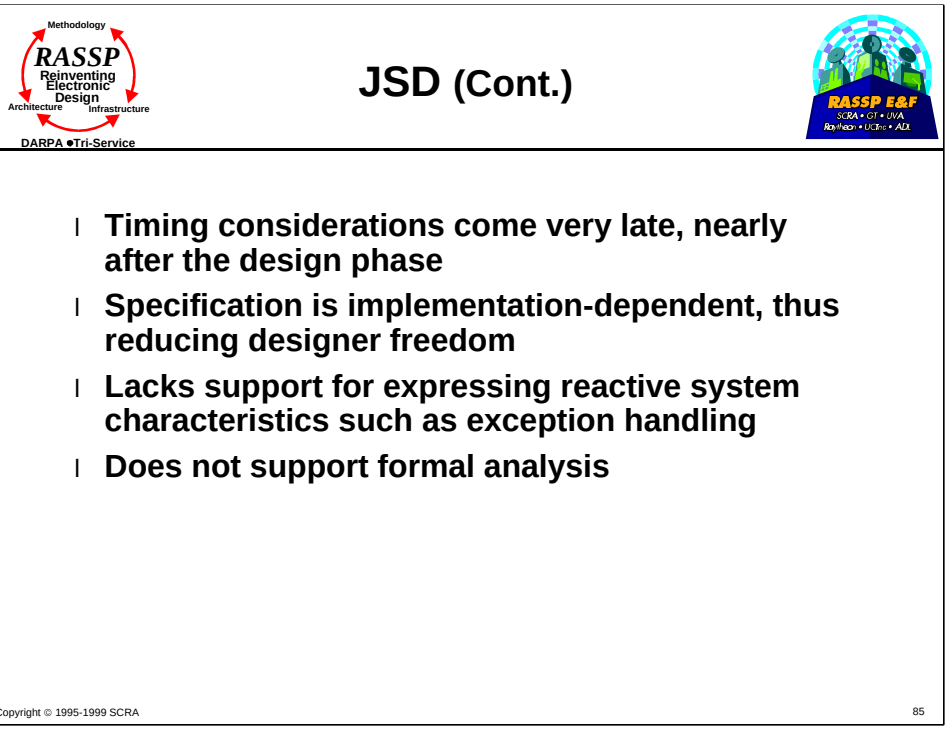
The methodology is based on entity modeling. The system requirements are expressed as Jackson's diagram for entities. The diagram presents a time-ordered specification of the actions performed on or by an entity. A system specification diagram is also created, which is a network of processes that model the real world. A process communicates with others by transmitting data and state information. It is also possible to model time explicitly by introducing explicit delays.

The basic modeling philosophy is that the structure of a system to be designed can be determined from the structure and evolution of data it must manage.

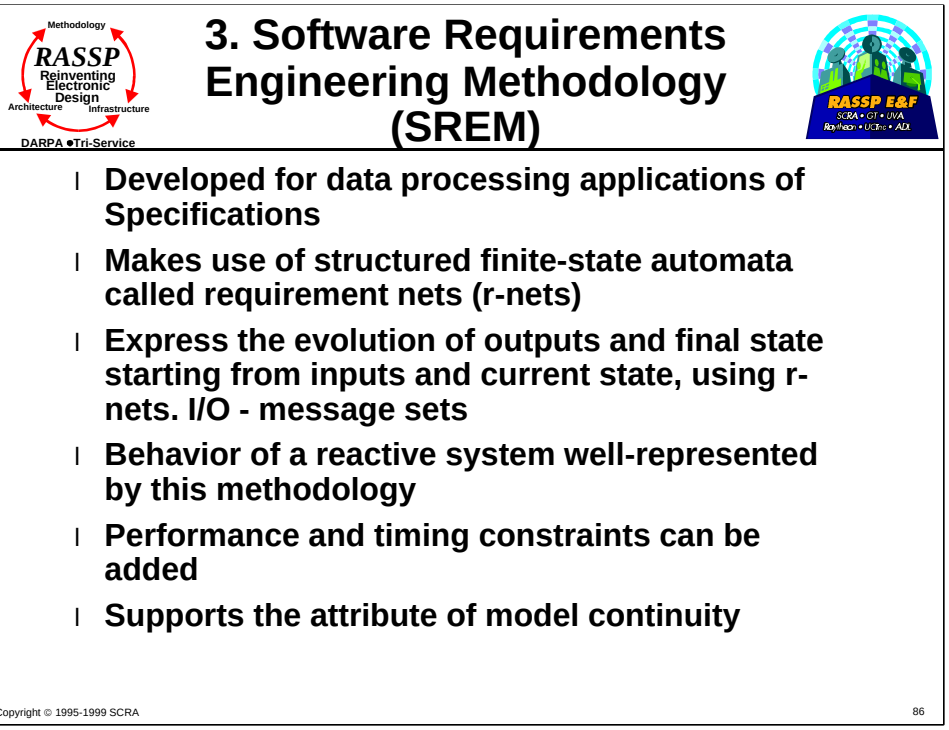


The methodology consists of two phases: specification and design. In the specification phase, the environment is described in terms of entities (real world objects the system needs to use) and actions (real world events that affect the entities). These actions are ordered by their expected sequence of occurrences and represented with Jackson diagrams. The actions and entities are then represented as a process network using system specification diagrams. The connection between these processes and the real world are defined. The creation of the initial process network can be seen as the end of specification phase.

During the design phase that follows the specification phase, the process network is successively elaborated by identifying further processes that are needed to execute the actions associated with the entities described in the Jackson Structure Diagram. The completed process network represents the final design which is then mapped to a set of hardware/software components.



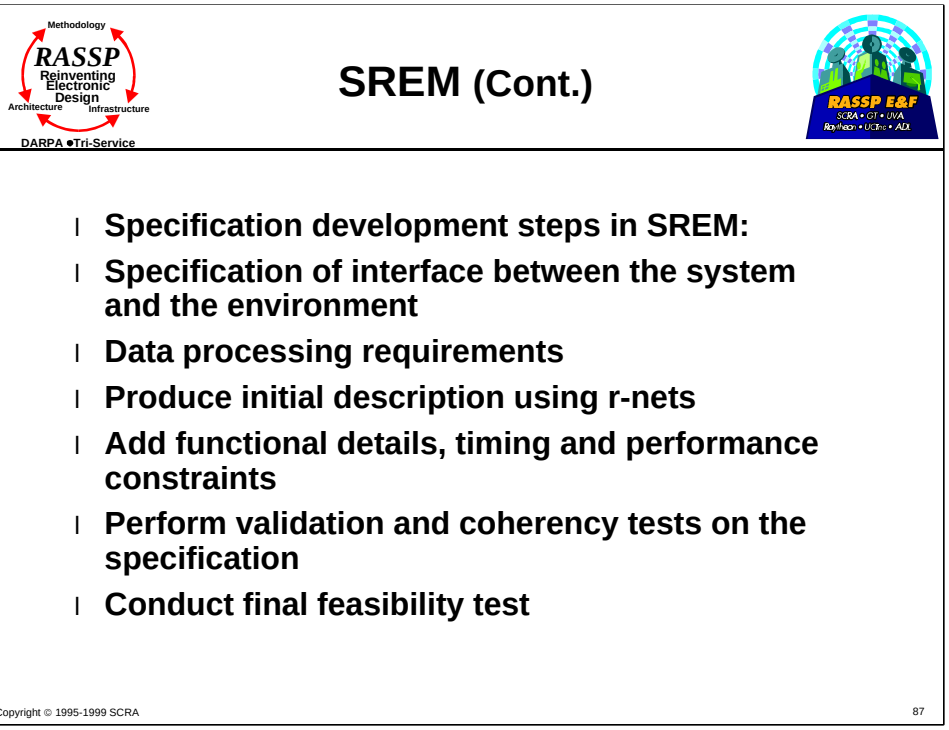
The JSD methodology supports model continuity by carrying the specification through further well defined design steps. However, timing considerations come very late, almost after the design phase. The specification is implementation dependent, since it is closely tied to an implementation, thereby reducing designer freedom. The methodology lacks support for expressing several reactive-system characteristics such as exception handling. Neither does it support any formal analysis or well defined execution semantics.



Software Requirements Engineering Methodology (SREM) was developed for creation, checking and validation of specifications of real-time and distributed applications for data processing.

The SREM method is useful for specification making use of structured finite-state automata called requirement nets (r-nets). R-nets express the evolution of outputs and final state starting from inputs and current state. Both inputs and outputs are structured as sets of messages, communicated by an interface connected to the environment.

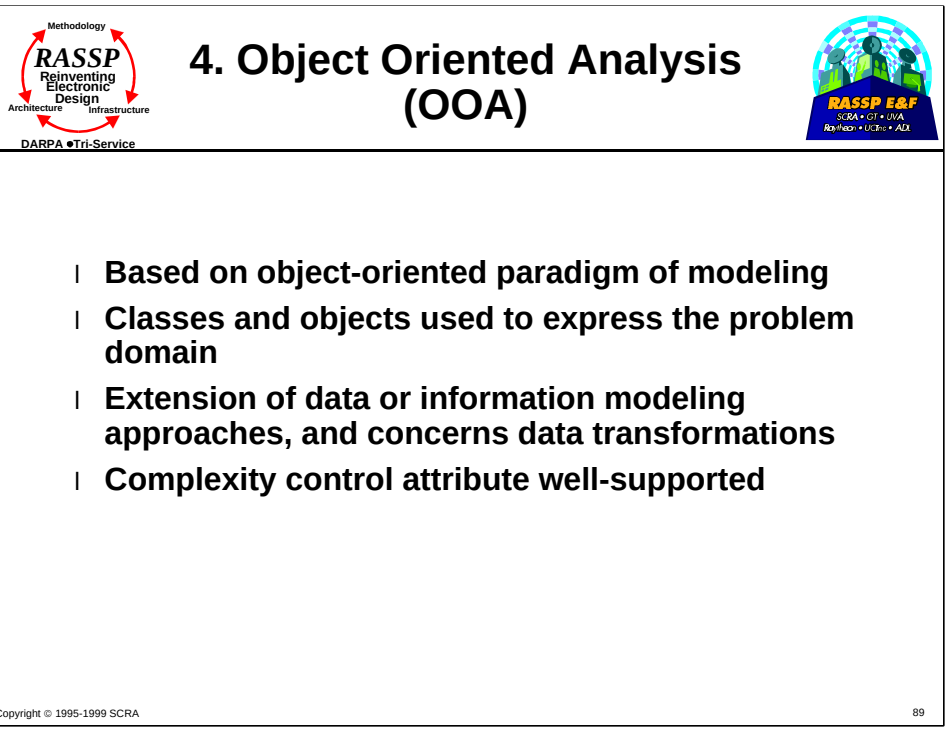
The behavior of a reactive system is well-represented by this methodology. The addition of performance specifications and timing constraints are also beneficial.



To develop a specification, the interface between the system and the environment and the data-processing requirements are specified. The initial description is produced using r-nets. Functional details, timing and performance constraints are then added. Next, validation and coherency checks are performed on the specification. A final feasibility study is conducted to guarantee that the specification will result in a feasible solution.

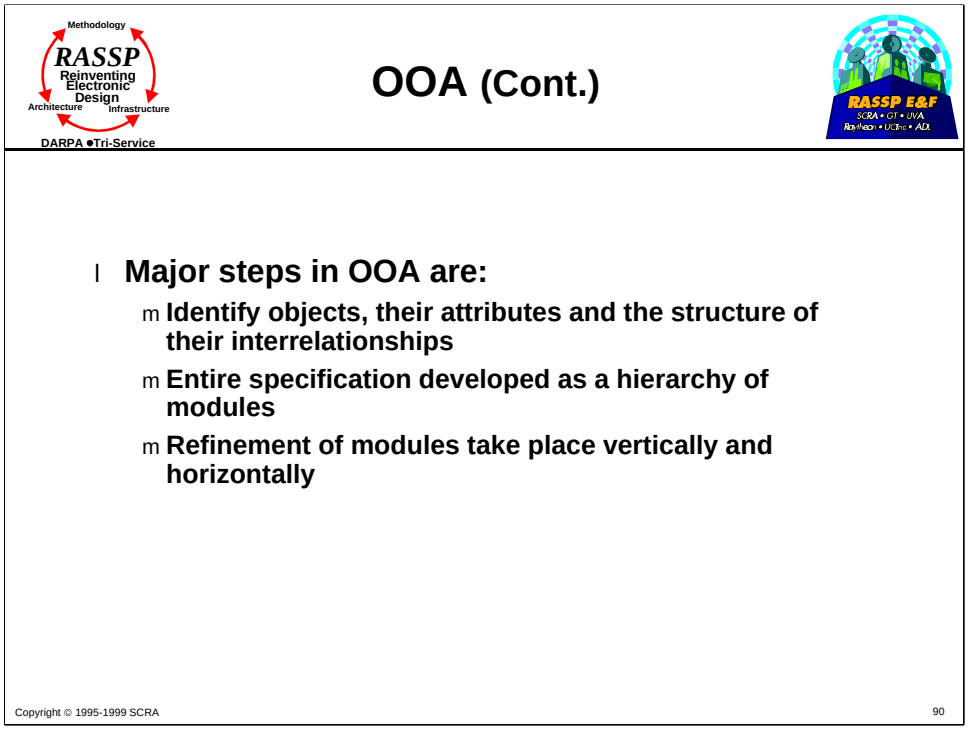
SREM (Cont.)

- **Does not support hierarchical decomposition of the specification**
  - **Task of specification requires too much detail**
- **Specification and implementation are tied too close to one another, thus limiting model continuity support**

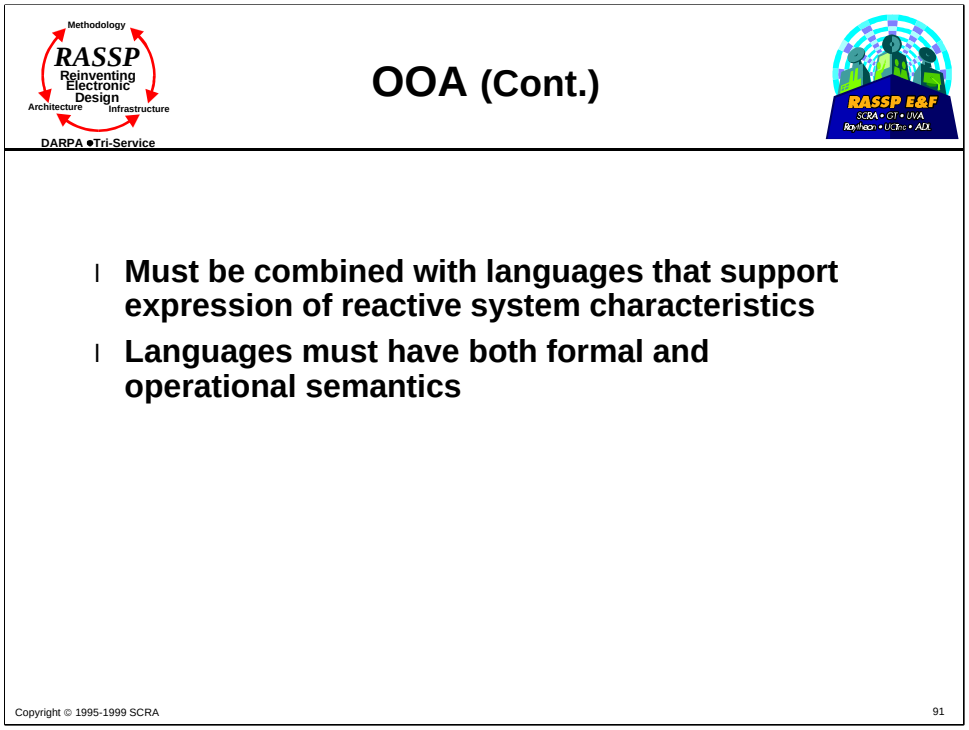


OOA stands for Object Oriented Analysis, and is based on the object-oriented paradigm of modeling. The world is modeled in terms of classes and objects that are suitable to express the problem domain. Different schemes have been suggested for OOA [CY90, Pnu86, SM88], they differ mostly in terms of notations and heuristics.

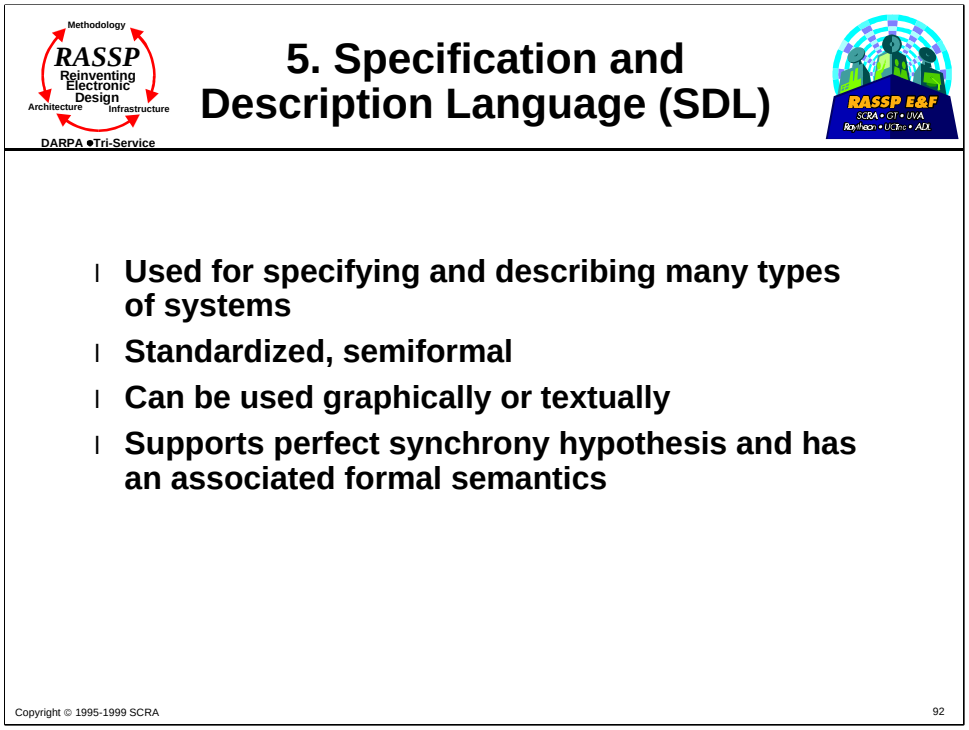
OOA can be seen as an extensions of data or information modeling approaches. The latter approaches focus solely on data. In addition to modeling data, OOA also concerns data transformations.



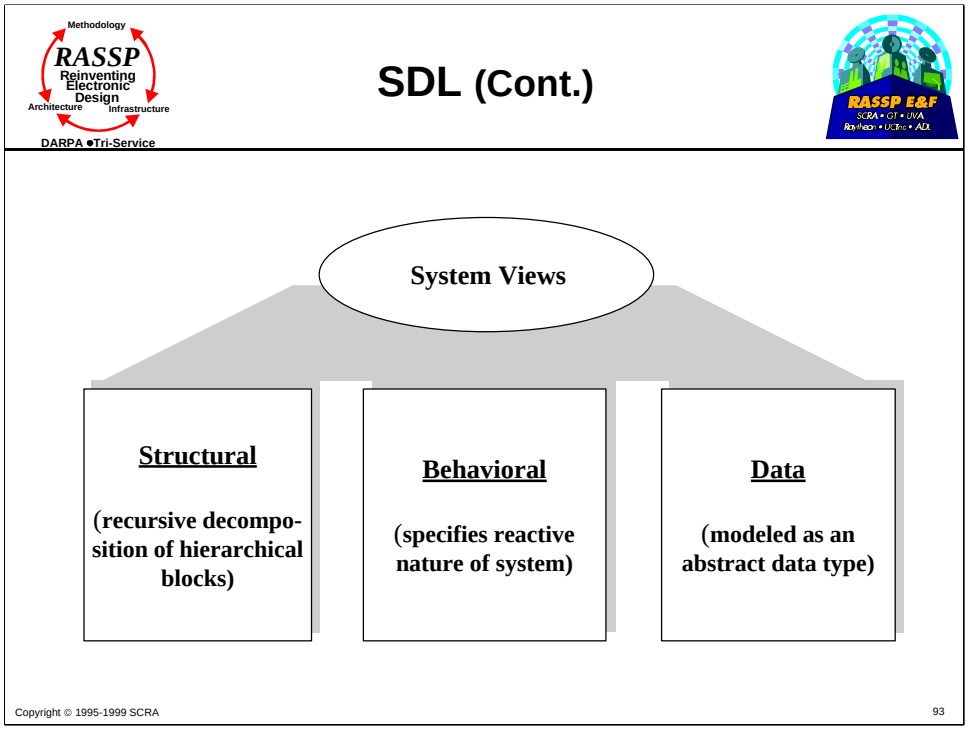
The major steps in OOA consist of identifying objects, their attributes, and the structure of their interrelationships. The entire specification is developed as a hierarchy of modules, where each module is successively refined both horizontally and vertically into further modules. The vertical refinement adds further properties to a module, whereas the horizontal refinement identifies a set of loosely-coupled, strongly-cohesive interacting sub-modules that define the behavior of the original module. The implementation of these modules is, however, not considered at the specification stage.



To exploit the strength of OOA, it should be combined with languages that support expression of reactive system characteristics and have both formal and operational semantics. For application to the domain of reactive systems, we find approaches that combine languages suitable for expressing reactive systems with object-oriented design principles.

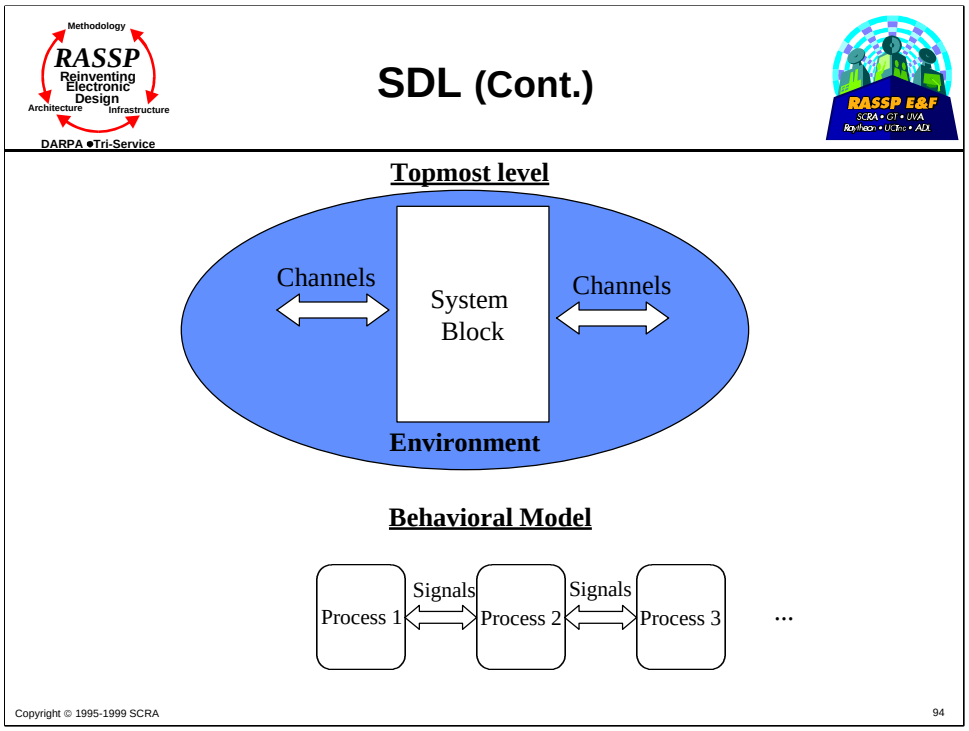


Specification and Description Language is a design methodology that has been standardized by CCITT, and is used for specifying and describing many types of systems. SDL is standardized, semiformal, and can be used both as a graphical or a textual language. SDL is primarily used for telecommunication systems .

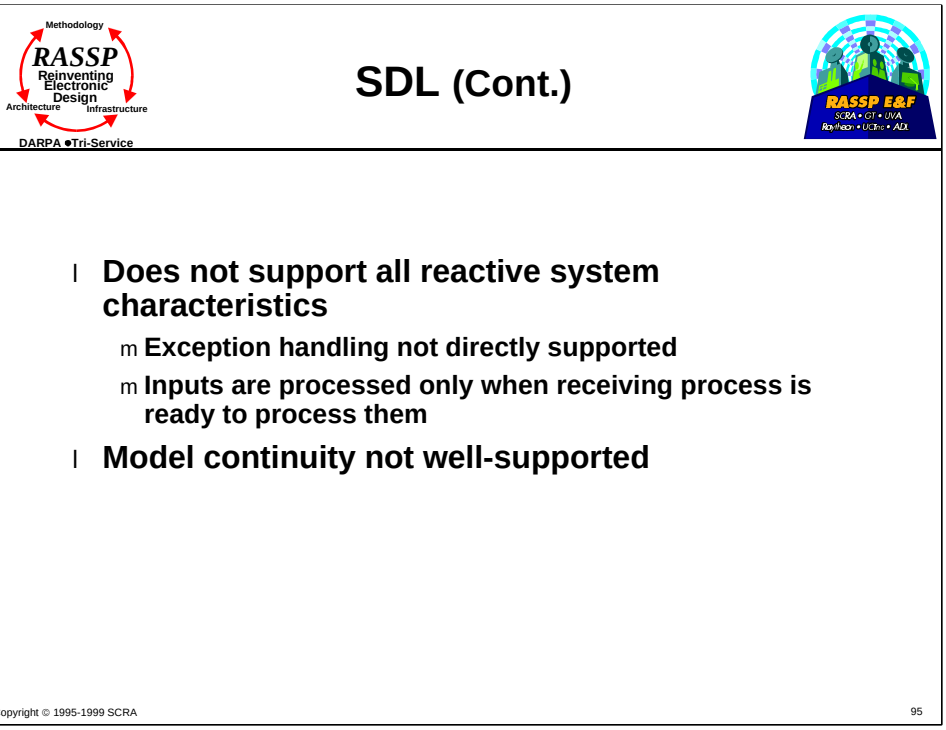


SDL provides three views of a system: structural, behavioral and data. It is the behavioral view of the system that is used to specify the system's reactive nature. The structural model is generated hierarchically, starting from a *block* that is recursively decomposed into a number of *blocks* connected together by *channels*. The data is modeled as an Abstract Data Type, where one describes the available data-operations and data-values but not how they are implemented.

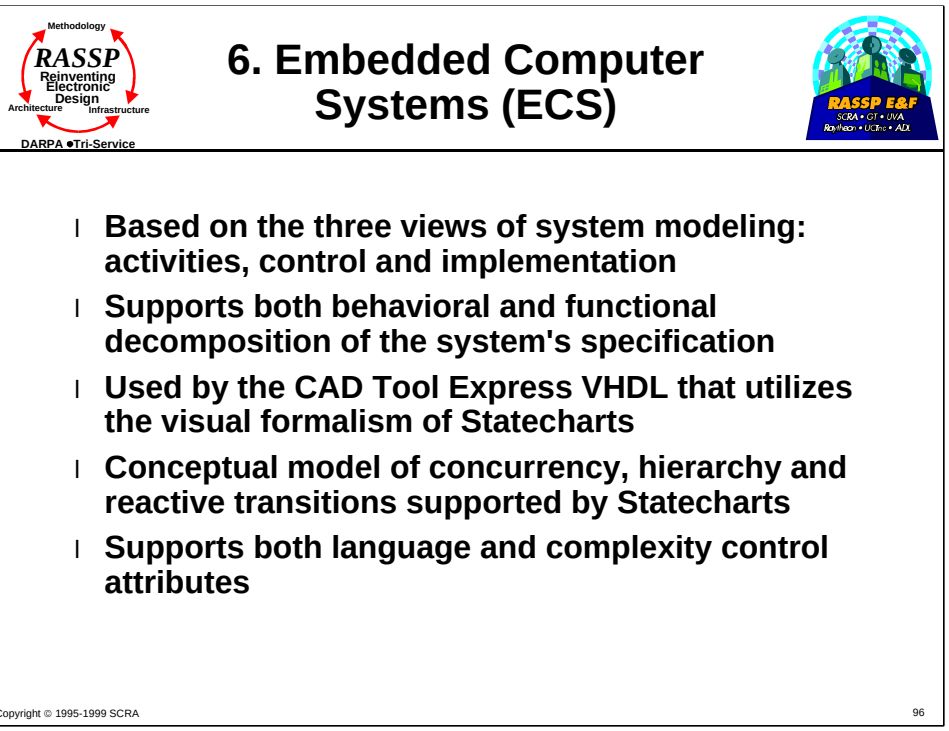
The system is modeled as a number of interconnected abstract machines. The machines communicate asynchronously.



At the topmost level, the *system block* has *channels* that allow interfacing with the system's environment. The behavioral model is a set of *processes* that are extensions of deterministic finite-state machines. The interaction between these *processes* is done via *signals*. These *processes* can be dynamically created and collectively represent the system behavior. Temporal ordering between the *signals* used in inter-*process* communication is specified using message-sequence charts, which are useful for debugging the specification.

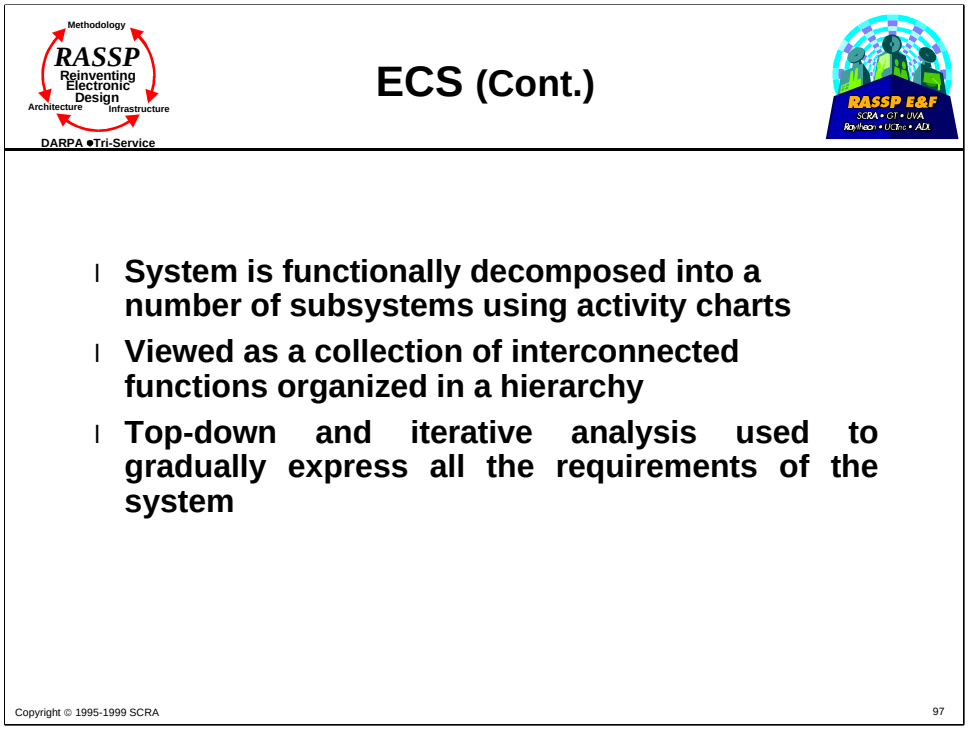


SDL supports the perfect-synchrony hypothesis and has an associated formal semantics. However, it does not support all the reactive system characteristics. For example, exception handling is not directly supported since the inputs are typically consumed by a process only when the receiving process is ready to process the input. Thus, if an exception condition is communicated as an input to the process, it may not be acted upon immediately. Rather, the exception will be handled when the receiving process is ready to process the arriving exception. The complexity-control attribute is well supported. Model-continuity is not well supported in the methodology.



The Embedded Computer Systems (ECS) methodology is based on the three views of system modeling: activities, control, and implementation. Express-VHDL is used as a computer-aided design tool for this methodology.

ECS supports both behavioral and functional decomposition of the system's specification. The system behavior is expressed using the visual formalism of Statecharts, an extension of FSM that significantly reduces the representational state-space explosion problem encountered by ordinary FSMs. This reduction is achieved due to the conceptual models of concurrency, hierarchy, and reactive transitions supported by Statecharts. A history operator, useful for expressing interrupt-handling, is also provided to significantly reduce the representational complexity.



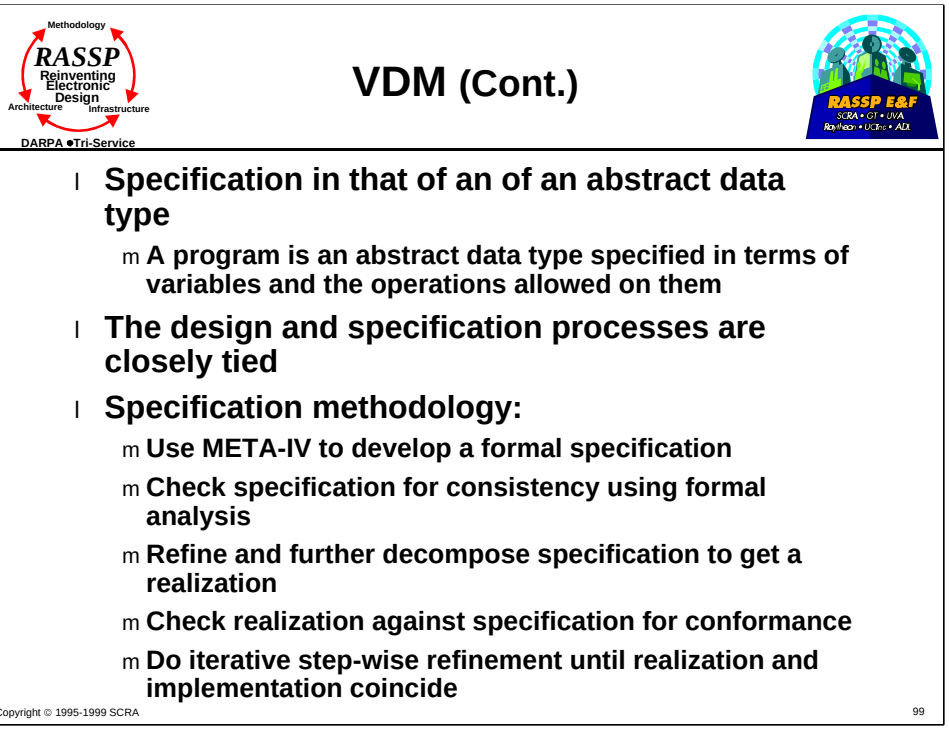
The system is functionally decomposed using activity charts, and is viewed as a collection of interconnected functions (activities) organized in a hierarchy. The activity charts visually depict the flow of information in the system, with the control of flow being represented by the associated Statecharts model.

To develop the specification, the methodology recommends a top-down and iterative analysis that gradually expresses all the requirements of the system. Conceptually, a system is decomposed into a number of subsystems, each carrying out a functionality, and a controller that coordinates the activities between these subsystems. The behavior of each system is represented by a Statecharts model. Each state in the Statecharts model can be refined further into AND and OR states. The default-entry states, needed synchronizations, associated timing constraints, etc. are specified next.

7. Vienna Development Method (VDM)

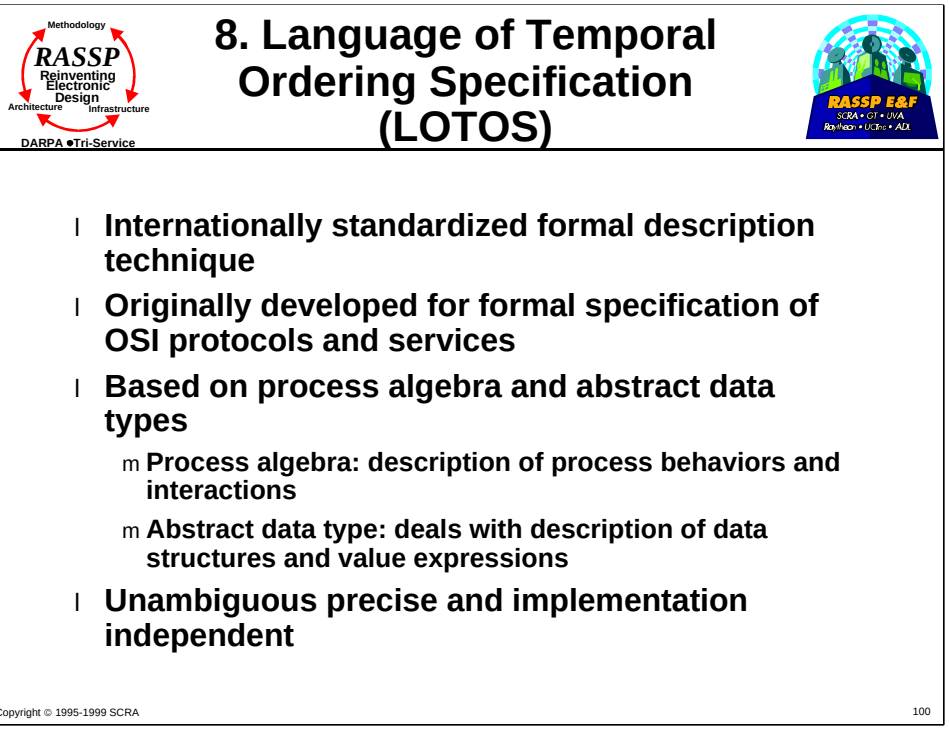
- **Abstract model-oriented formal specification and design method**
- **Based on discrete mathematics**
- **Specification is written as a specification of an abstract data type which is defined by a class of objects and a set of operations acting on these objects**
- **Supports model continuity very well**

VDM (Vienna Development Method) is an abstract model-oriented formal specification and design method based on discrete mathematics. The formal specification language of VDM is known as META-IV.



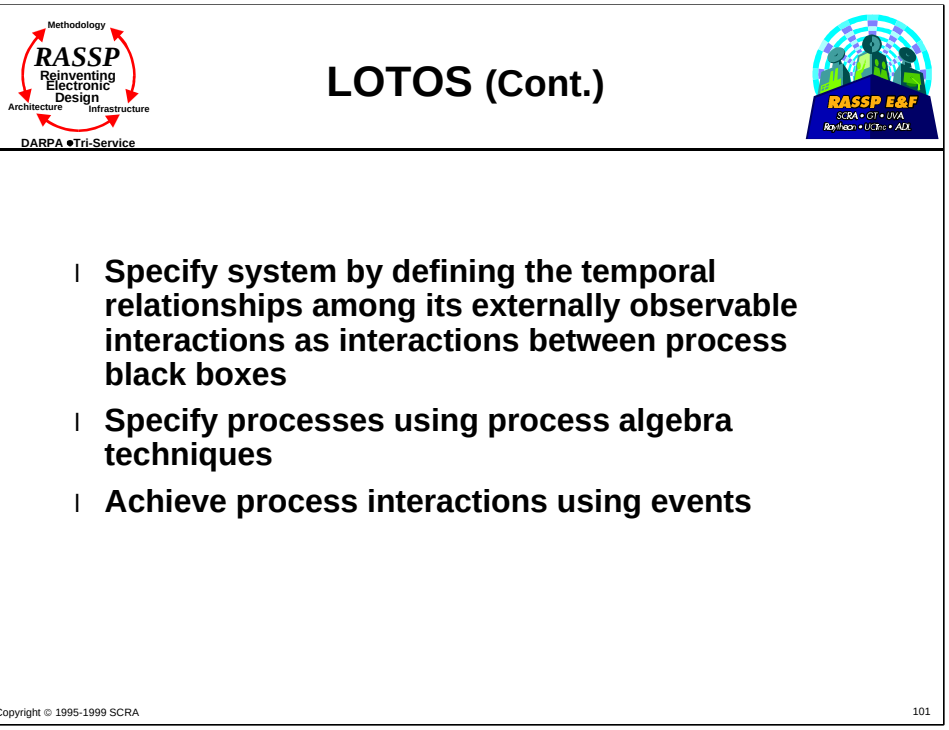
The specification is written as a specification of an abstract data type. The abstract data type is defined by a class of objects and a set of operations to act upon these objects while preserving their essential properties. A program is itself specified as an abstract data type, defined by a collection of variables and the operations allowed on these variables. The variables make the notion of state explicit, as opposed to property-based methods.

The specification development process is closely tied to the design process. The steps of the specification methodology is as follows. A formal specification is developed using the META-IV language. Once the specification is checked via formal analysis and found to be consistent, the specification is refined and further decomposed into what is called a *realization*. The realization is checked against the original specification for conformance. The specification is iteratively and step-wise refined until the realization is effectively a complete implementation.

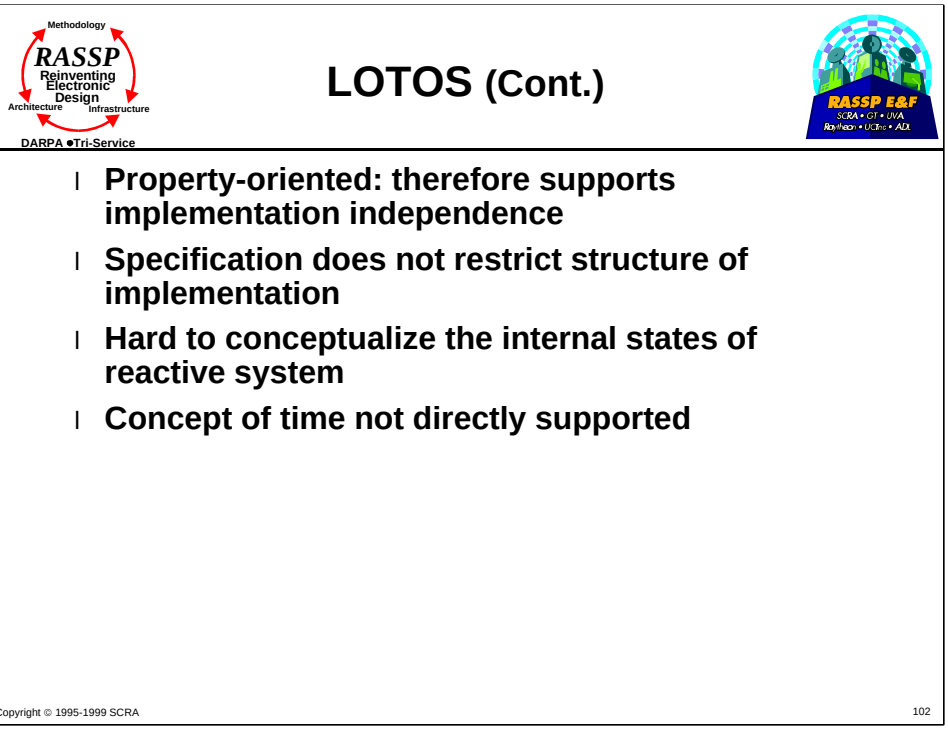


LOTOS (Language Of Temporal Ordering Specification) is an internationally standardized formal description technique, originally developed for the formal specification of OSI (Open Systems International) protocols and services.

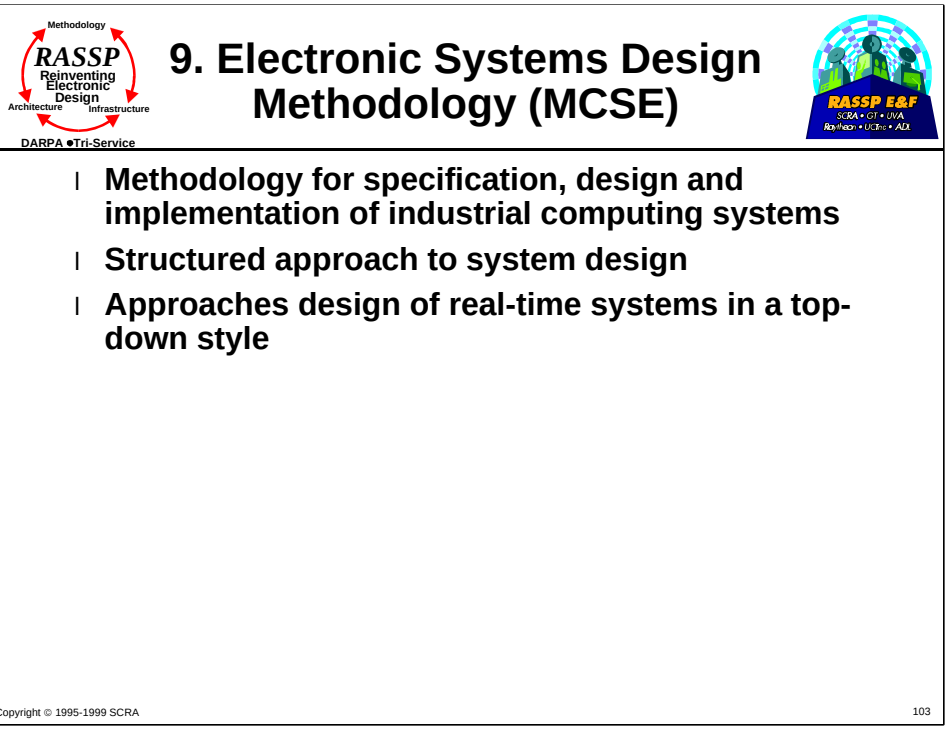
The specification is based on two approaches: process algebras and abstract data types. The process-algebra approach is concerned with the description of process behaviors and interactions and is based on Milner's Calculus of Communicating System and Hoare's work on Communicating Sequential Processes. The abstract data type approach is based on ACT-ONE which deals with the description of data structures and value expressions. The resulting specifications are unambiguous, precise, and implementation independent.



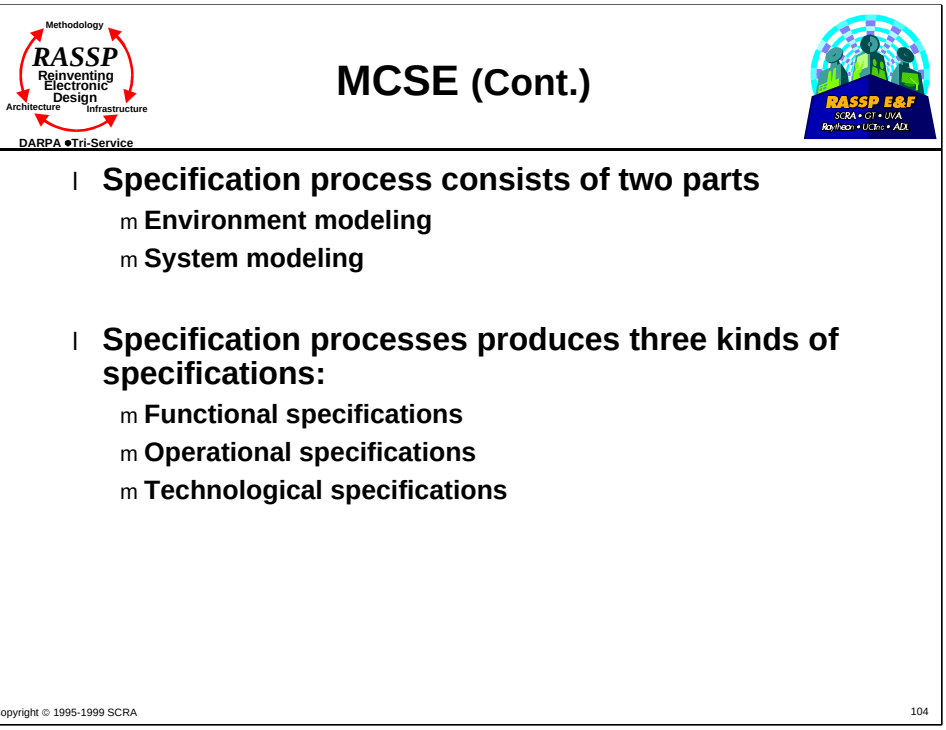
A system is specified by defining the temporal relationships among the interactions that make up its externally observable behavior. These interactions are between processes, which act as black-boxes. A black-box is an entity capable of performing both internal actions and external actions. The internal actions are invisible beyond its boundaries whereas the external actions are observable by an *observer* process. Interactions between these processes is achieved using *events*. The processes are specified using process algebra approach, which allows the description of behavior as a composition of basic behaviors using a rich set of combining rules.



Being property-oriented makes LOTOS hard to conceptualize the internal states of reactive system. Further, it also becomes hard to generate an implementation from the specification. The concept of time is also not directly supported.



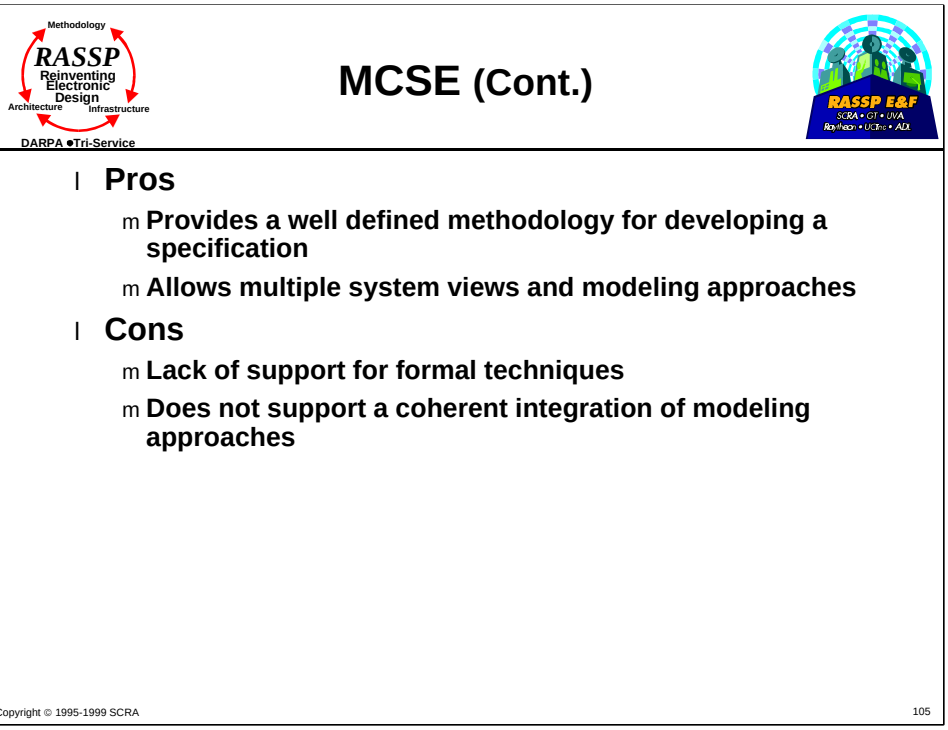
MCSE (Methodologie de Conception des Systemes Electroniques) is a methodology for the specification, design, and implementation of industrial computing systems. The methodology is characterized by its top-down approach to the design of real-time systems, and is structured into several steps from system specification to system implementation.



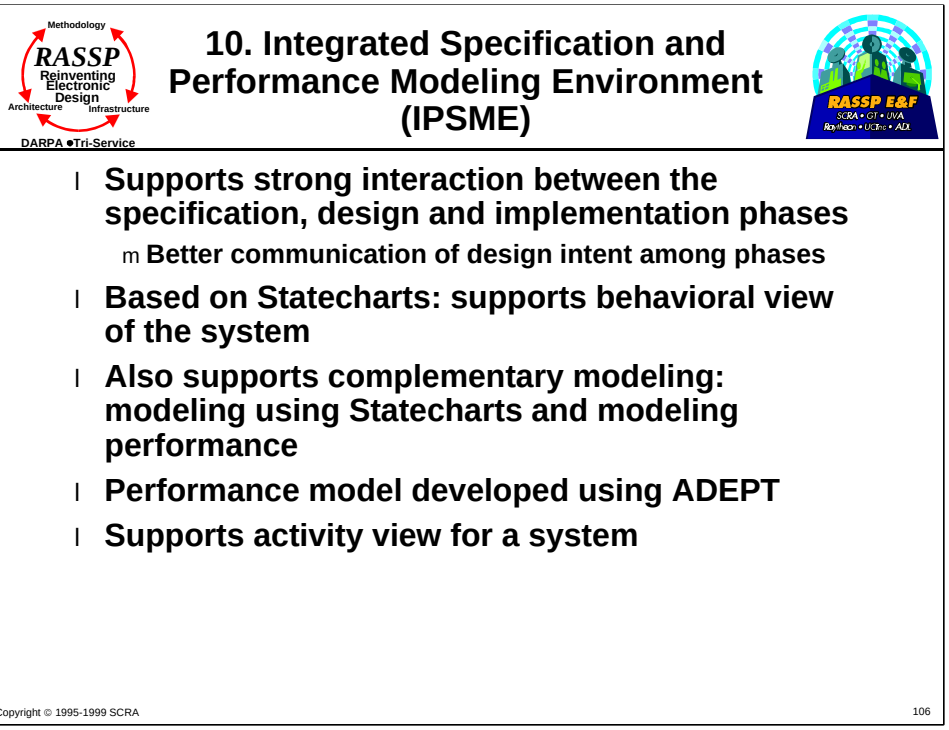
Three kinds of specifications are produced during the specification process. First, functional specifications include a list of system functions and a description of the behavior of the system's environment. Second, operational specifications concern the performance and other implementation details that are to be used in the system. Third and finally, technological specifications include specifications of various implementation constraints such as geographic distribution limitations, interface characteristics etc.

There are two main parts in the specification process: environment modeling and system modeling. In the environment-modeling part, the environment is first analyzed to identify the entities that are relevant to the system. Next, a model is created representing the identified entities and their interactions, thus providing a functional description of the environment. In the system-modeling part, the system under design is first delimited in terms of its inputs and outputs. Next, a functional specification of the system is developed, which describes the functions to be carried out by the system on its environment. This functional specification is developed by characterizing the system in terms of system inputs and outputs, system entities, or system activities.

There is a lack of support for formal techniques. While several modeling techniques and system views are supported, a coherent integration of these diverse approaches is not supported.

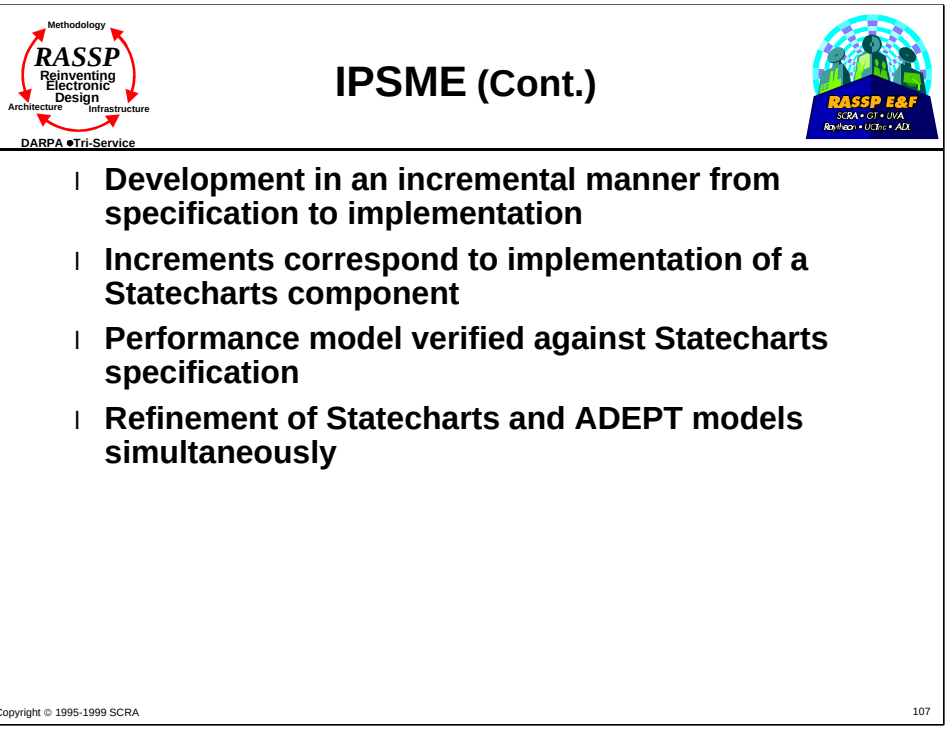


Pros and Cons associated with MCSE are described.

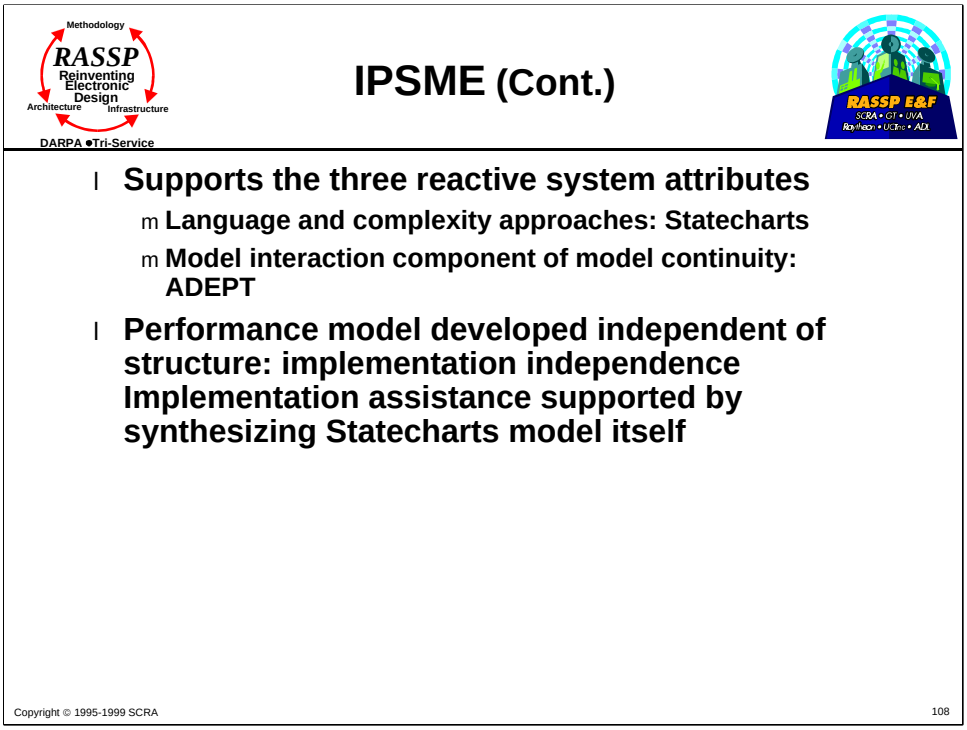


The Integrated Specification and Performance Modeling Environment is an evolving specification modeling methodology that supports a strong interaction between the specification phase of a design process with design and implementation phases. As a result of this interaction, there is an increased and better communication of design intent among these phases.

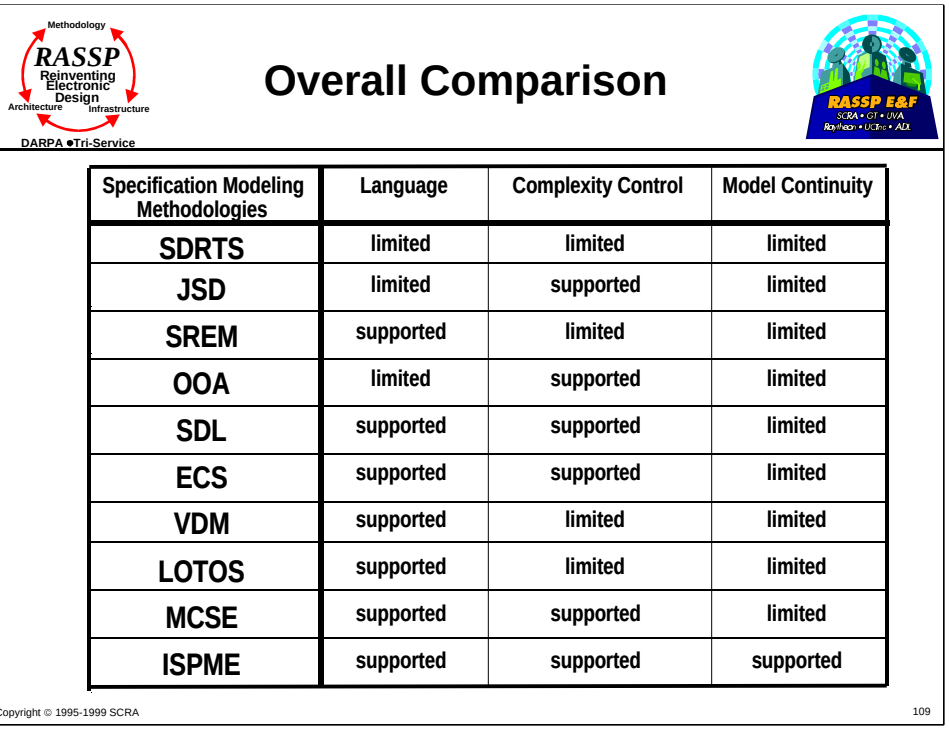
ISPME is based upon the language of Statecharts, similar to ECS modeling methodology. As a result, it supports the behavioral view of the system. However, ISPME also supports complementary modeling, where some aspects of the system are modeled using Statecharts, while the remaining aspects are represented as a performance model. The performance model is developed using ADEPT which is based on an extension of Petri-nets. Since a performance-model can coexist with the Statecharts specification, ISPME supports the activity view for a system.



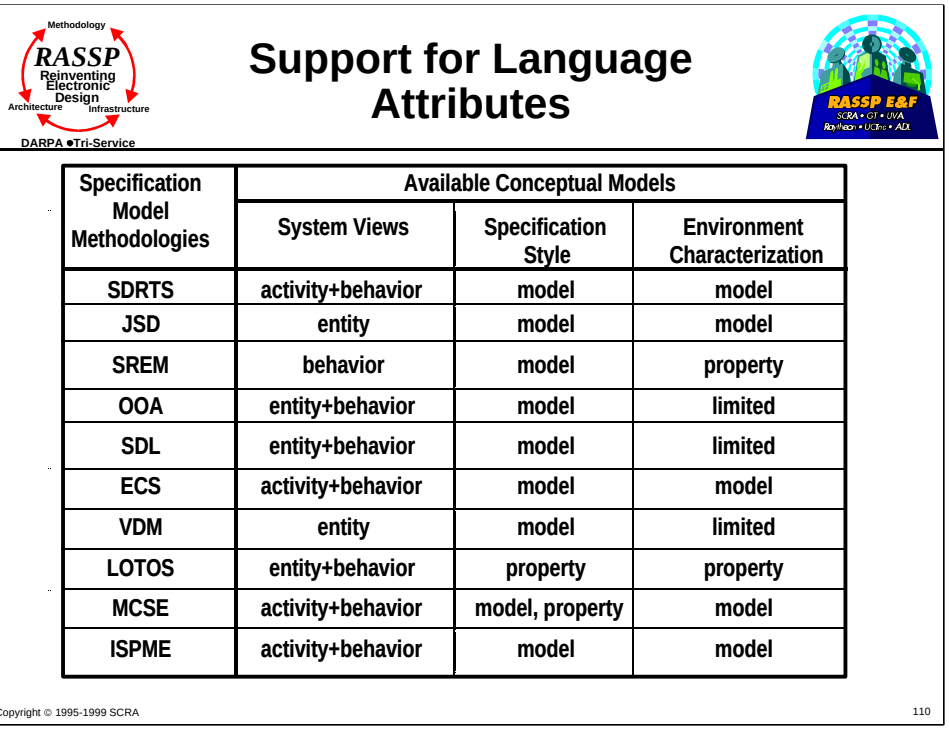
The methodology supports the development of a complete implementation from the specification in a incremental and iteratively refined manner. Each increment represents a proposed implementation of a component of the Statecharts model. The performance model of the proposed implementation is verified against its Statecharts counterpart. At each iteration, both the Statecharts and the ADEPT models can be refined, since it is possible that one may encounter inconsistencies between the specification and the implementation. As a result of this integrated-modeling approach, the methodology extends the specification phase to later design stages. Such extension improves communication of design intent between various phases of design.



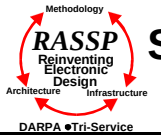
Description of IPSME continued.



We now tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.



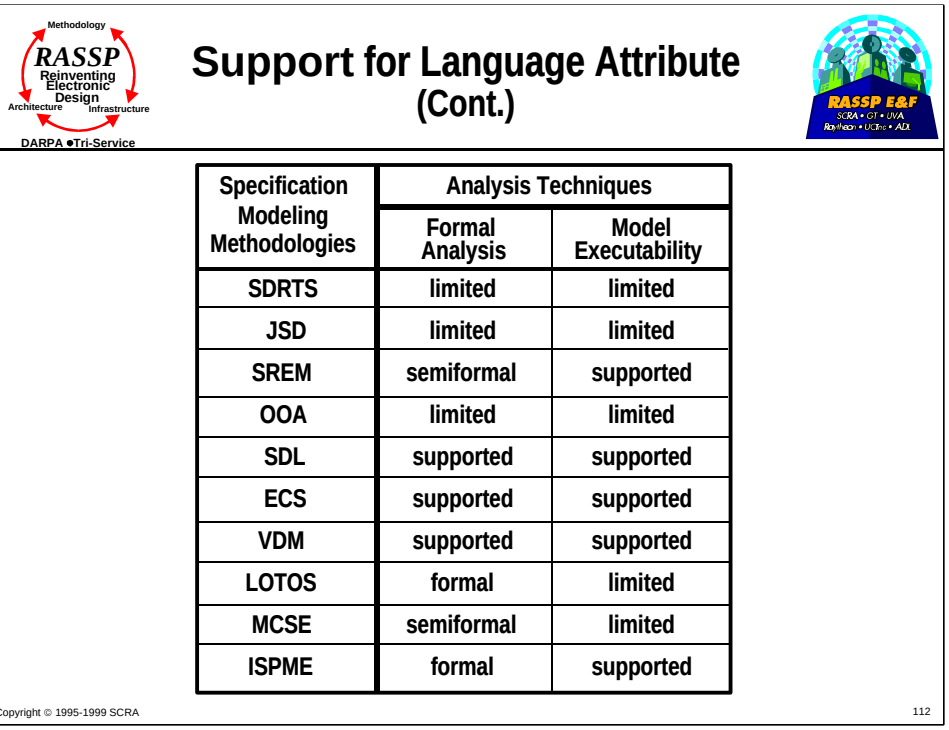
We now tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.



Support for Language Attribute (Cont.)

Specification Model Methodologies	Available Conceptual Models		
	Timing Constraints	Modeling Time	Exception Handling
SDRTS	indirect	limited	limited
JSD	direct	supported	limited
SREM	direct	supported	limited
OOA	indirect	limited	limited
SDL	indirect	supported	limited
ECS	indirect	supported	supported
VDM	indirect	limited	supported
LOTOS	indirect	limited	supported
MCSE	direct	supported	limited
ISPME	indirect	supported	supported

We now tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.

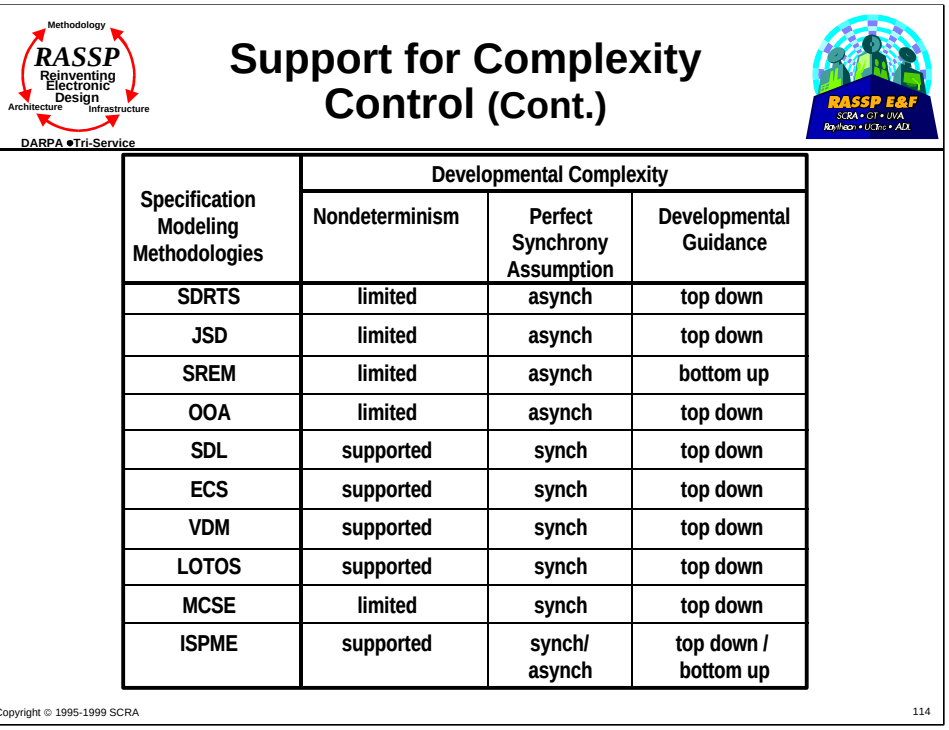


We now tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.

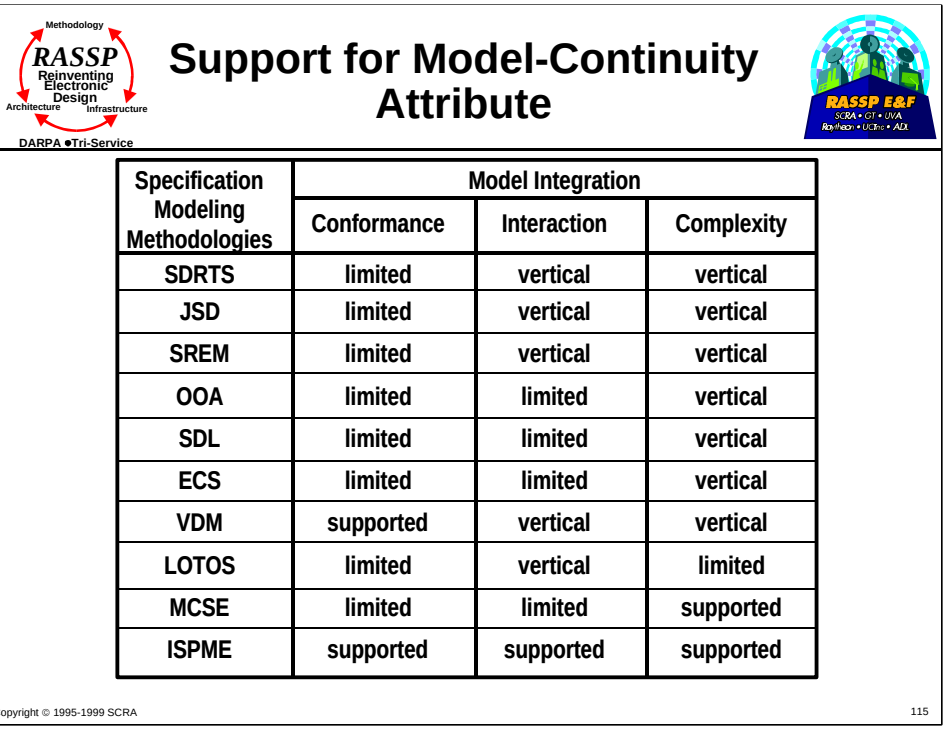
Support for Complexity Control

Specification Modeling Methodologies	Representational Complexity		
	Hierarchy	Orthogonality	Representation Scheme
SDRTS	supported	limited	graphical
JSD	supported	supported	graphical
SREM	limited	limited	graphical
OOA	supported	supported	textual
SDL	supported	supported	graphical
ECS	supported	supported	graphical
VDM	supported	supported	textual
LOTOS	supported	supported	textual
MCSE	supported	supported	graphical
ISPME	supported	supported	graphical

We continue to tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.



We continue to tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.

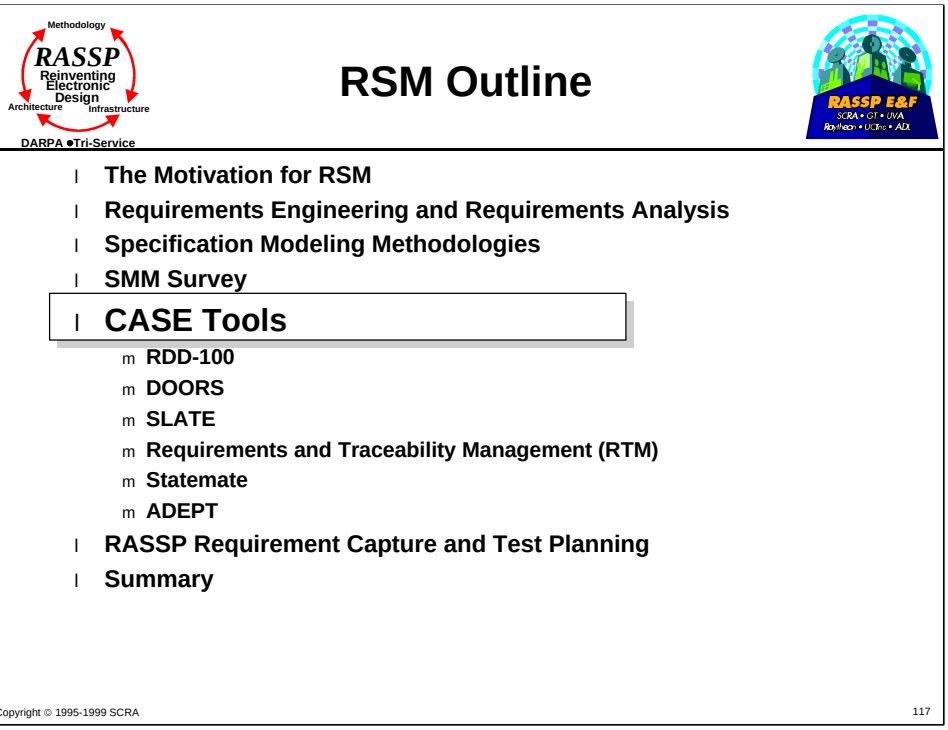


We continue to tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.

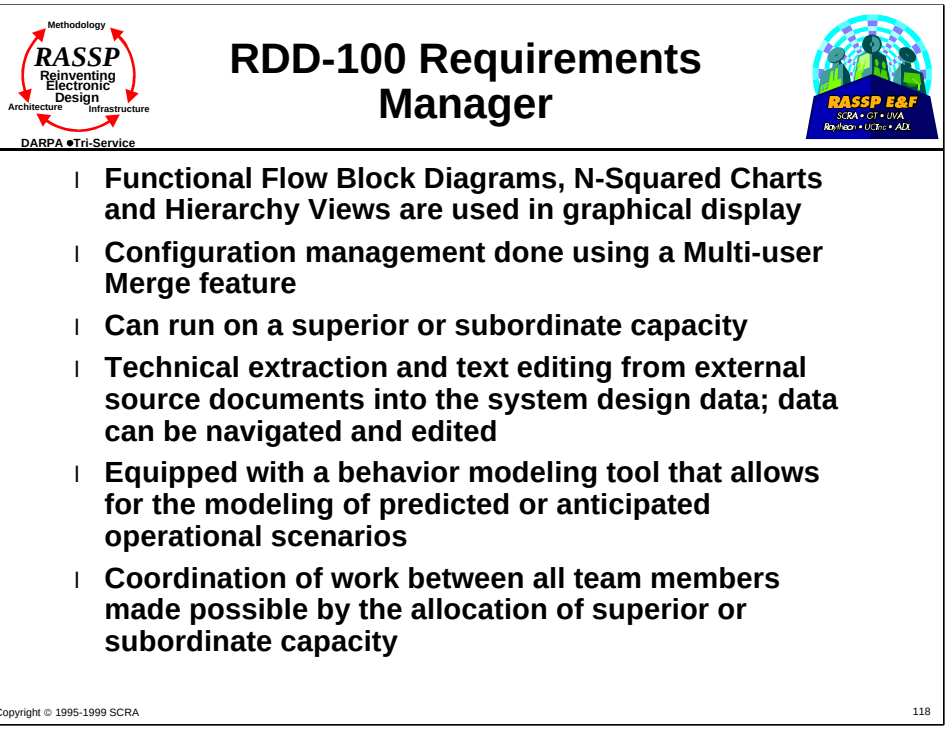
Support for Model-Continuity Attribute (Cont.)

Specification Modeling Methodologies	Implementation Assistance	Implementation Independence
SDRTS	limited	limited
JSD	limited	supported
SREM	supported	limited
OOA	limited	supported
SDL	supported	supported
ECS	supported	supported
VDM	supported	limited
LOTOS	limited	supported
MCSE	limited	supported
ISPME	supported	supported

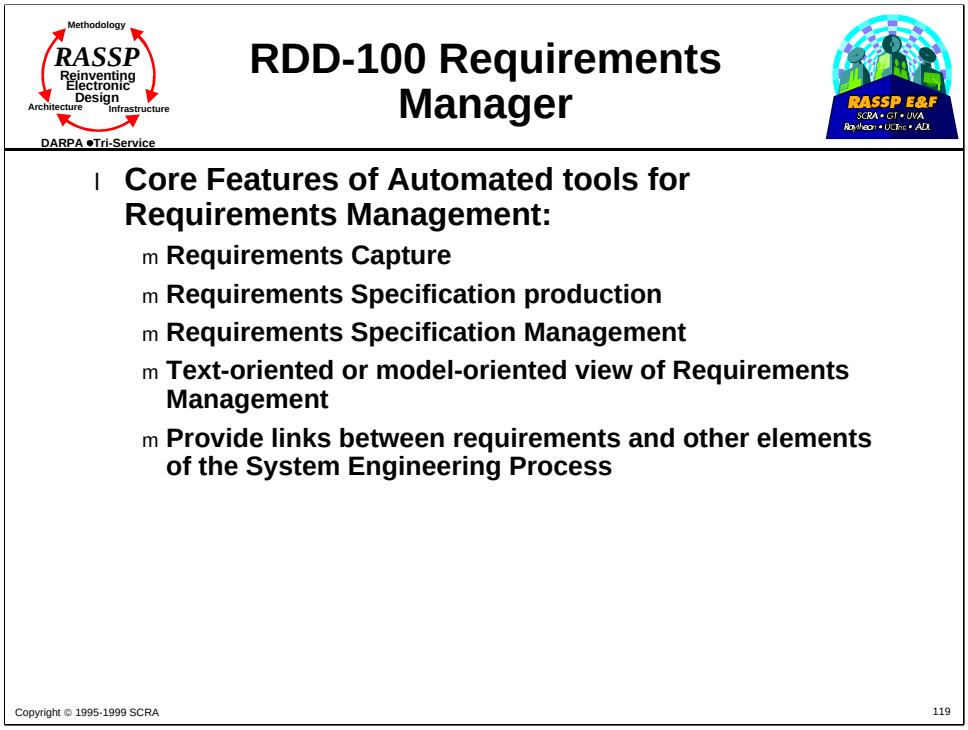
We continue to tabulate the various methodologies and the metrics as shown. Our rankings are relative, and subjective.



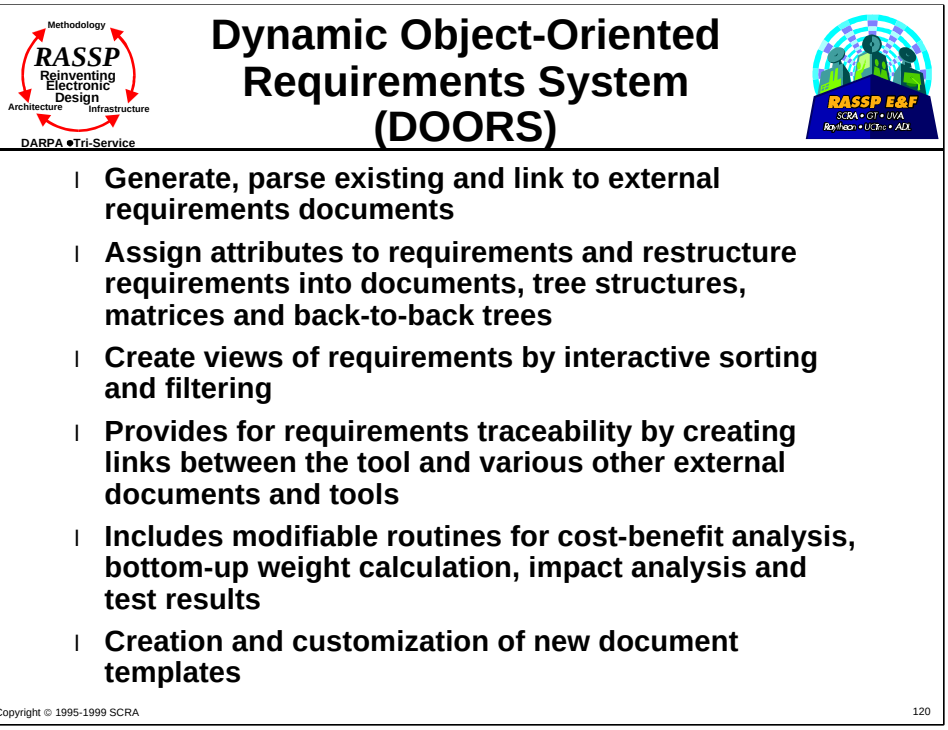
Several tools exist in current practice that could assist in the implementation of the RSM methodology. We will describe some of them briefly.



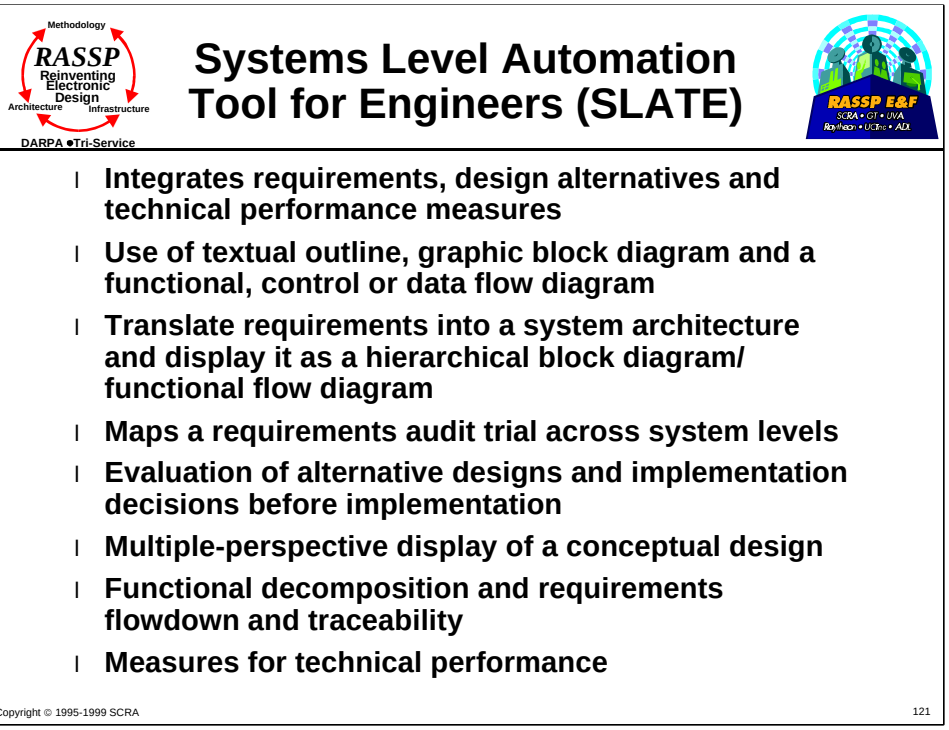
RDD-100 is a commonly used tool for capturing requirements.



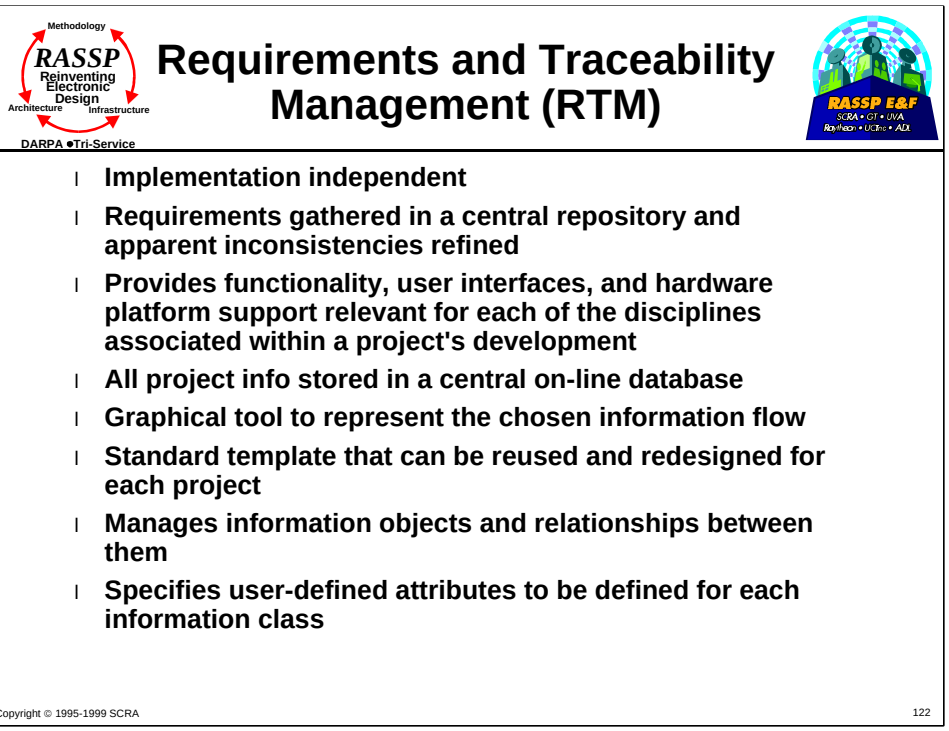
The functionality of RDD-100 is described in the above slide.



DOORS is another tool that supports the RSM process.



SLATE also supports the RSM process.

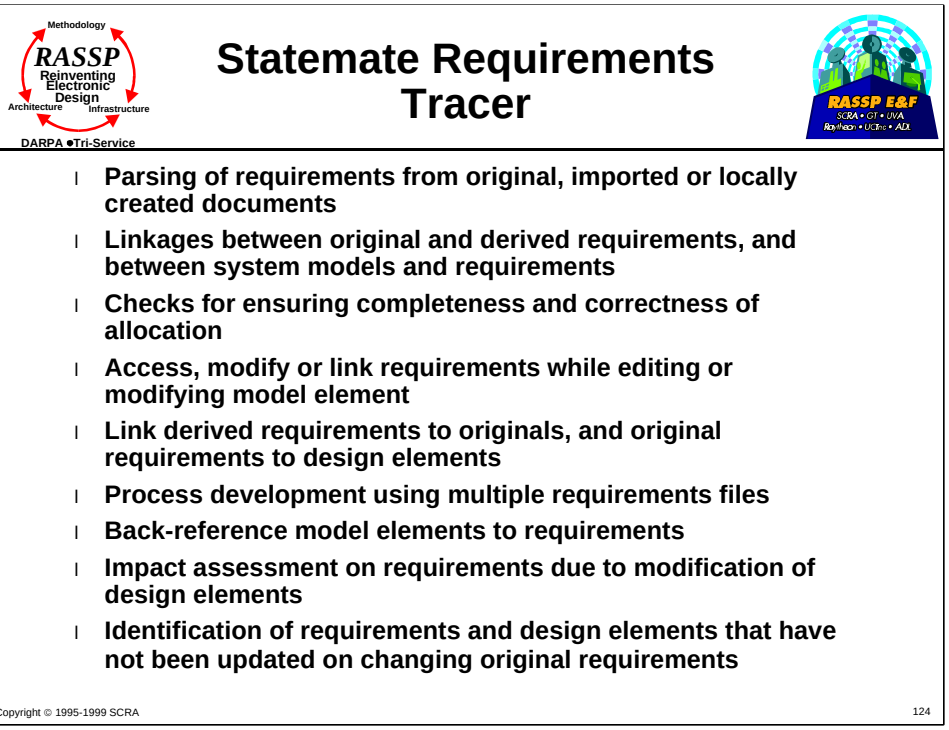


RTM also supports the RSM process as outlined in this module.

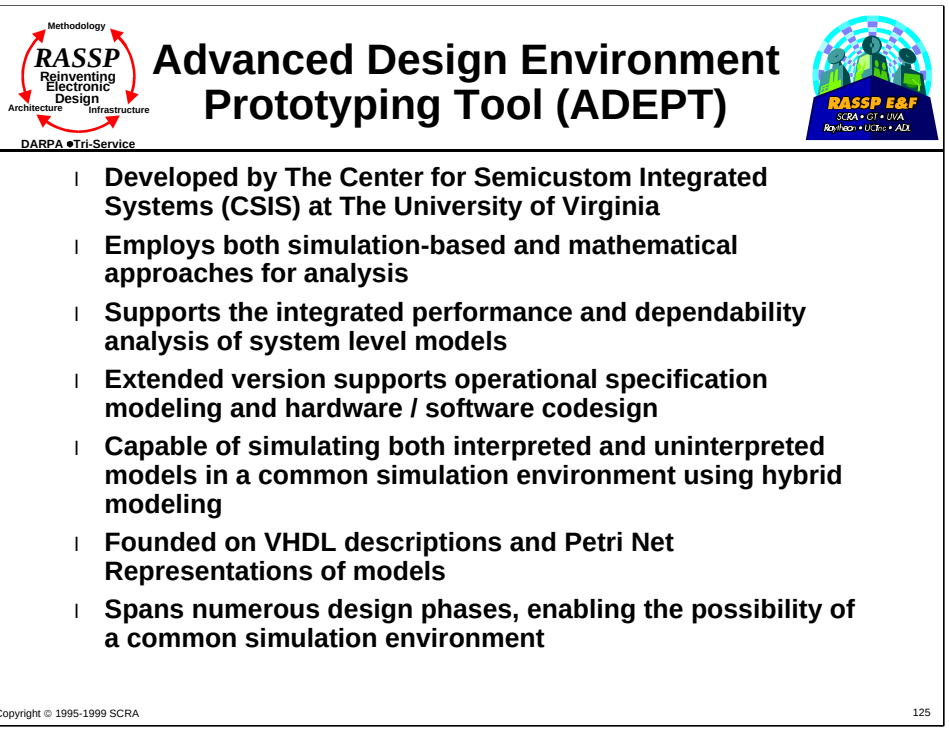
Requirements and Traceability Management (RTM)- (Cont.)

- Captures the contents of requirements documents into a project database
- Identifies requirements by selecting the style or component type in the document corresponding to an information class
- Requirements can be extracted from an ASCII document and stored in the database as information objects
- Edit, merge or expand information to show information which is implied
- Combine different documents in a number of different views
- Assign attribute values to characterize each requirement
- Select, view and edit individual records
- Track dynamically, individual requirements through the project life-cycle

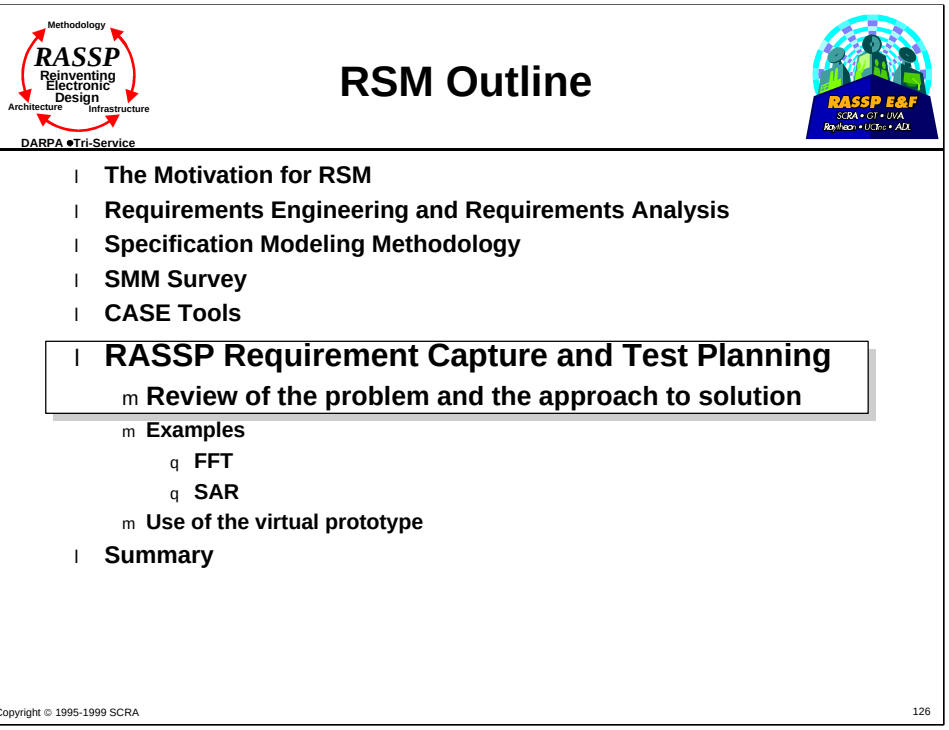
RTM description is continued on this slide.



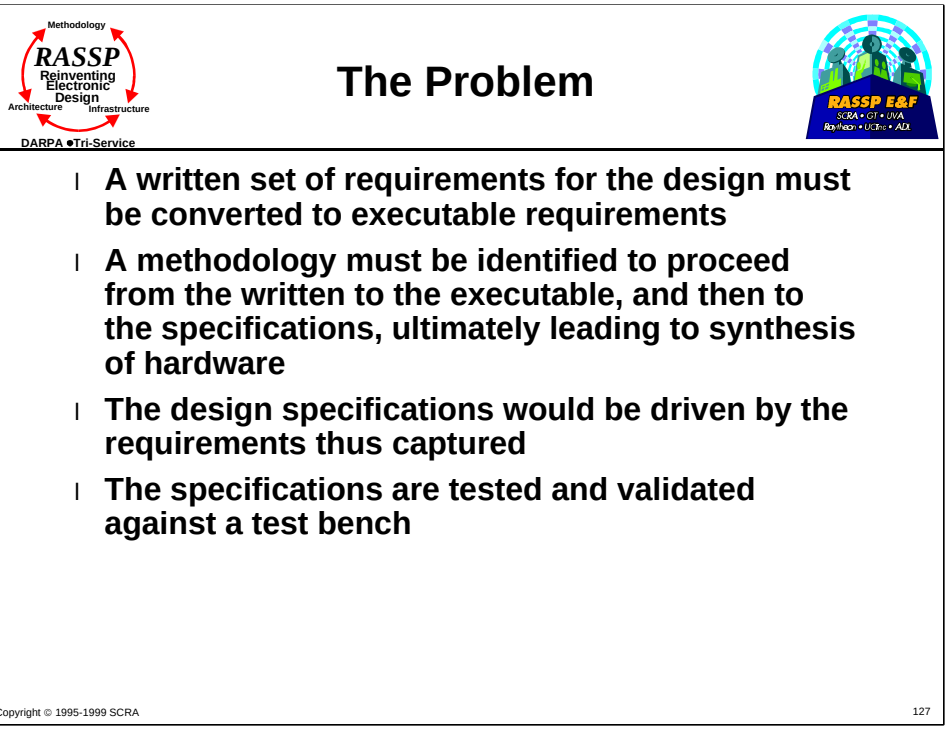
Statemate is a tool developed by iLogix that allows capture of the requirements/specifications.



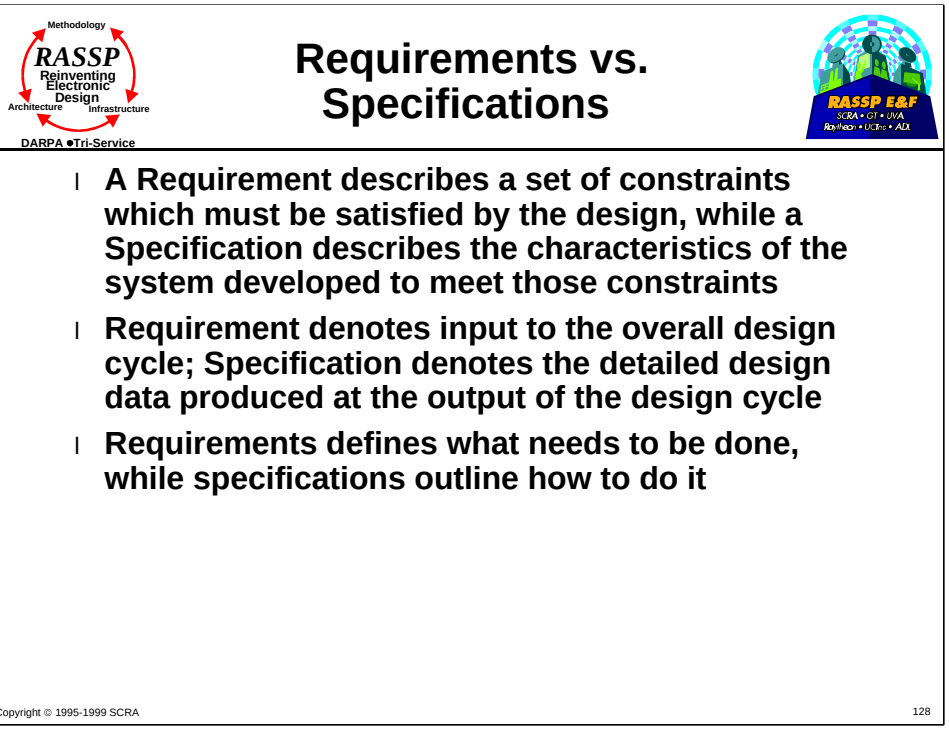
ADEPT is a tool developed at the University of Virginia and models architectures of systems using Petri net representations of systems.



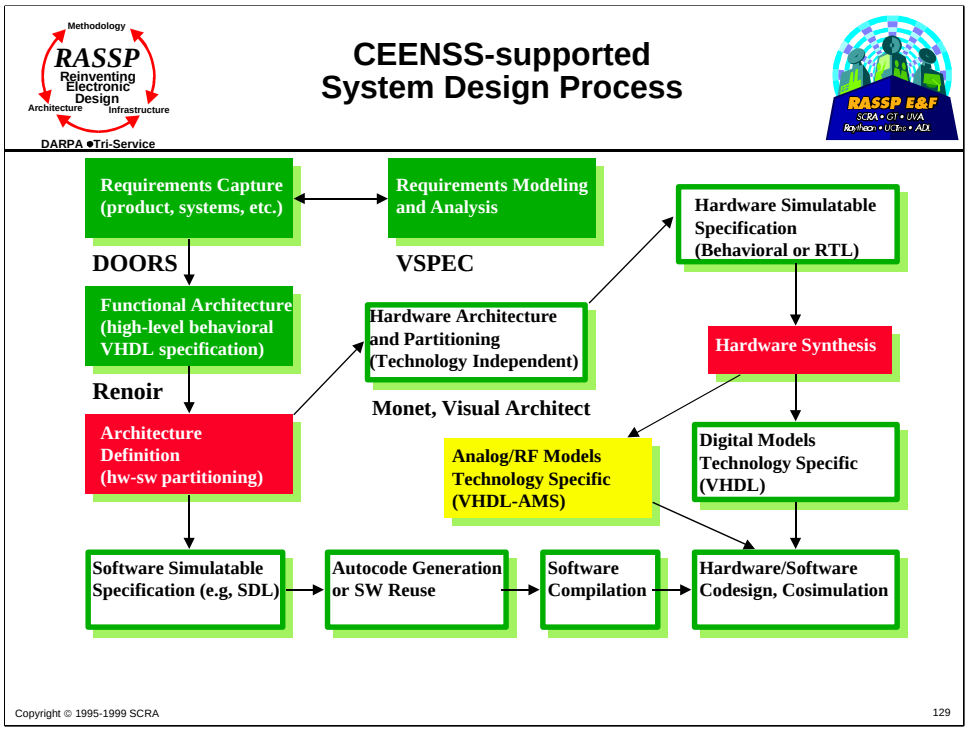
We now describe some RASSP contributions in the area of RSM methodologies.



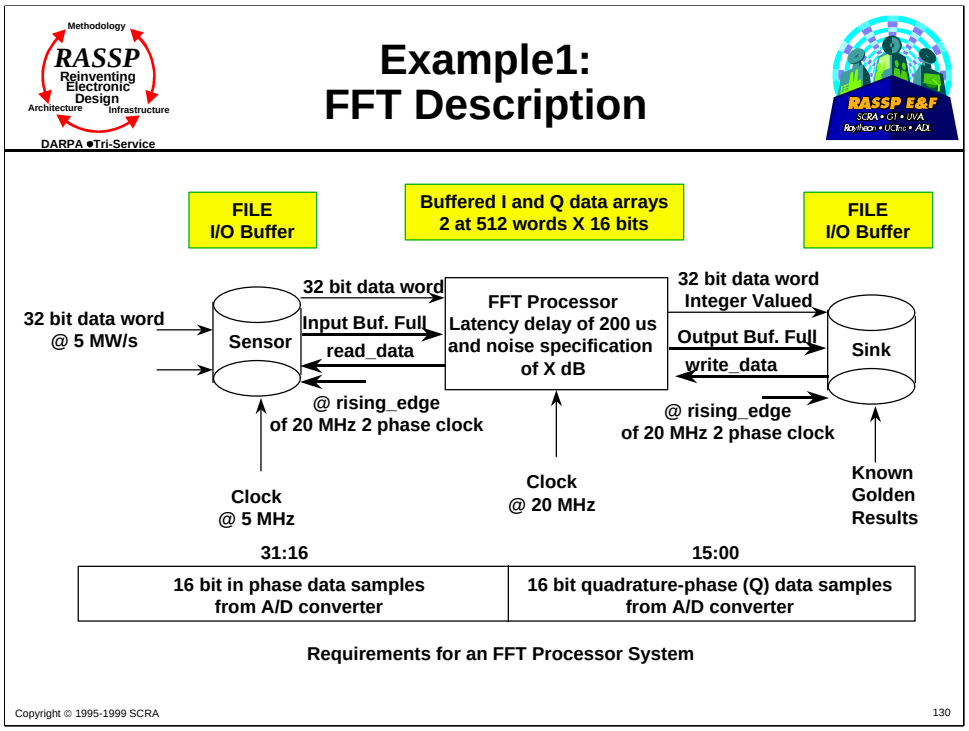
The technical problem within RSM is described in the context of the RASSP methodology.



The difference between Requirements and Specifications is highlighted in this slide.



The CEENSS program has recently proposed a RSM methodology and supporting Tools that are very similar to the RASSP process, and are shown in this slide. A top down design flow is shown together with the verification steps, and the detailed software/hardware design/integration steps.



The design requirements for a hypothetical signal processor is shown above. The written requirements corresponding to this figure are outlined in the lab document, where the process of Requirements and Specifications is demonstrated using this example.

Example1: Requirement to Executable Requirement

- **Requirement: " The system clock will operate with 2 phases and at a certain given frequency"**
- **VHDL Code Fragment:**

```
Begin -- architecture
…
internal_signals:
 phase1 <= NOT phase1 AFTER period;
 phase2 <= phase1 AFTER phase_delay;
generated_signals:
 ph1 <= phase1 AND NOT phase2;
 ph2 <= NOT phase1 AND phase2;
End;
```


These examples are also taken from the Lab 1 document and the corresponding VHDL code.

Example 1:Requirement to Executable Requirement- (Cont.)

- **Requirement: "The processor shall buffer input data into 512 element buffers and process the data with max. latency of 20 us"**
- **VHDL code:**

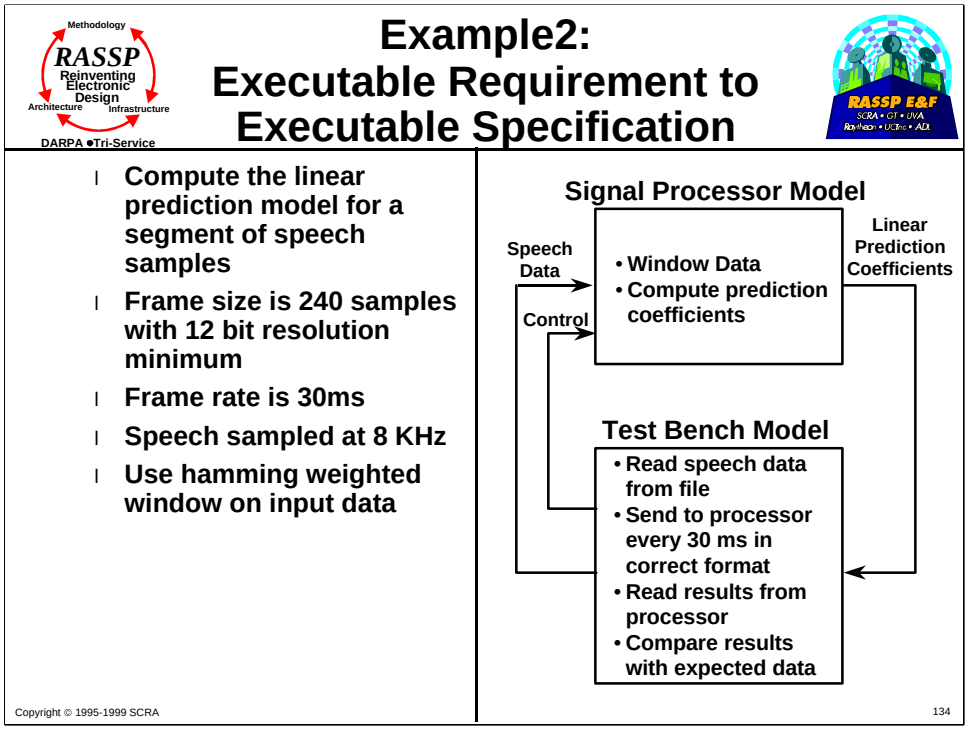
```
…
IF counter = 512 THEN
 stop_read_data <= '1', '0' after 5 ns;
 -- do Signal Processing here
 start_write_data <= '1' after 20 ns; -- start
  writing data to the output
 counter := 0; -- reset the counter so that more
  data can be read
END IF;
…
```


The English requirement is converted to an executable form as shown in the VHDL code.

Example1: Scope of Executable Requirements

Item	Requirement	Represented by executable requirement
Data I/O ports	Data format specified	✔
	Protocol specified	✔
	Timing & data rate specified	✔
	Physical	-
Control port	Commands and behavior specified	✔
	Data format specified	✔
	Protocol specified	✔
	Timing & data rate specified	✔
	Physical	-
Modes of operation	A 512 point FFT algorithm	✔
Accuracy	Max. error to be specified	✔
Latency	20 us	✔
Physical constraints	Size, weight, power	-
Testability	Best practice	-
Scalability	TBD	-
Environment	Air cooled	-
Assumed quantity	TBD	-

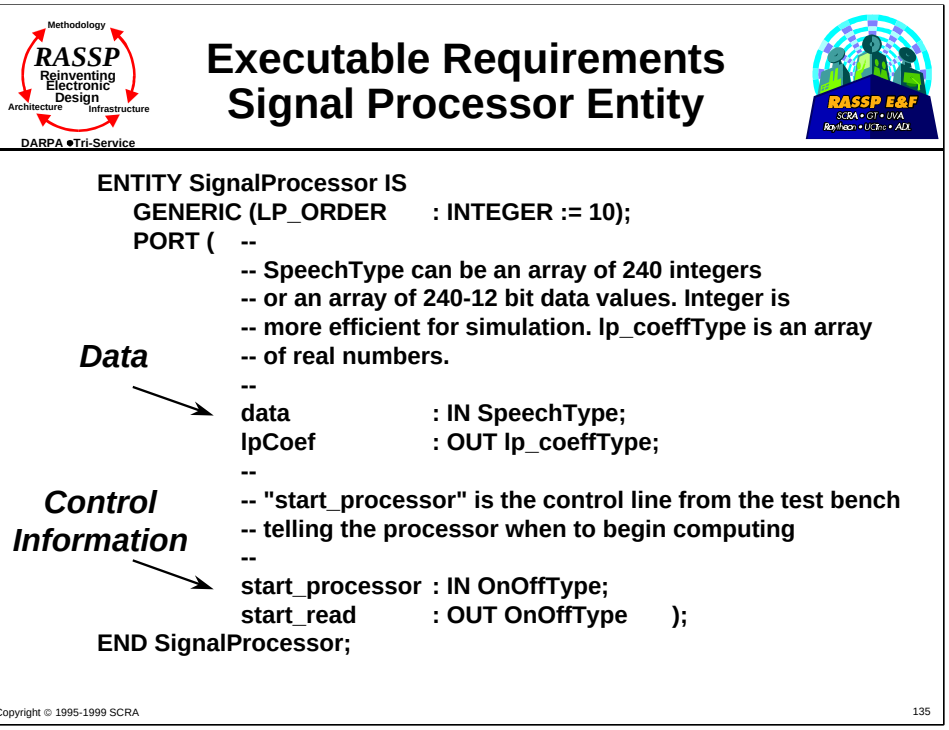
The requirements for the example signal processor are summarized in the above table. The scope of the executable requirements is shown by tick marks.



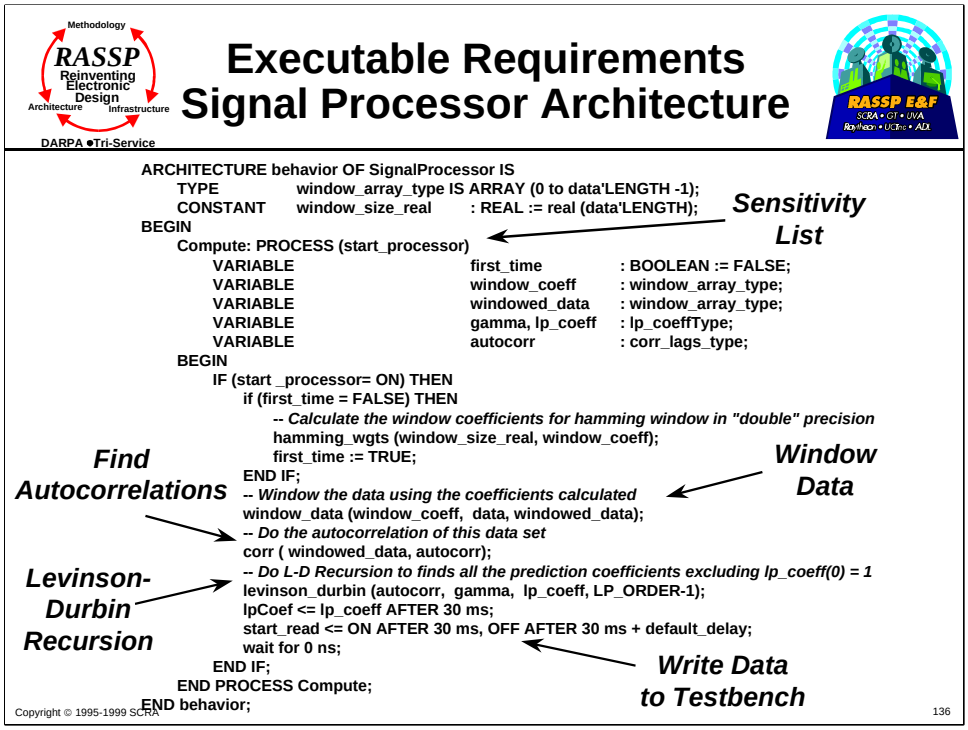
This example will be used to illustrate how an executable requirement may be specified in executable form.

This is a simple example for illustrative purposes only. Speech is sampled at 8 kHz and input to a signal processor in frame sizes of 240 samples representing 30 ms of speech data. The processor must window the speech data using a hamming window function and calculate the linear prediction model for each of the speech segments.

The test bench inputs data to the signal processor and monitors its outputs. The test bench reads its speech data from a file and outputs the linear prediction coefficients to a file after the processor has completed its computations.



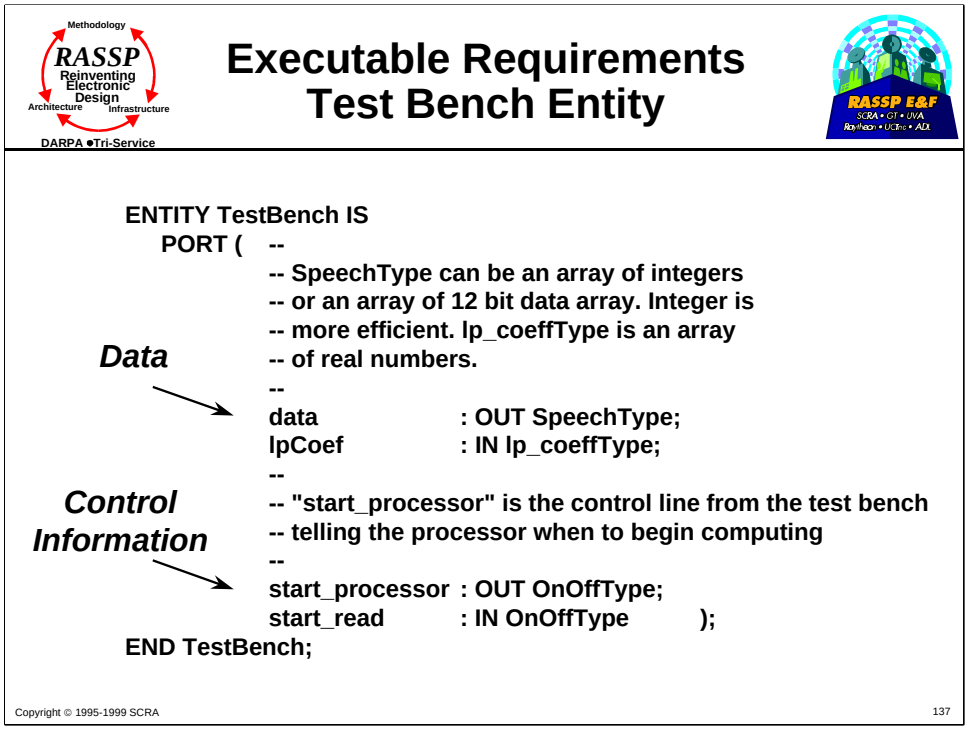
This is the entity description of the signal processor. Since the processor is a linear prediction computational engine, the linear prediction (LP) order is passed as a generic. The ports used for input and output to the processor include its data and control lines. The data lines consist of 240 samples of speech data in the format "SpeechType" and the output data are the LP coefficients. The control input information indicates when the processor starts computing the coefficients and the output control line indicates to the test bench when to read the results.



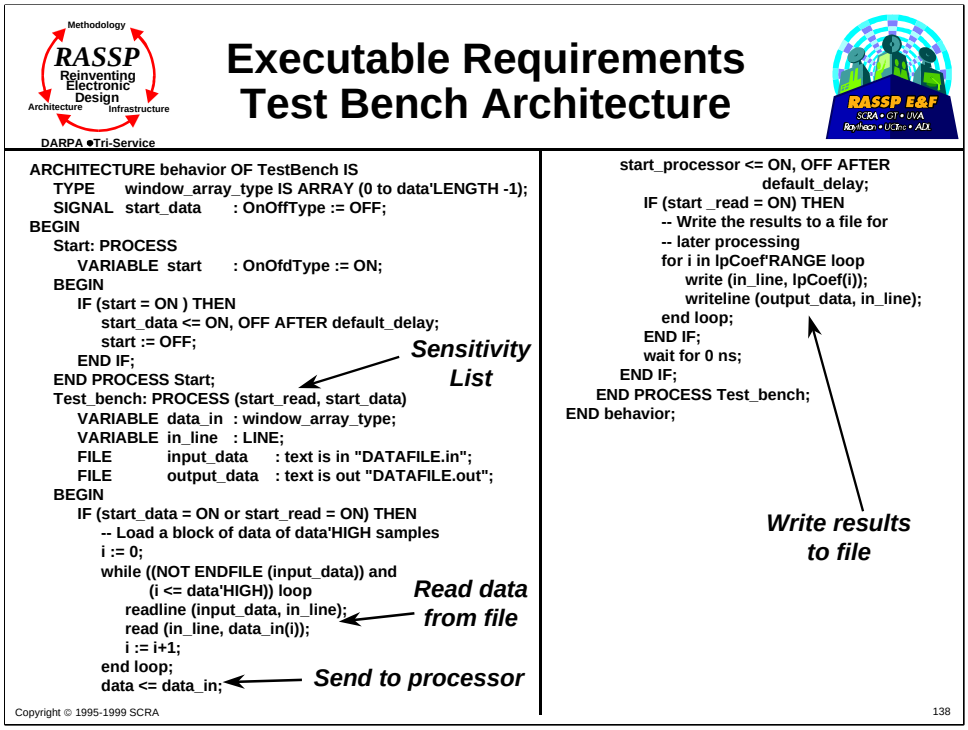
The architecture describing the functionality of the signal processor is shown above. It consists of one process executed in zero time and sensitive to the "start_processor" signal from its input port. When it receives this signal, the speech data is assumed to present and the processor begins calculating the LP coefficients, It calls a sequence of procedures to perform the necessary functions (window_data, corr, levinson-durbin). When it has finished, it puts the results on the lpCoef lines and sets the trigger to the testbench.

Procedural calls used to perform functionality.

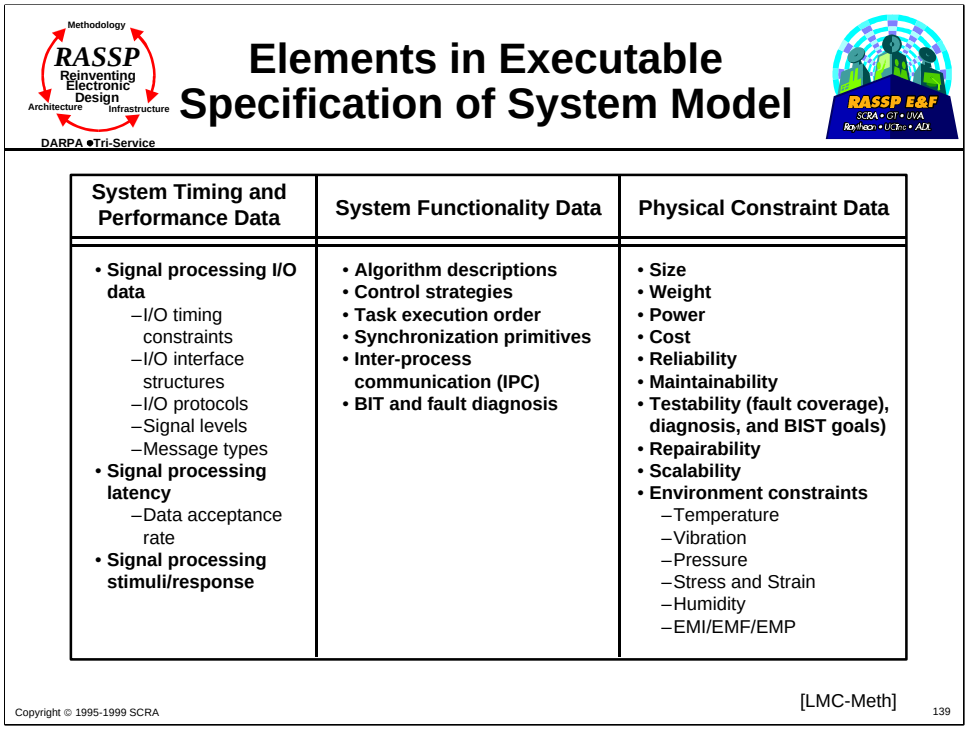
Results put on output lines after the maximum specified delay time.



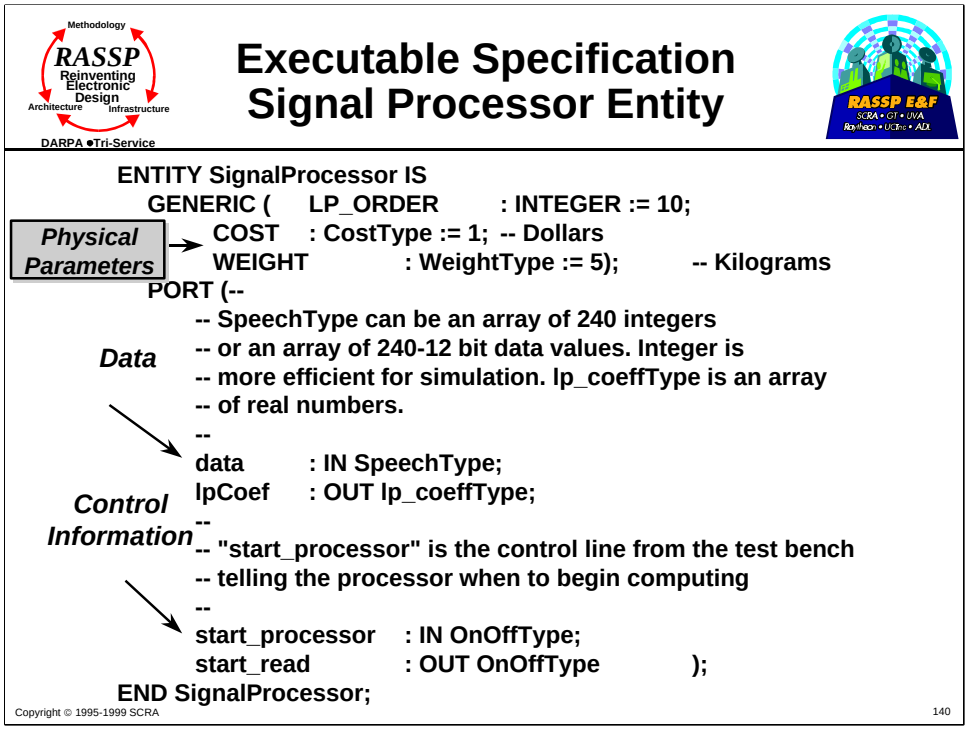
This is the entity description of the test bench for the signal processor. The ports used for input and output to the test bench include its data and control lines. The output data lines consist of 240 samples of speech data in the format "SpeechType" and the input data are the LP coefficients. The control input information indicates when the test bench starts reading the coefficients and the output control line indicates to the processor when to start processing the data sent by the test bench.



This is a possible architecture for the given test bench. It contains two processes, one to start the reading of data at time zero and the second to read and store data at the appropriate times in the future. The data is read from a file in the form of 12 bit speech samples. It is then sent to the processor by assigning it to the signal "data" and setting the trigger signal "start_processor". The results are stored at the end of the routine when the "start_read" signal is set to ON.



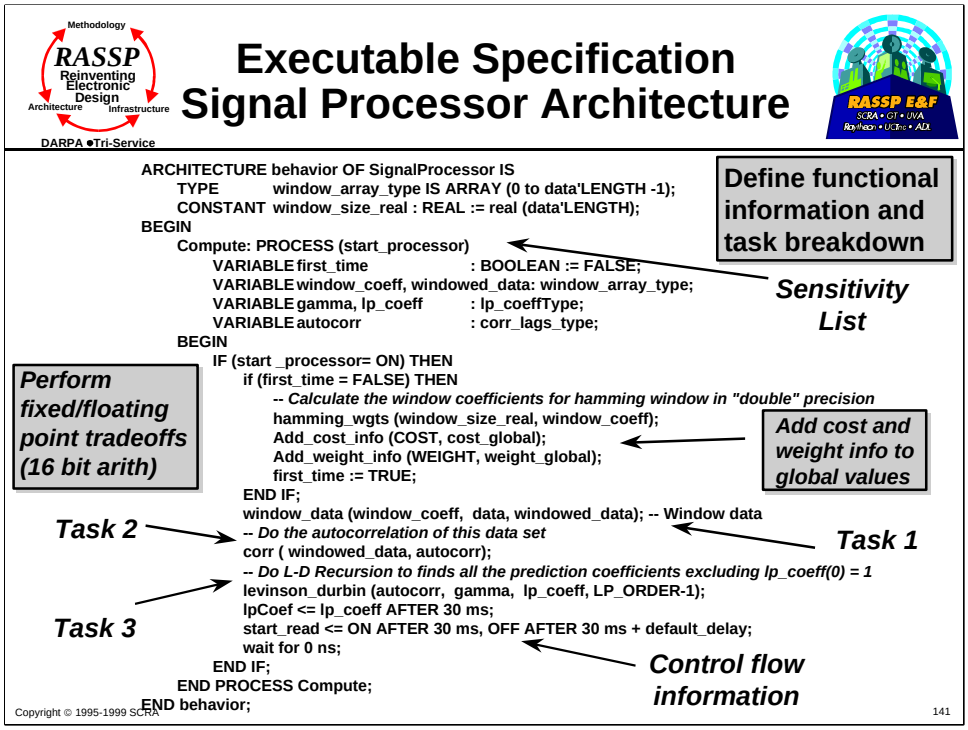
This chart presents the elements contained in the executable specification. The three main categories of the previous slides are expanded upon. The data in this simulation is not fixed at the end of the systems definition design process but can be modified as more information becomes available from future design stages. For example, the size and weight can be estimated initially and as more numbers become available the high level model is updated to track the lower level details.



This is the entity description of the signal processor. Since the processor is a linear prediction computational engine, the linear prediction (LP) order is passed as a generic. The ports used for input and output to the processor include its data and control lines. The data lines consist of 240 samples of speech data in the format "SpeechType" and the output data are the LP coefficients. The control input information indicates when the processor starts computing the coefficients and the output control line indicates to the test bench when to read the results.

Additional information passed to this entity include some of the physical constraints from the previous slide such as cost, weight, etc. Additional information could be included such as power, test inputs, etc.

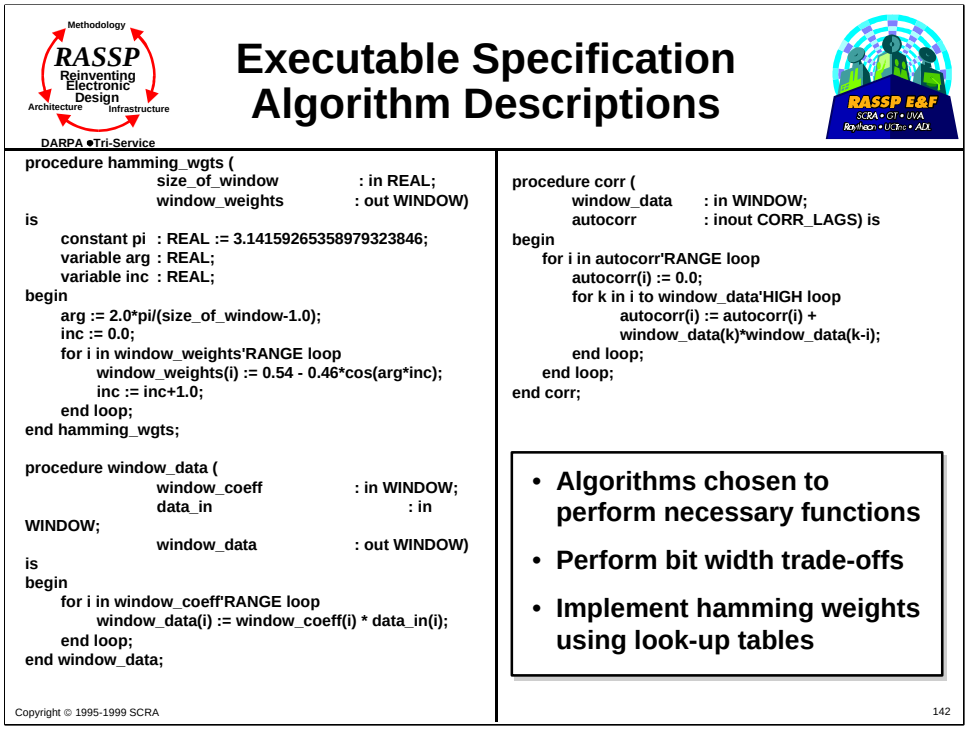
This example shows the case where physical parameters were added to the model entity description.



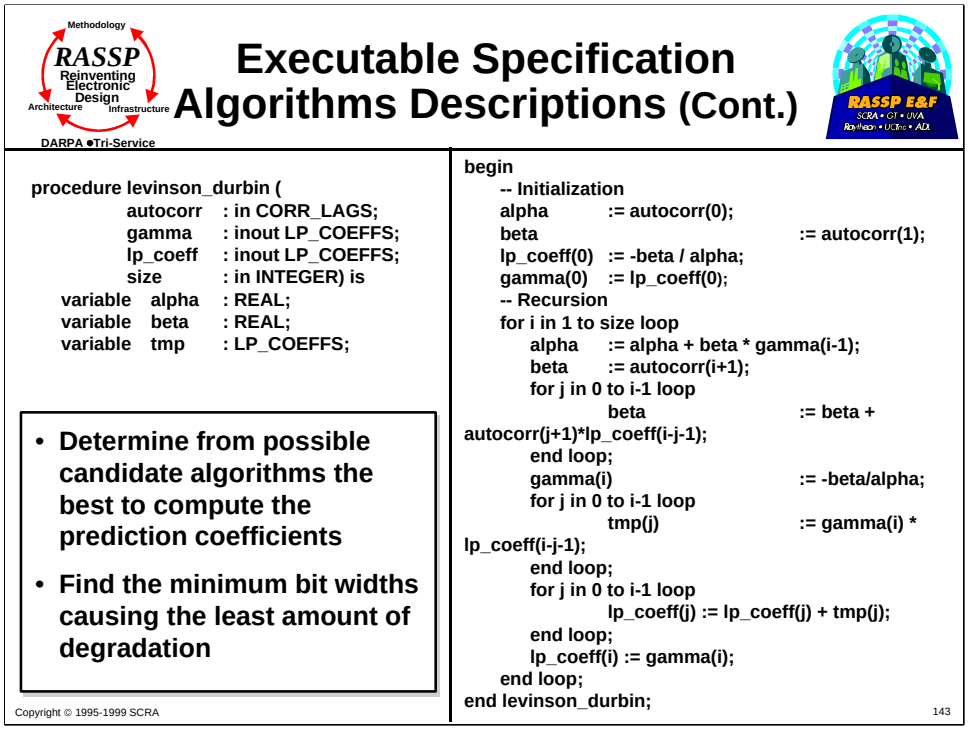
The architecture describing the functionality of the signal processor is shown above. It consists of one process executed in zero time and sensitive to the "start_processor" signal from its input port. When it receives this signal, the speech data is assumed to present and the processor begins calculating the LP coefficients. It calls a sequence of procedures to perform the necessary functions (window_data, corr, levinson-durbin). When it has finished, it puts the results on the lpCoef lines and sets the trigger to the testbench.

We include additional information at this stage by defining the process flow, functionality and the task breakdown. Fixed/floating point decisions are made at this point by determining the optimal bit widths to do the computations without loosing the quality of the speech data.

The physical constraint information can be added to global signals to keep information on cost, weight, etc. when there are other components in the system contributing to the total.

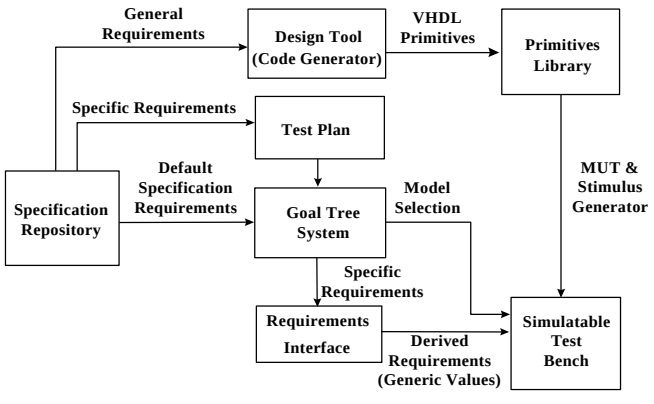


Also in this stage, we define how each of the algorithms are to be implemented. The hamming weights would most likely be implemented in a lookup table but computed from the equations above and rounded to the amount of digits required. The procedures for windowing data and computing the autocorrelation are also defined.



There are many algorithms for computing the linear prediction coefficients. In this design stage, we determine the best algorithm to use for the application. In this case, the levinson-durbin algorithm was chosen due to its efficient method for computation.

Example3: A Test Planning System used for the SAR model



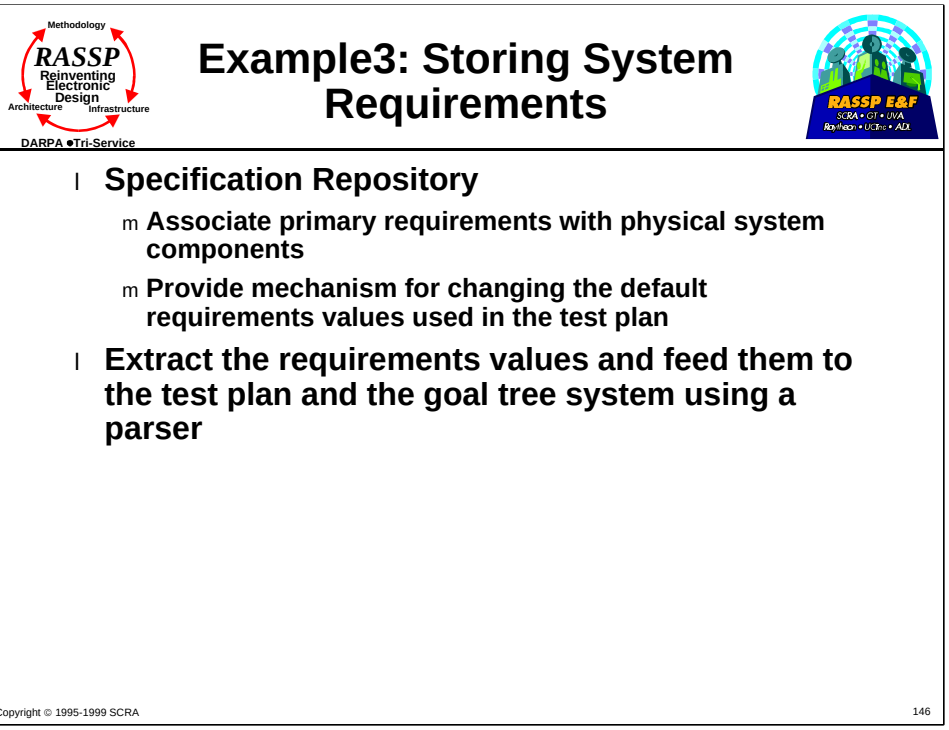
144

See [Armstrong94] is there a library for each class of system.

Example3: An Example Test Bench Requirement

- **The SAR processor test bench must generate a sequence of samples for the returned 'linear FM pulses' at a fixed sampling rate (general requirement)**
- **Specific requirements specify the values of the system parameters**
  - **Linear FM pulses is a system parameter that implies several system parameters including bandwidth, pulse width, carrier frequency and the pulse repetition rate, whose values form specific requirements**

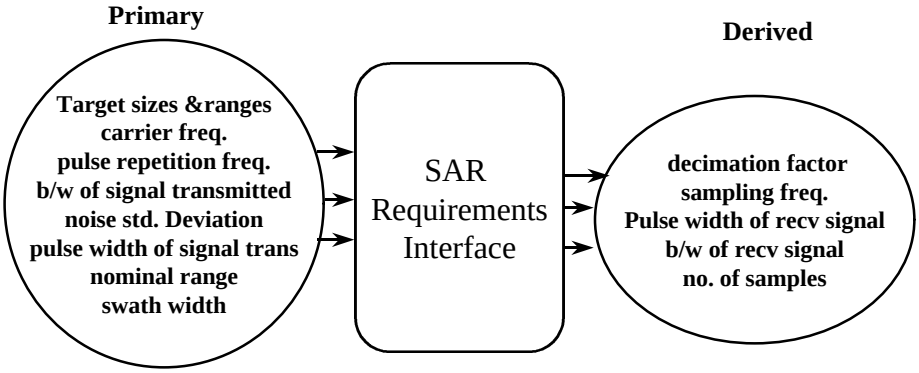
[Armstrong] Test Planning for CHDL DSP models



A specification repository is a system level block diagram, where the blocks correspond to the physical system components with associated primary requirements. The system diagram can be developed using schematic capture tools such as SGE. With SGE, the primary requirements are represented as symbol attributes of their associated blocks. One can click the mouse on a block and an associated window then pops up, where the primary requirements can be edited and altered. A parser extracts the requirements values and feeds them forward to the test plan and the goal tree system. The specification repository provides a mechanism for the test engineer to change the default requirements values used in the test plan.

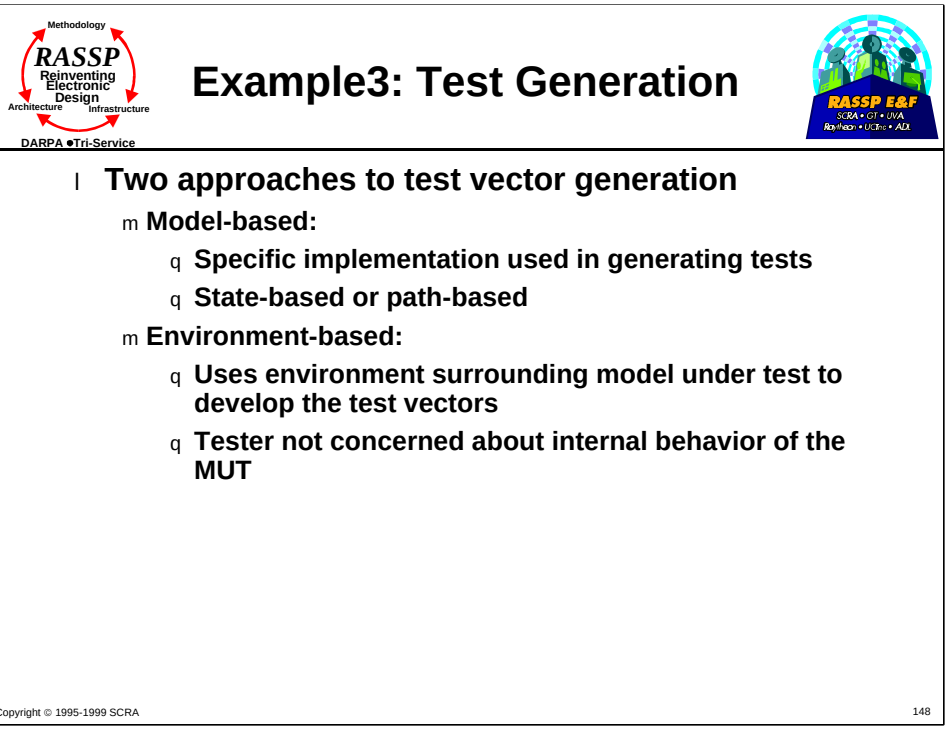
Example3: Linking Using the Requirements Interface for SAR

- **Link the test plan to the test bench using the Requirements Interface**
  - **The RI takes primary specific requirements from the test plan and produces the derived requirements as output**



147

Here we describe the testbench descriptions (primary and derived) within the context of executable requirements/specifications process.

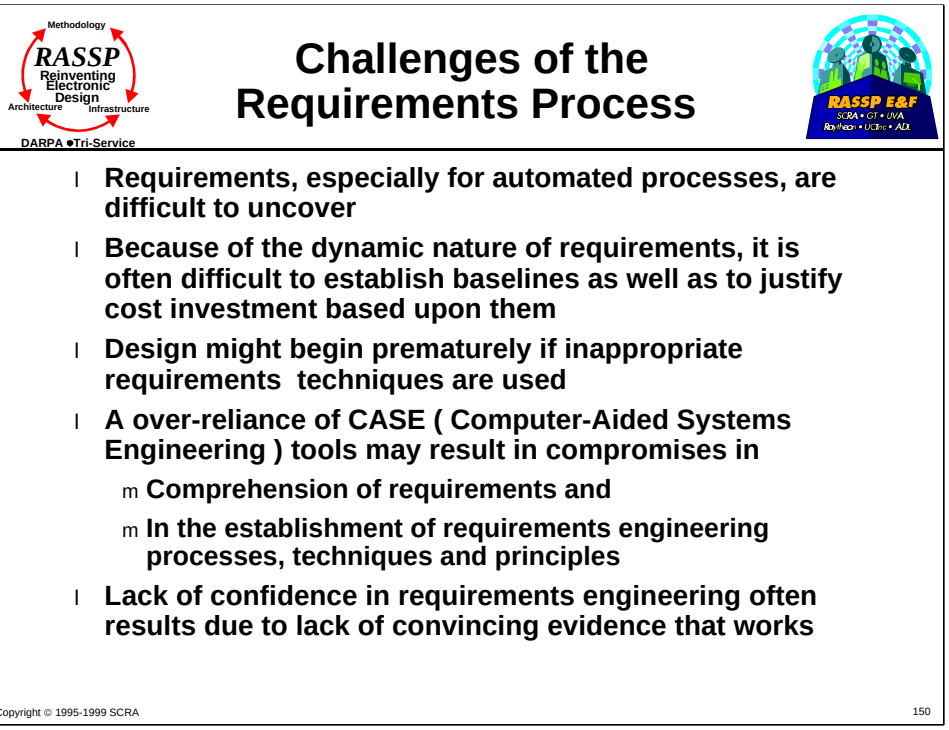


Test generation (TG) is the process of determining the stimuli to test a digital system. Test vectors can be developed manually, but for large system models this is an arduous task. A number of approaches to automatic test generation for hardware description language models have been developed recently. There are two approaches to test vector generation. One approach is model-based test generation. It is a form of white box testing, where the specific implementation of the model under test is used to generate the tests. The model-based approach includes two types: state-based testing and path-based testing. In state-based testing, the state of the MUT is characterized by the state of its internal signals and variables. Stuck-at-fault testing belongs to this type. Path-based testing looks at the control structure of the MUT and then develops test vectors to ensure that all paths through the code for the system are executed during a test session. A second approach to test vector generation is environment-based test generation as it uses the environment surrounding the model under test to develop the test vectors. It is a form of black box testing since the tester is completely unconcerned about the internal behavior and structure of the MUT. This later approach employs environment-based test generation as the type of inputs experienced by the signal processing models are a complicated function of the model environment and could not be directly generated from the models themselves.

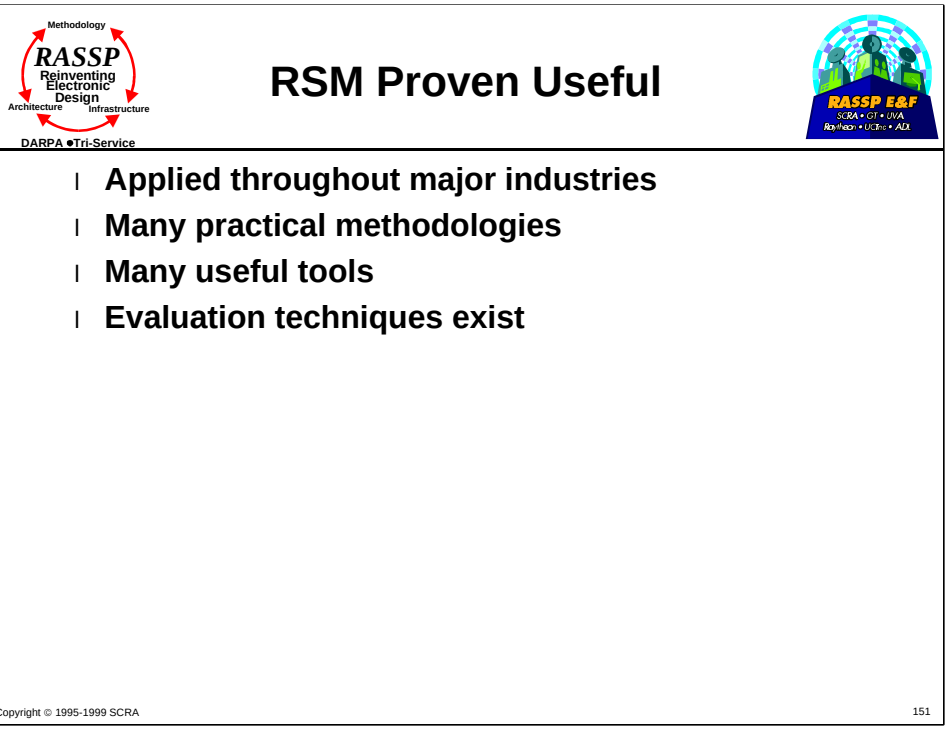
RSM Outline

- The Motivation for RSM
- Requirements in the Systems Definition Phase
- Requirements Engineering and Requirements Analysis
- Specification Modeling Methodologies
- SMM Survey
- CASE Tools
- RASSP Requirement Capture and Test Planning
- **Summary**

We now summarize the contents of this module.



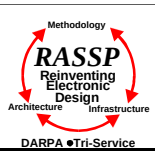
The main challenges of the requirements process as described in this module are recited above.



A number of RASSP benchmarks and case studies show detailed implementation of the RSM methodology as described in this module.

References

- [ARMSTRONG94] Armstrong, James R., Frank, Geoffrey. "Test Bench Development for RASSP DSP Models", 1st Annual RASSP Conference Proceedings, August 15-18, 1994.
- [CY90] Coad P., Yourdon E. *Object-Oriented Analysis*. Prentice-Hall, 1990.
- [Pnu86] Pnueli A. "Application of temporal logic to the specification and verification of reactive systems." Current Trends in Concurrency. Lecture Notes in Computer Science. Eds: de Bakker et al. Vol. 224, No. 9. Sep 1992.
- [Sarkar94] A., Waxman, R. and Cohoon, J.P.,"Specification Modeling Methodologies for Reactive System Design," University of Virginia, Departmental Report.
- [SM88] Schlaer S., Mellor S.J. Object-Oriented Systems Analysis. Yourdon Press, 1988.
- [IEEE] All referenced IEEE material is used with permission.
- [IEEE1220] IEEE Standard 1220-1994 Trial-Use Standard for Application and Management of the Systems Engineering Process
- [Lincoln3] "Executable Requirements: Opportunities and Impediments," MIT-LL RASSP Program, Anderson A.H.
- [LMC-Meth] RASSP Methodology Version 2.0, Lochkeed Martin ATL
- [Hsia] "Status Report- Requirements Engineering", Hsia, Davis and Kung



Additional Reading

- Alford M.W., Ansart J.P., Hommel G., Lamport L., Liskov B., Mullery G.P., Schneider F.B. Distributed Systems. Methods and Tools for Specification. Lecture Notes in Computer Science, Springer-Verlag 1982.
- Aylor J. H. and Waxman R. and Johnson B.W. and Williams R.D. The Integration of Performance and Functional Modeling in VHDL. In Performance and Fault Modeling with VHDL. Schoen, J. M., Prentice Hall, Englewood Cliffs, NJ 07632, 1992, pages 22-145.
- Aylor J. H., Williams R. D., Waxman R., Johnson B. W., and Blackburn R. L. A Fundamental Approach to Uninterpreted/Interpreted Modeling of Digital Systems in a Common Simulation Environment. UVA Technical Report TR # 900724.0, University of Virginia, Charlottesville, USA, July 24, 1990.
- Budkowski S. and Dembinski P. An Introduction to Estelle: A Specification Language for Distributed Systems. Computer Networks and ISDN Systems. North-Holland, Vol. 14, 1987.
- Belina F., Hogrefe D., and Sarma A. SDL with Applications from Protocol Specifications. Prentice Hall, 1991.

Detailed references provide additional material related to the topic of this module.

Additional Reading

- Bjorner D., Jones C.B. The Vienna Development Method: The Meta-Language. Lecture Notes in Computer Science. No. 61. Springer-Verlag. 1978.
- Blackburn R. L., and Thomas D. E. Linking the Behavioral and Structural Domains of Representation in a Synthesis System. DAC 85:374-380.
- Calvez J.P. Embedded Real-Time Systems: A Specification and Design Methodology. Wiley Series in Software Engineering Practice. 1993.
- Cameron J.R. "An overview of JSD". IEEE Transactions on Software Engineering. Vol SE-12. No. 2. February 1986.
- CCITT. Recommendation Z.100: Specification and Description language (SDL). Volume X, Fascicle X.1, Blue Book, October 1988.
- Coleman D. "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design". IEEE Transactions on Software Engineering. Vol. 18, No. 1. Jan 1992.
- Chu C. M., Potkonjak M., Thaler M., and Rabaey J. HYPER: An Interactive Synthesis Environment for High Performance Real Time Applications. Proceeding of the International Conference on Computer Design, pages 432-435, 1989.

Additional Reading

- Davis A. " A Comparison of Techniques for the Specification of External System Behavior". Communications of the ACM. Vol 31, No. 9. 1988.
- DeMarco T. Structured Analysis and System Specification. Yourdon Computing Series, Yourdon Press, Prentice Hall, 1979.
- Dutt N. D. and Gajski D. D. Designer Controlled Behavioral Synthesis Proceedings of the 26th Design Automation Conference, pages 754-757, 1989.
- Drusinsky D. and Harel D. Using Statecharts for hardware description and synthesis. In IEEE Transactions on Computer-Aided Design, 1989.
- Dijkstra E.W. "Guarded commands, nondeterminacy, and formal derivation of programs". Communications of the ACM, Vol 18, No. 8. 1975.
- Ehrig H. and Mahr B. Fundamentals of Algebraic Specification - 1. EATCS Monographs on Theoretical Computer Science 6. Springer-Verlag. 1985.
- Feijs L.M.G. and Jonkers H.B.M. Specification and Design with COLD-K. LNCS 490, pp. 277-301.
- Fraser M.D., Kumar K., Vaishnavi V.K. "Strategies for Incorporating Formal Specifications in Software Development". Communications of the ACM. Vol. 37, No. 10. Oct 1994 p 74-86.
- Færgemand O. and Olsen A. New Features in SDL-92. Tutorial, Telecommunications Research Laboratory, TFL, Denmark. 1992.

Additional Reading

- Gajski D.D., Vahid F., and Narayan S. A system-design methodology: Executable-specification refinement. In Proceedings of the European Conference on Design Automation (EDAC), 1994.
- Halbwachs N. Synchronous Programming of reactive Systems. Kluwer Academic Publishers, 1993.
- Harel D. "Biting the Silver Bullet: Toward a Brighter Future for System Development". IEEE Computer. Vol. 25, No. 1. Jan 1992, pages 8-24.
- Harel D. "On Visual Formalisms". Communications of the ACM. Vol. 31, No. 5. 1988 p 514-530.
- Harel D. "Statecharts: A Visual Formalism For reactive Systems". Science of Computer Programming, Vol 8. 1987, pages 231-274.
- Hu D.C. and DeMicheli G. HardwareC - a language for hardware design. Stanford University, Technical Report CSL-TR-90-419, 1988.
- Harel D., Lachover H., Naamad A., Pnueli A., Politi M., Sherman R., Shtull-Trauring A., Trakhtenbrot M. "Statemate: A working environment for the development of reactive systems". Proceedings of 10th International Conference on Software Engineering. Singapore, 11-15 April 1988, p 122-129.
- Hatley D.J. and Pirbhai I.A. Strategies for Real-time System Specification. Dorset House Publishing, New York, 1987.

Additional Reading

- IEEE Standard VHDL Language Reference Manual. IEEE Inc., NY, 1988.
- i-Logix Inc. ExpressVHDL Documentation, Version 3.0. 1992.
- ISO/IS 8807. Information Processing Systems - Open Systems Interconnection: LOTOS - A Formal Description Technique. 1989.
- Jackson M.A. System Development. Prentice-Hall, 1983.
- Jackson M.A. Principles of Program Design. Academic Press, 1975.
- Lor K. E. and Berry D. M. Automatic Synthesis of SARA Design Models from System Requirements. IEEE Transactions on Software Engineering 17(12):1229-1240 December 1991.
- Lavi J.Z. and Winokur M. "Embedded computer systems: requirements analysis and specification: An industrial course". Proceedings of SEI Conference, Virginia April 1988. Lecture Notes in Computer Science: Software Engineering Education, No. 327. Ed. G. A. Ford, Springer-Verlag p 81-105.
- Maraninchi F. Argos: A graphical synchronous language for the description of reactive systems. Report RT-C29, Univeriste Joseph Fourier, 1991.

Additional Reading

- Monarchi D.E. and Puhr G.I. "A Research Typology for Object-Oriented Ananlysis and Design". Communications of the ACM. Vol. 35, No. 9. Sep 1992.
- Manna A. and Pnueli A. The temporal logic of reactive and concurrent systems: specification. Berlin Heidelberg New York: Springer. 1991.
- Peterson "Role of Executable Specifications", Capt.Greg Peterson, Wright Lab
- Saracco R., Smith J., Reed R. *Telecommunication Systems Engineering using SDL*. Elsevier Science Publishers. 1989.
- Sarkar A. An Integrated Specification and Performance Modeling Approach for Digital System Design. Ph.D. Thesis. University of Virginia. Charlottesville, U.S.A. 1995.
- Sarkar A., Waxman, R. and Cohoon, J.P.,"Specification Modeling Methodologies for Reactive System Design,", which appears in the book: *High-Level System Modeling: Specification Languages*, Edited by Berge, Levia, & Rouillard, Kluwer, 1995, pp 1-34.
- Sarkar A., Waxman R., and Cohoon J.P. "System Design Utilizing Integrated Specification and Performance Models". Proceedings, VHDL International Users Forum, Oakland, California, May 1-4, 1994, pp 90-100.
- Spivey J.M. Understanding Z: *A specification Language and its Formal Semantics*. Cambridge University Press. 1988.

Additional Reading

- Thomas D.E. and Moorby P. The Verilog Hardware Description Language. Kluwer Academic Publishers, 1991.
- Wing J.M. "A specifier's introduction to formal methods". IEEE Computer, Vol 23, No. 9. 1990. pp 8-24.
- Woo N., Wolf W., and Dunlop A. Compilation of a single specification into hardware and software. AT&T Bell Labs, 1992.
- Ward P.T., Mellor S.J. Structured Development for Real-time Systems, Vols 1 & 2. Yourdon Computing Series, Yourdon Press, Prentice Hall 1985.
- Zave P. and Shell W. Salient features of an executable specification language and its environment. IEEE Transactions on Software Engineering 12(2):312-325 Feb, 1986.

Additional Reading

- EIA Standard 632 for Systems Engineering
- A Systems Engineering Process Model: Conceptual Approach
- Systems Engineering-Capability Maturity Model, Software Engineering Institute, Carnegie Mellon University, url http://www.sei.cmu.edu/publications/documents/95.reports/95.mm.003.html
- Dept. of Air Force Acquisition Risk Management Guide
- High-Level System Modeling Specification & Design Methodologies, Edited by Waxman, Berge, Levia, & Rouillard, Kluwer, 1996
- "Embedded Real-Time Systems - A Specification and Design Methodology", Jean Paul Calvez, John Wiley & Sons, 1993
- "Method Integration- Concepts and Case Studies", Edited by Klaus Kronlof, John Wiley&Sons, 1993

Additional Reading

- "Benchmark I Requirements", Alan H Anderson, MIT Lincoln Lab
- "Executable Requirements Benchmarking," Alan H Anderson, MIT Lincoln Lab
- EIA-IS 632 Standard
- "Rapid Digital System Prototyping-Current practice, Future Challenges", V K Madisetti, IEEE D&T, Fall '96, pp18
- Richards M., Gadient A., Frank G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997
- Advanced Design Environment Prototyping Tool Tutorial, University of Virginia
- "Role of Executable Specifications", Capt. Greg Peterson, Wright Labs
- "Final Review", TRW, September 29, 1998.