Survey of Model-Based Systems Engineering (MBSE) Methodologies

Jeff A. Estefan
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California, U.S.A.
Jeffrey.A.Estefan@jpl.nasa.gov

1. Introduction

1.1 Purpose

The purpose of this report is to provide a cursory description of some of the leading Model-Based Systems Engineering (MBSE) methodologies used in industry today. It is intended that the material described herein provides a direct response to the INCOSE MBSE Roadmap element for a "Catalog of MBSE lifecycle methodologies" [1].

In this report, a *methodology* is defined as a collection of related processes, methods, and tools [2]. A MBSE methodology can be characterized as the collection of related processes, methods, and tools used to support the discipline of systems engineering in a "model-based" or "model-driven" context. The intent of this survey is to educate the reader about the various candidate MBSE methodologies that are commercially available as well as the control- and state-based MBSE methodology that has been developed at NASA's Jet Propulsion Laboratory (JPL), which has been published in the open literature.

1.2 Scope

This memo describes the result of a MBSE methodology survey only; it is not a methodology assessment. The material contained herein is expected to be reviewed and shared by the INCOSE MBSE Initiative team and its governing leaders. It should be noted that this is a cursory survey and only the top-level synopses of each candidate methodology is described. Detailed descriptions of each can be found in the cited references.

While it is recognized that modern day systems are not only gaining in overall complexity, they are also becoming more software intensive. Nevertheless, the scope of this survey report is on MBSE methodologies from the holistic, lifecycle wide systems engineering perspective and not specifically targeted toward embedded systems or software-intensive systems in general. There are some notable model-based methodologies that focus on embedded and software-intensive systems such as the Embedded Computer System Analysis and Modeling (ECSAM) methodology from Lavi and Kudish [3],[4] and Model-Based [System] Architecture and Software Engineering (MBASE) from Boehm and Port [5],[6]; however, these methodologies are not described herein. The interested reader can review the cited references at the end of this report for more information on these methodologies. In future revisions of this survey, the scope may expand to include model-based methodologies for embedded and software-intensive systems in addition to mainstream MBSE methodologies.

As will be described, tools are an important element of any MBSE methodology; however, a survey of MBSE tools is beyond the scope of this report. It is expected that during an organization's candidate MBSE methodology assessment process (including impact to native processes and procedures), a tool survey and assessment will occur concurrently or shortly thereafter, followed by selection and piloting of relevant tools. This latter effort requires participation from the organization's systems engineering practitioner community because that is the community that will most heavily be using the tools.

It is intended that this report be a living document and updated on a periodic basis based on feedback and input by not only members of the INCOSE MBSE Initiative team but also by members of the INCOSE community at large.

1.3 Overview

This report is organized as follows: Section 2 characterizes the difference between methodologies and processes, methods, and lifecycle models (development, acquisition, and systems engineering). Also described is the role of models in the systems engineering process and the seminal work by Wymore on the mathematical foundation of MBSE. Section 3 documents the survey results of leading MBSE methodologies used in industry. Section 4 describes the role of the Object Management Group™ (OMG™) Unified Modeling Language™ (UML®) and Systems Modeling Language™ (OMG SysML™), which are industry-standard, visual modeling languages used to support the disciplines of software and systems engineering, and how these modeling standards relate to MBSE methodologies. Section 5 discusses the role of OMG™ Model-Driven Architecture® (MDA®) to the discipline of systems engineering. In addition, the Executable UML Foundation is briefly introduced. Section 6 provides a list of references used in preparation of this survey report and for the benefit of the reader. Finally, Section 7 provides a list of acronyms and abbreviations used in this report.

2. Differentiating Methodologies from Processes, Methods, and Lifecycle Models

In order to better understand key features of the various leading MBSE methodologies surveyed in this study, it is critically important to establish the terminology associated with processes, methods, and methodology, and to acknowledge the myriad lifecycle models used in the acquisition and development of large-scale, complex systems. Without such grounding, it will be extremely difficult to map any assessment and selection of candidate MBSE methodologies into the fabric of the systems engineering environment within a particular organization.

2.1 Process, Method, Tool, Methodology, and Environment Defined

The word *methodology* is often erroneously considered synonymous with the word *process*. For purposes of this study, the following definitions from Martin [2] are used to distinguish methodology from process, methods, and tools:

- A *Process* (*P*) is a logical sequence of tasks performed to achieve a particular objective. A process defines "WHAT" is to be done, without specifying "HOW" each task is performed. The structure of a process provides several levels of aggregation to allow analysis and definition to be done at various levels of detail to support different decision-making needs.
- A *Method* (*M*) consists of techniques for performing a task, in other words, it defines the "HOW" of each task. (In this context, the words "method," "technique," "practice," and "procedure" are often used interchangeably.) At any level, process

tasks are performed using methods. However, each method is also a process itself, with a sequence of tasks to be performed for that particular method. In other words, the "HOW" at one level of abstraction becomes the "WHAT" at the next lower level.

- A *Tool* (*T*) is an instrument that, when applied to a particular method, can enhance the efficiency of the task; provided it is applied properly and by somebody with proper skills and training. The purpose of a tool should be to facilitate the accomplishment of the "HOWs." In a broader sense, a tool enhances the "WHAT" and the "HOW." Most tools used to support systems engineering are computer- or software-based, which also known as Computer Aided Engineering (CAE) tools.

Based on these definitions, a *methodology* can be defined as a collection of related processes, methods, and tools. A methodology is essentially a "recipe" and can be thought of as the application of related processes, methods, and tools to a class of problems that all have something in common [7].

Associated with the above definitions for process, methods (and methodology), and tools is environment. An *Environment* (*E*) consists of the surroundings, the external objects, conditions, or factors that influence the actions of an object, individual person or group [2]. These conditions can be social, cultural, personal, physical, organizational, or functional. The purpose of a project environment should be to integrate and support the use of the tools and methods used on that project. An environment thus enables (or disables) the "WHAT" and the "HOW."

A visual graphic that depicts the relationship between the so-called "PMTE" elements (Process, Methods, Tools, and Environment) is illustrated in Figure 2-1 along with the effects of technology and people on the PMTE elements.

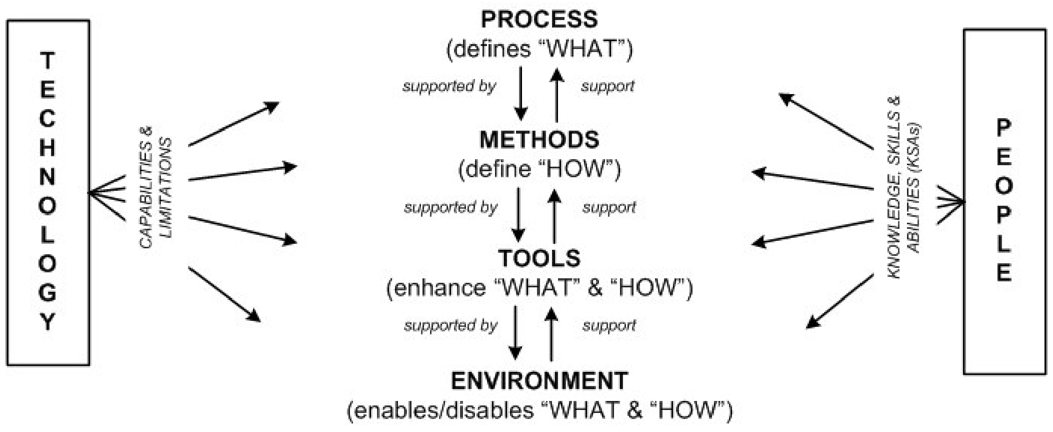


Figure 2-1. The PMTE Elements and Effects of Technology and People.

As stated by Martin [2], the *capabilities* and *limitations* of technology must be considered when developing a systems engineering development environment. This argument extends, of course, to an MBSE environment. Technology should not be used "just for the sake of technology." Technology can either help or hinder systems engineering efforts. Similarly, when choosing the right mix of PMTE elements, one must consider the *knowledge*, *skills* and *abilities* (KSA) of the people involved [2]. When new PMTE elements are used, often the KSAs of the people must be enhanced through special training and special assignments.

2.2 Lifecycle Development Models

A number of lifecycle development models have been created and applied to large-scale system and software development projects used in government, industry, and academia, but most are grounded in one of three seminal models. These are 1) Royce's *Waterfall Model* [8], Boehm's *Spiral Model* [9], and Forsberg and Moog's *"Vee" Model* [10],[11]. A graphical depiction of each of these lifecycle development models is shown in Figure 2-2.

There are large volumes of literature that describe each of these models; therefore, elaboration of each will not be provided here. Suffice it to say that variations of the waterfall and spiral models to support structured as well as iterative and incremental development have been used extensively in software development projects, while the "Vee" model and modified versions of the "Vee" have been applied extensively in the areas of systems engineering and systems development.

In addition to recognizing that such major lifecycle development models exist, they can also serve as *metamodels* for lifecycle development. In other words, they provide the lifecycle development templates on which project- or domain-specific plans are built. This will be more evident during the review of the various MBSE methodologies described in Section 3, many of which leverage one of these three lifecycle development models.

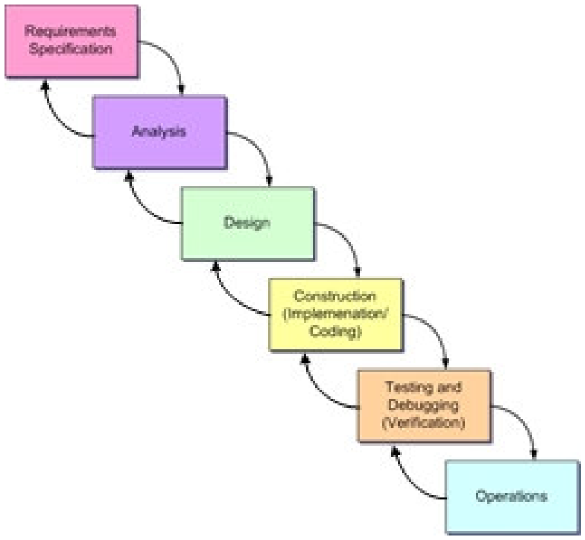
2.3 Acquisition Lifecycle Models

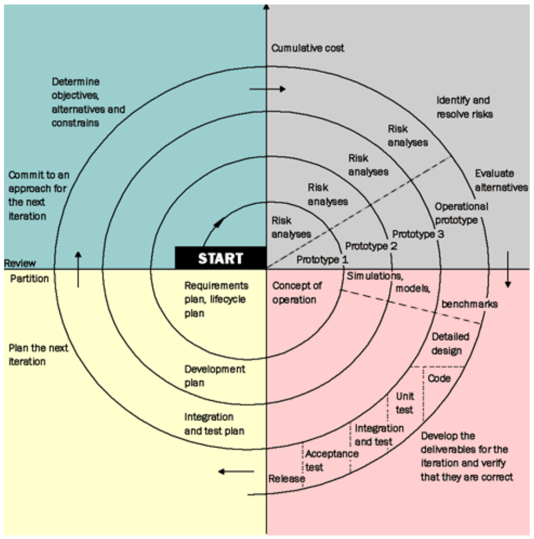
U.S. Government departments and agencies such as the U.S. Department of Defense (DoD) and the National Aeronautics and Space Administration (NASA) are responsible for managing billions of tax payer dollars annually in the development and acquisition of large-scale, complex systems. Consequently, these agencies must follow rigid acquisition guidelines to insure that they are good stewards of U.S. tax payer dollars, and that there is accountability for investment in such large-scale, potentially very costly programs.

DoD acquisition reform was instituted in May 2003 to help streamline the defense acquisition process, which in the past, was so onerous it took literally decades to field new weapons systems. DoD best practices for acquisition are rooted in DoD policy directives and instructions, namely, DoD Directive (DoDD) 5000.1 *The Defense Acquisition System* and DoD Instruction (DoDI) 5000.2 *Operation of the Defense Acquisition System* [12],[13]. DoD's revised acquisition policy includes a lifecycle framework and is depicted in Figure 2-3.

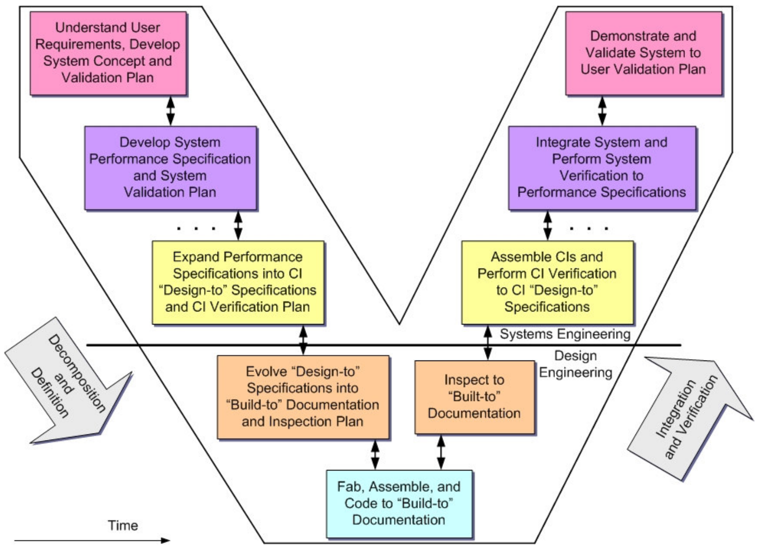
Milestone A represents the start of the development phase, Milestone B represents program start, and Milestone C represents production commitment. Milestones correspond to decision "gates" on which major programmatic decisions (e.g., funding) are made during gate review processes. IOC and FOC are abbreviations for Initial and Full Operational Capability, respectively. Further elaboration of the DoD acquisition lifecycle model will not be provided here. What is important to note for this report is that the acquisition model contains key lifecycle phases as well as decision milestones and gate reviews.

Similar to the DoD acquisition lifecycle model, the NASA lifecycle model has a set of key lifecycle phases as well as decision milestones and gate reviews (see Figure 2-4).


Requirements Specification
Analysis
Design
Construction (Implementation/ Coding)
Testing and Debugging (Verification)
Operations



(a) (b)


Understand User Requirements, Develop System Concept and Validation Plan
Develop System Performance Specification and System Validation Plan
Expand Performance Specifications into CI "Design-to" Specifications and CI Verification Plan
Evolve "Design-to" Specifications into "Build-to" Documentation and Inspection Plan
Fab, Assemble, and Code to "Build-to" Documentation
Inspect to "Built-to" Documentation
Assemble CIs and Perform CI Verification to CI "Design-to" Specifications
Integrate System and Perform System Verification to Performance Specifications
Demonstrate and Validate System to User Validation Plan
Systems Engineering
Design Engineering
Decomposition and Definition
Integration and Verification
Time

(c)

Figure 2-2. Seminal Lifecycle Development Models: (a) Waterfall, (b) Spiral, (c) "Vee".

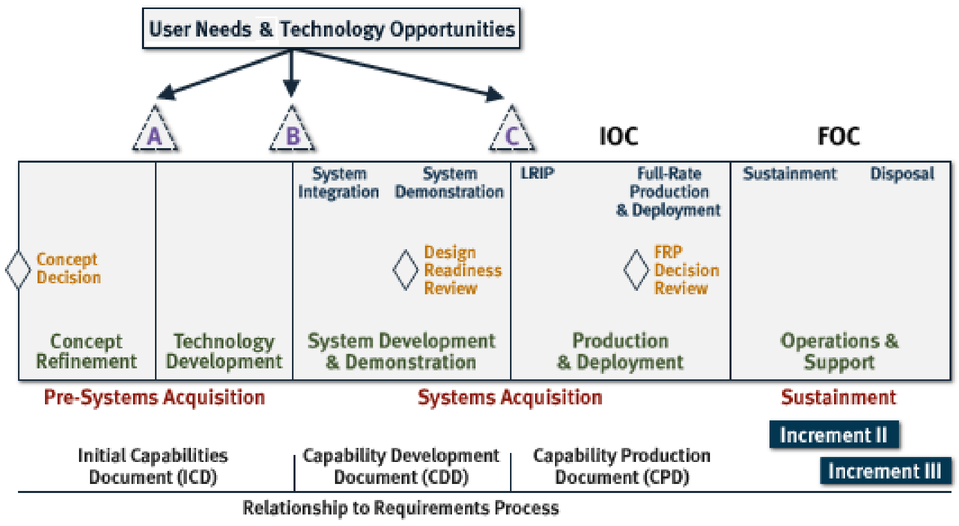


Figure 2-3. DoD Lifecycle Framework.

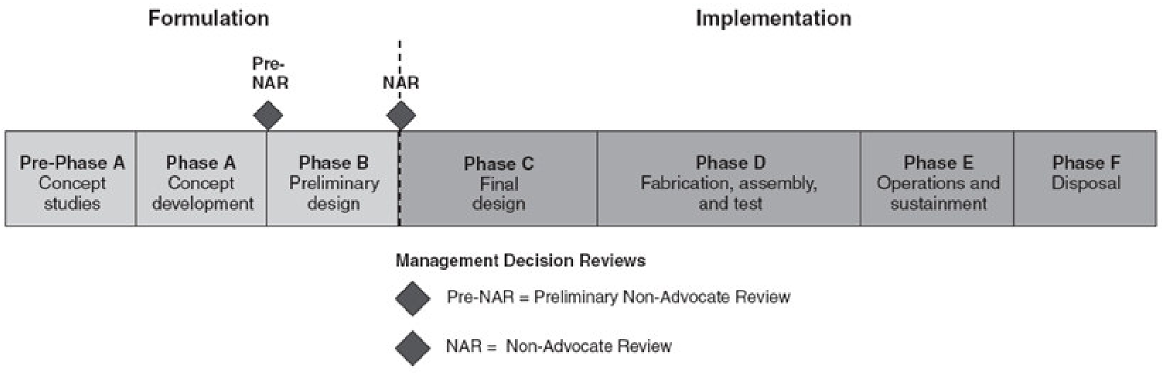

Formulation
Implementation
Pre-NAR
NAR
Pre-Phase A Concept studies
Phase A Concept development
Phase B Preliminary design
Phase C Final design
Phase D Fabrication, assembly, and test
Phase E Operations and sustainment
Phase F Disposal
Management Decision Reviews
Pre-NAR = Preliminary Non-Advocate Review
NAR = Non-Advocate Review

Figure 2-4. NASA Project Lifecycle.

NASA best practices for acquisition are rooted in NASA policy directives and requirements; specifically, NASA Policy Directive (NPD) 7120.4 *Program/Project Management* and NASA Policy Requirement (NPR) 7120.5 *NASA Program and Project Management Processes and Requirements* [14],[15]. Because NASA is a federal agency, programs the agency funds must also pass decision milestones and gate reviews to ensure programs are meeting cost, schedule, and technical baselines.

As with the development lifecycle models described in Section 2.2, the DoD and NASA acquisition lifecycle models captured here can be considered metamodels on which project- or domain-specific plans are built. Development lifecycles and acquisition lifecycles differ in many ways, but the critical difference between them is that development lifecycles can be applied one or more times during a single acquisition lifecycle.

One of the reasons for describing acquisition models as part of this MBSE survey is to acknowledge the heritage of these traditional, document-driven, programmatic reviews and the challenge organizations face when attempting to adopt more advanced, electronic- or model-driven techniques such as MBSE. Traditionally, acquisition program reviews have relied on paper documents, because that was the state-of-the-art at the time government acquisition lifecycle models were first initiated [16]. Advances in information technology over the last decade or so have afforded the opportunity to create "electronic" documents using Microsoft® Word and PowerPoint and Adobe® Acrobat®; however, such electronic resources are still often considered "hardcopy" document artifacts. This is evident as these artifacts are almost always printed on paper for members of review boards during decision milestone and gate reviews. Despite the fact that information technology has advanced to a point where the technology can easily support fully electronic- or model-driven programmatic reviews, the traditional document-driven approach is likely to continue for the foreseeable future. Therefore, whatever MBSE methodology and approach that is assessed and utilized by an organization will have to ultimately map back to the organization's project lifecycle and decision milestones and gates (and subsequently *gate products*) as part of the programmatic review process.

2.4 Systems Engineering Process Standards and Capability Models

A systems engineering (SE) process is a process model that defines the primary activities ("WHAT") that must be performed to implement systems engineering. SE processes are related to the phases in an acquisition lifecycle model in that the process usually begins at an early stage of the system lifecycle, typically the very beginning of a project; however, on some occasions, the SE process can also begin at the middle of an acquisition lifecycle.

A variety of SE process standards have been proposed by different international standards bodies, but most SE process standards in use today have evolved from the early days of DoD-MIL-STD 499. The heritage of these SE process standards together with industry standard capability models and the relationship between them is illustrated in Figure 2-5 [17]. Also shown is the relationship to relevant ISO/IEC software process standards.

The ANSI/EIA 632 *Processes for Engineering a System* standard [18] and the IEEE 1220-1998 *Standard for Application and Management of the Systems Engineering Process* [19] were sources into the creation of ISO/IEC 15288:2002 *Systems Engineering—System Life Cycle Processes* [20]. ISO/IEC 19760 *Guide for ISO/IEC 15288 — System Life Cycle Processes* is, as the name implies, a guidance document for ISO/IEC 15288.

The Institute for Electrical and Electronic Engineers (IEEE) has since standardized on ISO/IEC 15288 (which they refer to as IEEE Std 15288™-2004) [21]. In addition, the International Council on Systems Engineering (INCOSE) has announced a commitment to adoption of the 15288 standard, some of the elements of which have been integrated into the INCOSE Systems Engineering Handbook v3 [22].

Because all three full SE process standards are available and used in practice, it is important to at least acknowledge the distinction between them. A graphical depiction of the three full standards that illustrates their primary scope is shown in Figure 2-6.

NASA too has recognized the importance of these industry standards with elements referenced and incorporated into the recently ratified NASA NPR 7123.1A *Systems Engineering Processes and Requirements* [23]. The NPR distinguishes between the three industry standards as follows: "ANSI/EIA 632 is a commercial version that evolved from the never released, but fully developed, 1994 Mil-Std 499B. It was intended to provide a framework for developing and supporting a universal SE discipline for both defense and commercial environments. ANSI/EIA 632 was intended to be a top-tier standard further defined to lower-level tier standards that define specific practices. IEEE 1220 is a second-tier standard that implements ANSI/EIA 632 by defining one way to practice systems engineering. ISO/IEC 15288, on the other hand, defines system lifecycle processes for the international set, plus for any domain (i.e., transportation, medical, commercial, et al.).”

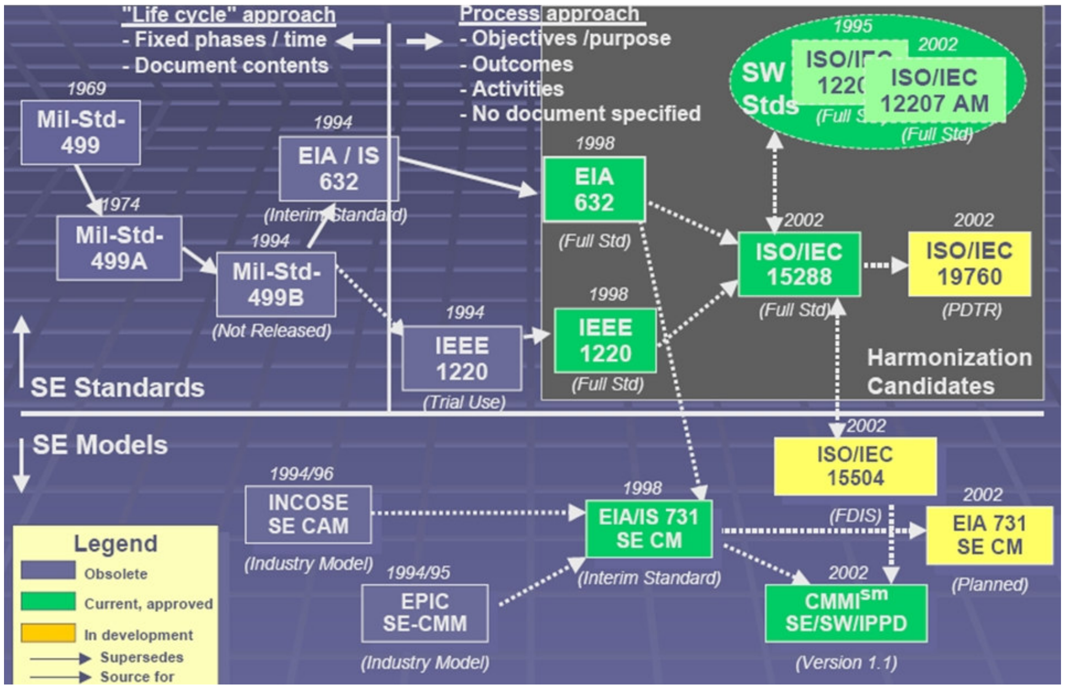


Figure 2-5. Heritage of Systems Engineering Process Standards and Capability Models.[1]

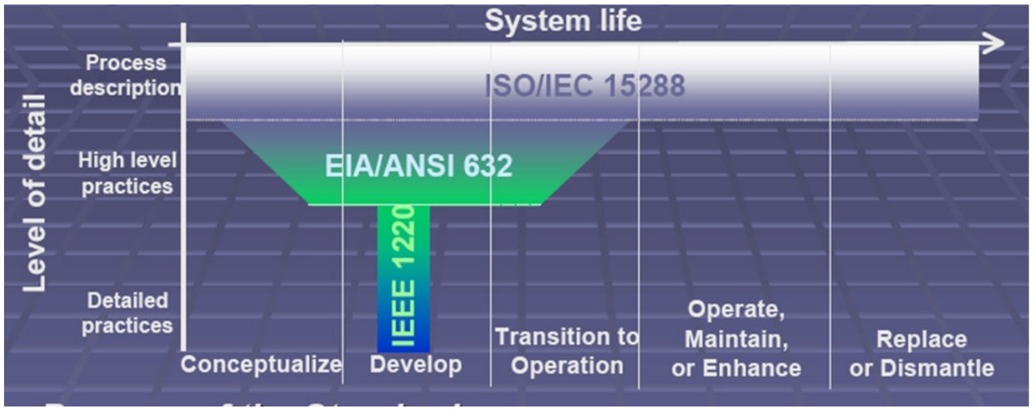


Figure 2-6. Breadth and Depth of Leading SE Process Standards.

As seen in Figure 2-6, the ISO/IEC 15288 standard follows more closely the acquisition lifecycle models that were described in Section 2.3. The 15288 Std. system lifecycle is

[1] Note that the status of some of these SE process standards and maturity models is somewhat dated since the source of this diagram was extracted from a G. Roedler briefing dated Sep. 17, 2002 [17]. In ISO/IEC terms, PDTR stands for *Preliminary Draft Technical Report* and FDIS stands for *Final Draft Technical Standard*; ISO/IEC 19760 has since been released as a final technical report [*Source*: Michael Gayle, Jet Propulsion Laboratory (private communication), Mar. 16, 2007].

shown in Figure 2-7 while system lifecycle process elements of the 15288 Std. are captured in Figure 2-8.

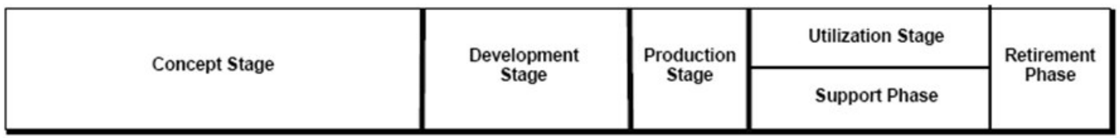


Figure 2-7. ISO/IEC 15288 System Lifecycle.

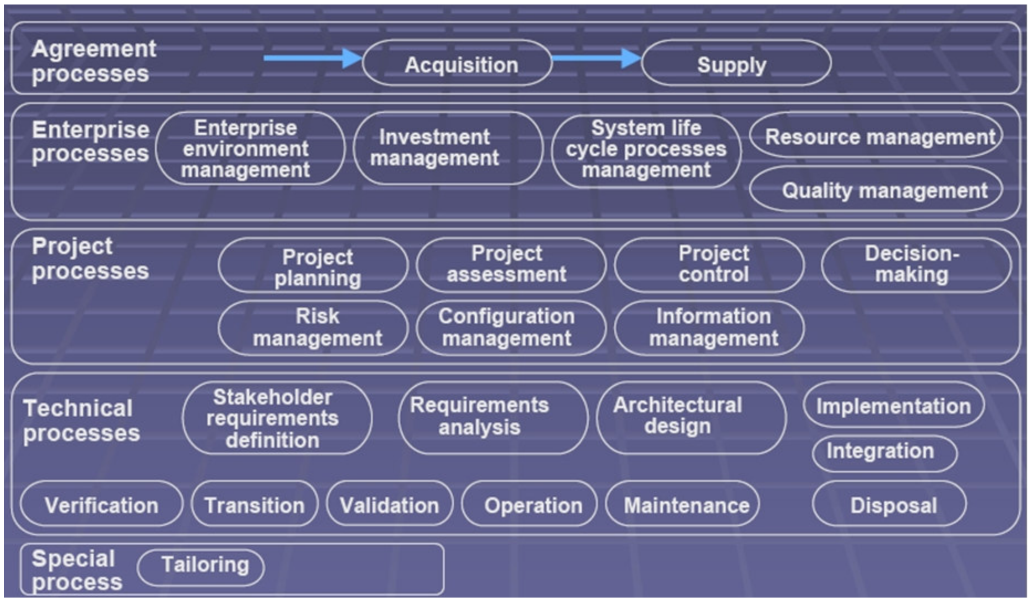


Figure 2-8. ISO/IEC 15288 Process Elements.

The *purpose* of each major SE process model standard can be summarized as follows [17]:

- ISO/IEC 15288 – Establish a common framework for <u>describing the lifecycle of systems</u>.
- ANSI/EIA 632 – Provide an integrated set of fundamental <u>processes to aid a developer</u> in the engineering or re-engineering of a system.
- IEEE 1220 – Provide a standard for <u>managing a system</u>.

Indeed, the IEEE 1220 provides useful guidance on developing a Systems Engineering Management Plan (SEMP), and a template is provided in Annex B of the standard. The NASA NPR 7123.1A also provides useful guidance on preparation of a SEMP. The NPR defines a SEMP as providing "the specifics of the technical effort and describes what technical processes will be used, how the processes will be applied using appropriate activities, how the project will be organized to accomplish the activities, and the cost and schedule associated with accomplishing the activities." Relative to the NASA acquisition lifecycle, the SEMP is used to "establish the technical content of the engineering work early in the Formulation Phase for each project and updated throughout the project life cycle."

2.5 Models in Support of SE Processes

In a nutshell, model-based engineering (MBE) is about elevating models in the engineering process to a central and governing role in the specification, design, integration, validation, and operation of a system. For many organizations, this is a paradigm shift from traditional document-based and acquisition lifecycle model approaches, many of which follow a "pure" waterfall model of system definition, system design, and design qualification. One of the biggest communication barriers that exists between the traditional engineering design disciplines (including the discipline of systems engineering) and MBE is that in a model-based process, activities that support the engineering process are to be accomplished through development of increasing detailed models. Skipper suggests that this communication chasm has existed for years and many managers and practitioners still do not identify with the fact that various MBE process models and supporting methodologies are intended to show *emphasis* rather than be purely waterfall, and that the entire system model grows over time (see Figure 2-9).[2]

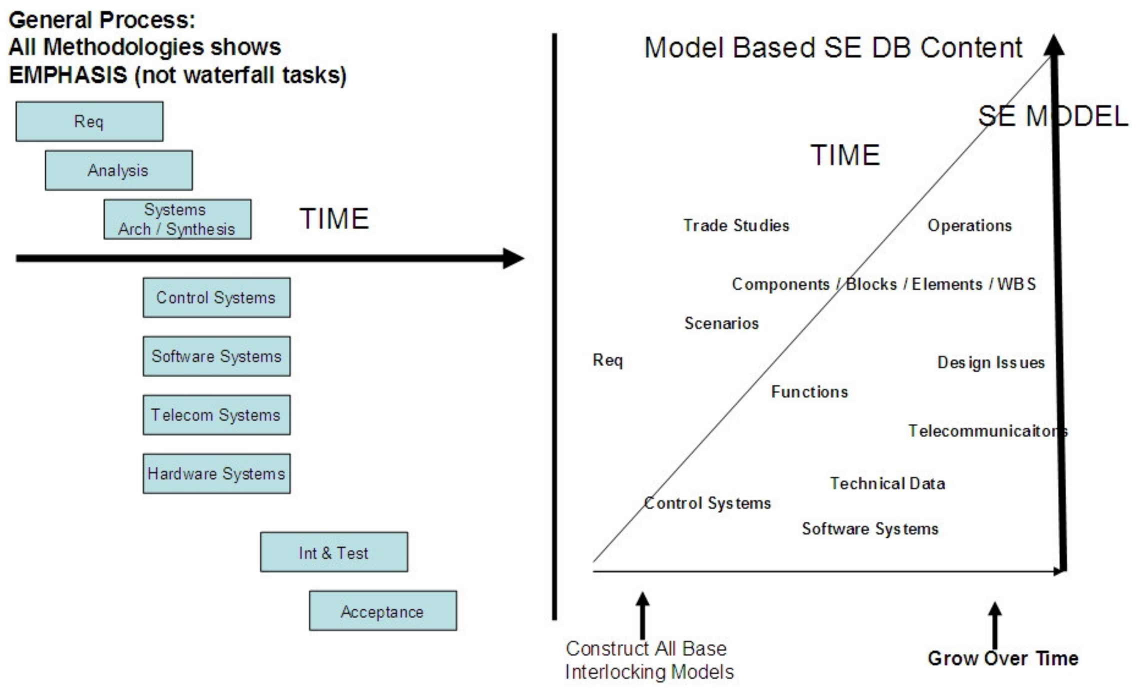


Figure 2-9. Generic SE Process and Integrated Model (Entire Model grows over Time, Not "Pure" Waterfall).

Baker et al. [24] articulate some of the key foundational concepts of model driven system design (MDSD) and contrast the model-driven approach with standard SE process models; in this case, the SE process model specified by the IEEE 1220 standard.[3] The authors suggest that basic sub-processes apply to each of the major development phases of a project (i.e., system definition, preliminary design, detailed design, and design qualification)

[2] Joseph Skipper, Jet Propulsion Laboratory (private communication), California Institute of Technology, Apr. 6, 2007.

[3] Some authors use the term "MDSD" (Model-Driven System Design) and other use MBSE (Model-Based Systems Engineering). While subtleties exist between the two terms, the latter is primarily used in this report and any reference to MDSD is intended to be synonymous with MBSE.

and that MDSD the basic sub-processes are repeated as many times as necessary. An illustration of the basic sub-processes for MDSD is shown in Figure 2-10.

The authors proceed to describe various distinctive features of MDSD for each of the four major development phases of the project. The interested reader is encouraged to review these features in the cited reference as they will not be repeated here.

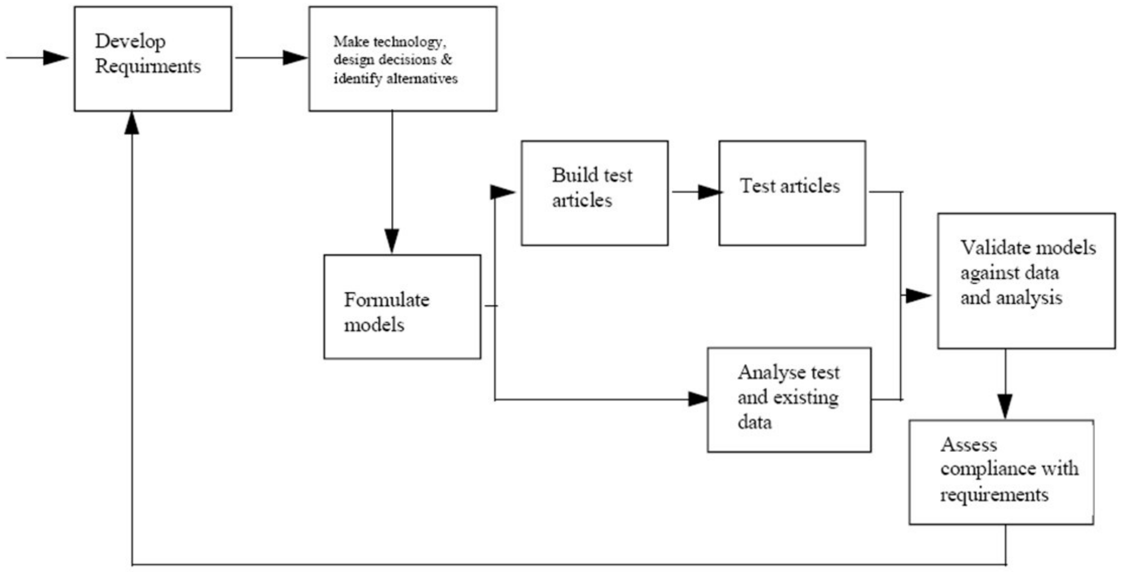

Develop Requirments
Make technology, design decisions & identify alternatives
Formulate models
Build test articles
Test articles
Analyse test and existing data
Validate models against data and analysis
Assess compliance with requirements

Figure 2-10. Sub-Processes for MDSD.

Another important concept that is introduced in the Baker et al. paper [24] is the notion of an *information model* for MDSD, which is illustrated in Figure 2-11.

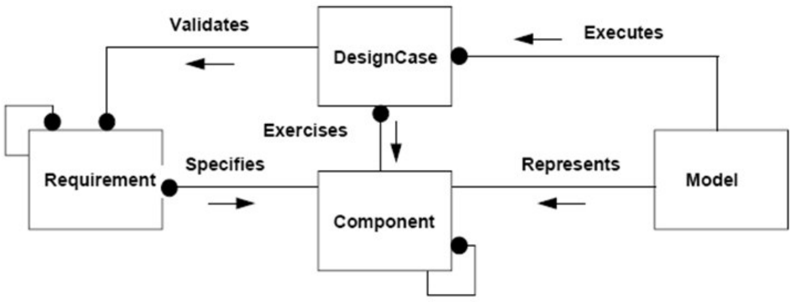


Figure 2-11. Information Model for MDSD.

Boxes show kinds of information, lines represent relationships, arrows show the direction of the relationship (not the direction of information flow), and bullets show a "many" relationship. The diagram elements can be interpreted as follows:

- Requirements specify Components
- Requirements may be decomposed into other Requirements
- Components may be decomposed into other Components

- ➢ Design Alternates satisfy Requirements
- ➢ Design Alternates represent Components
- ➢ Models execute Design Alternates
- ➢ Models represent Components

An information model is a very important part of MDSD as it facilitates the ability to view MDSD from the kinds of "information" to be used in such an approach and their relationships. Once again, a concurrent, incremental process is encouraged in which, as Baker et al. state, "in early states, the models are low fidelity and geared towards decision making; eventually, models become sufficiently faithful for compliance assessment" [24].

Also described in the cited paper is a useful and insightful contrast between document-centered system design and MDSD.

2.6 Mathematical Foundation of MBSE

It is difficult to cover the subject of MBSE and not acknowledge the seminal contributions of A. Wayne Wymore to the mathematical foundations of systems science and systems engineering [25]. Although the mathematical theory described in Wymore's early work is not for the faint of heart, his theory as applied to MBSE—published in the book entitled *Model-Based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*—establishes a rigorous mathematical framework for the statement of the problem of the design of a system in a model-based context [26].

According to Wymore's autobiographical retrospectives, the work described in his MBSE book does not provide a mathematical structure which could lead to the actual design of the system [27]; however, it is suggested that body of work is currently under development as "Phase 2" of MBSE (MBSE2) under the name *System Functional Analysis and System Design*. The interested reader need not wait for publication of MBSE2 in order to obtain exposure to a pragmatic perspective to system design and modeling that leverages some of the core Wymorian theory and principles. For this, the reader should consider the book by Chapman, Bahill, and Wymore entitled *Engineering Modeling and Design* [28].

The basis of system theory and systems engineering, according to Wymore, is *modeling*, and the concept of "system" is human interpretation via senses (i.e., a mental model) [25],[29]. A *system model* is a description that separates the perceived universe into two parts, the part "inside" the system and the part "outside" the system. *From* the "outside" the system receives inputs. *To* the "outside" the system delivers outputs. The "inside" of the system is described, initially, as states. The state of the system at any time is a function of its state at a previous time and the intervening inputs (including "noise"). System designs are system models. When modelers describe some part of reality as a "real" system, it means that their system mode "adequately" represents the reality. For the purpose of accurate communication, mathematical definitions of various classes of system models are postulated [29].

In Wymore's MBSE book, he starts by providing a discussion of the definition of systems engineering as well as a non-mathematical description of systems engineering process before embarking on the core mathematical theory. The mathematical theory presented is restricted to discrete time system models were chosen primarily for pedagogical reasons. According to Wymore's autobiographical account, "restrict[ing] all of MBSE to discrete time and that the basic set of system models would be the set of sequential machines with Moore readout functions. These decisions freed me from topological considerations occasioned by aspects of continuous time to allow me to concentrate on the mathematical structures of the objects of interest to systems engineers" [27].

A brief review of the book and its organization is described by Klir [30]. According to the Preface, "The material in the text was chosen to attain three principal objectives:

- to provide the system theoretic foundations necessary to the study and practice of systems engineering,
- to explicate mathematical system theory as the basis for the development of models and designs of large-scale, complex systems consisting of personnel, machine and software, and
- to introduce the student to the tricotyledon theory of system design (T3SD) and to the considerations involved in applying the theory to the design of real systems."

The phrase *tricotyledon theory* was chosen by Wymore to name the specific mathematical system theory he developed to facilitate the process of system design.[4] The three basic spaces of system design are described shortly. The notation employed in the tricotyledon theory and examples of systems described in the book is often referred to as "Wymorian notation" [31].

Following the introduction of systems engineering and systems, the MBSE book provides a rigorous mathematical foundation of discrete time system models, aspects of discrete time models such as causality and time invariance, system coupling and resultants, system homomorphisms (of like "shape"), and system modes [26].

A key aspect that is elaborated on in the second part of the MBSE book is Wymore's introduction of T3SD and identification of the six core categories of system design requirements (SDR), which he defines as follows:

SDR = (IOR, TYR, PR, CR, TR, STR) where

IOR is the I/O requirement,

TYR is the technology requirement,

PR is the performance requirement,

CR is the cost requirement,

TR is the trade-off requirement, and

STR is the system test requirement.

[4] The term *cotyledon* (as opposed to tricotyledon) is actually a botanical term used to represent structure in the embryo of a seed plant that may form a leaf after germination, what is commonly known as a 'seed leaf.' Wymore comments that the term was adopted during early development of his theory when visual caricatures were drawn to help conceptualize the spaces of system designs of various kinds. These caricatures or cartoons took on a leaf shape and an (unnamed) student suggested use of the term "cotyledons" (or seed leaves) because they were generated by the statement of the problem of the design of a system and the ultimate system design would eventually "flower" from these cotyledons. Since there are three basic spaces of system designs, the theory came to be known as the tricotyledon theory of system design, or T3SD for short.

According to Wymore, across all projects, the system has to verify that these six conditions are met. This has direct relevance to the discipline of systems engineering. Systems engineering considers as many alternative implementable system designs as possible and selects the best with respect to the tradeoff requirement, finding the implementable systems design that is *optimum* with respect to the tradeoff requirement and most likely to pass the system test, if possible [26],[29].

Mathematical foundation and visual depiction of each of these six categories that comprise the SDR is further elaborated in the remaining chapters of MBSE book. Application of the T3SD theory suggests that the statement of a system design problem requires the definition of three orders, one each over the *functionality*, the *buildability*, and the *implementability* cotyledons (or "spaces") representing the mathematical structures of the PR, the CR, and the TR, respectively. Also described in the MBSE book is the space of implementable system tests items, denoted ISTISR, associated with the STR as generated by the IOR and the TYR [26],[29]. A visual representation of ISTISR and its relation to the three system design spaces that comprise the tricodyledon is illustrated in Figure 2-12.

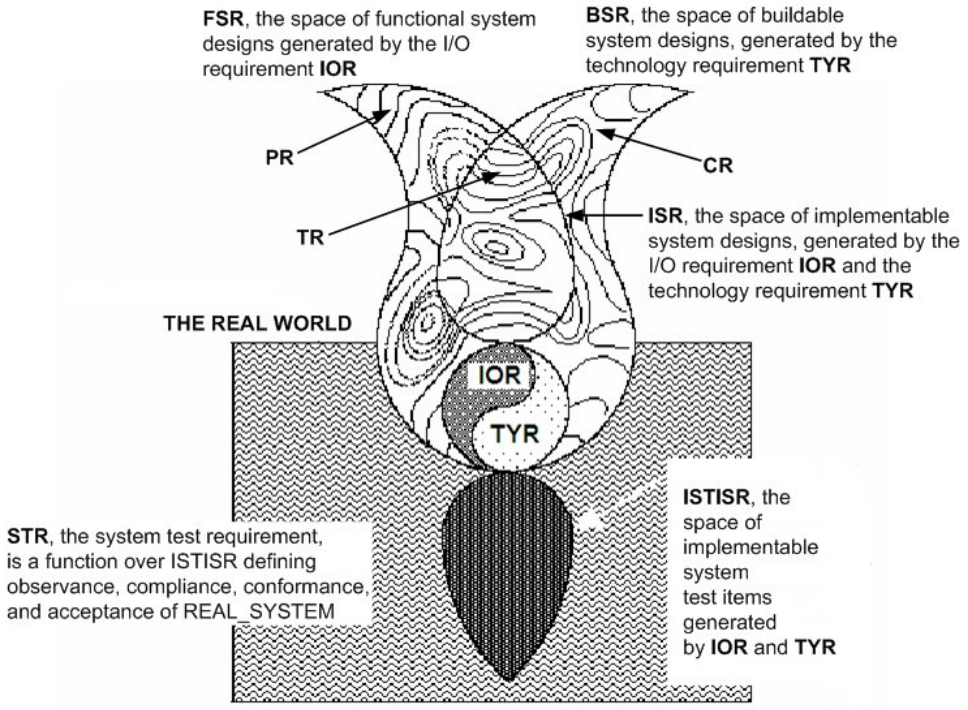

FSR, the space of functional system designs generated by the I/O requirement **IOR**
BSR, the space of buildable system designs, generated by the technology requirement **TYR**
PR
CR
TR
ISR, the space of implementable system designs, generated by the I/O requirement **IOR** and the technology requirement **TYR**
THE REAL WORLD
IOR
TYR
STR, the system test requirement, is a function over ISTISR defining observance, compliance, conformance, and acceptance of REAL_SYSTEM
ISTISR, the space of implementable system test items generated by **IOR** and **TYR**

Figure 2-12. The system test requirement anchors the system design problem to the real world.[5]

In summary, although Wymore's roots as a pure mathematician with a background in topological algebra and functional analysis are clearly visible in his work on formulating the mathematical theory behind systems science and systems engineering (including MBSE), the value of his seminal work and its applicability to the discipline of systems engineering

[5] **FSR**, **BSR**, and **ISR** represent the *functionality*, *buildability*, and *implementability* cotyledons, respectively. The contours within **FSR** represent equivalent functional system designs, generated by the performance requirement **PR**. Similarly, contours within the **BSR** represent equivalent buildable system designs, generated by the cost requirement **CR**. And contours within the **ISR** represent equivalent implementable system designs, generated by the trade-off requirement **TR**.

should not be overlooked. We will continue to monitor the progress of Dr. Wymore's research and future publications centered on Phase 2 of MBSE (MBSE2) and any other aspects related to MBSE.

3. Leading MBSE Methodologies

The following is a cursory review of some of the more notable MBSE methodologies that have received attention in the various industry forums and publications and are intended to serve as candidates for adoption and tailoring to an organization's SE practices and procedures. A brief synopsis of each methodology is described. Also included in this survey of MBSE methodologies is a JPL-developed methodology known as State Analysis.

Reader Warning: Although references to candidate MBSE methodologies will be made, some providers refer to or name their methodology a "process"—an unfortunate consequence that often leads to confusion. For purposes of this survey, methodology is implied, even if the formal offering uses the term "process" to describe or name the methodology.

3.1 IBM Telelogic Harmony-SE

3.1.1. Overview

Harmony-SE is a subset of a larger integrated systems and software development process known as *Harmony*® [32]. Development of Harmony-SE and Harmony® originated at I-Logix, Inc., formerly a leading provider of modeling tools for the embedded market.[6] Figure 3-1 graphically depicts the Harmony integrated systems and software development process.

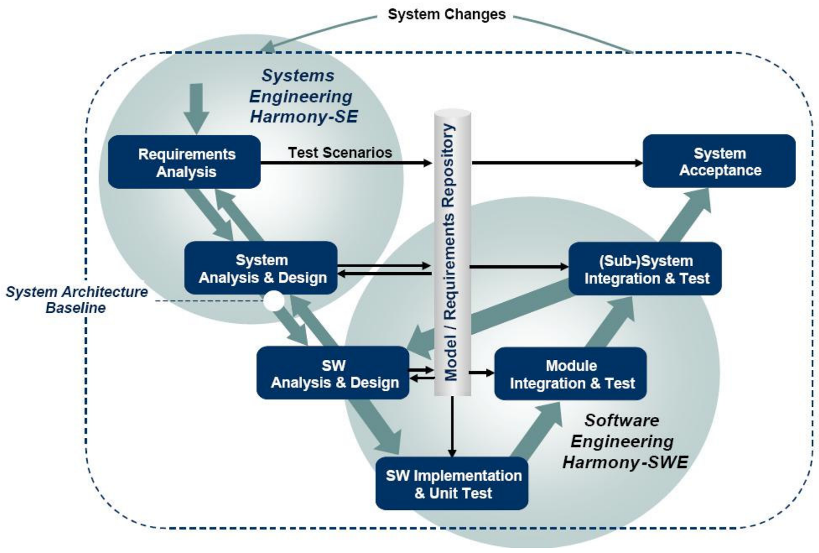

System Changes
Systems Engineering Harmony-SE
Requirements Analysis
Test Scenarios
Model / Requirements Repository
System Acceptance
System Analysis & Design
(Sub-)System Integration & Test
System Architecture Baseline
SW Analysis & Design
Module Integration & Test
Software Engineering Harmony-SWE
SW Implementation & Unit Test

Figure 3-1. Harmony® Integrated Systems and Software Development Process.

[6] I-Logix was acquired by Telelogic AB in March 2006. More recently, Telelogic AB was acquired by IBM Corporation in April 2008 and, while it retains the Telelogic brand, it is officially referred to as "Telelogic – An IBM Company." The IBM Telelogic product portfolio has grown in recent years not only due to the I-Logix acquisition by the former Telelogic AB but also its acquisition of Popkin Software, which included the System Architect tool that is widely used within the DoD acquisition community. IBM Telelogic is perhaps best known for its DOORS® product suite for requirements management and tracking.

The Harmony process was designed to be tool- and vendor-neutral, although elements of the process are supported by the Telelogic *Rhapsody* model-driven development environment (formerly, I-Logix Rhapsody) and by the Telelogic Tau offering. Note that the Harmony process somewhat mirrors the classical "Vee" lifecycle development model of system design (cf., Section 2.2). The process assumes model and requirements artifacts are maintained in a centralized model/requirements repository.

The systems engineering component of Harmony shown in the upper left corner of Figure 3-1 (i.e., Harmony-SE) has the following stated key objectives:

- Identify / derive required system functionality.
- Identify associated system states and modes.
- Allocate system functionality / modes to a physical architecture.

Harmony-SE uses a "service request-driven" modeling approach along with Object Management Group™ Systems Modeling Language™ (OMG SysML™) artifacts [33]. In the service request-driven modeling approach, system structure is described by means of SysML structure diagrams using *blocks* as basic structure elements. Communication between blocks is based on messages (*services requests*). Provided services are at the receiving part of service requests and state/mode change or operations (activities) are described as *operational contracts*. Functional decomposition is handled through decomposition of *activity operational contracts*. A SysML visual representation of the service request-driven approach is shown in Figure 3-2.

Task flow and work products (artifacts) in the Harmony-SE process include the following three top-level process elements:

- Requirements analysis
- System functional analysis
- Architectural design

Figure 3-3 better illustrates these process elements along with the flow of some of the primary work products:

Note that in addition to the use of a model/requirements repository as shown in the Harmony process (Figure 3-1), a test data repository is also recommended in order to capture use case scenarios.

Detailed task flows and work products are provided for each of the three process elements (shown as the dark filled boxes in the center of Figure 3-3) with detailed guidance provided in the Harmony-SE/SysML Deskbook [34].

An example of such a task flow and associated work products for the System Functional Analysis process element is illustrated in Figure 3-4. Similarly, an example of the task flow and associated work products for the Subsystem Architectural Design sub-process of the Architectural Design process is depicted in Figure 3-5.

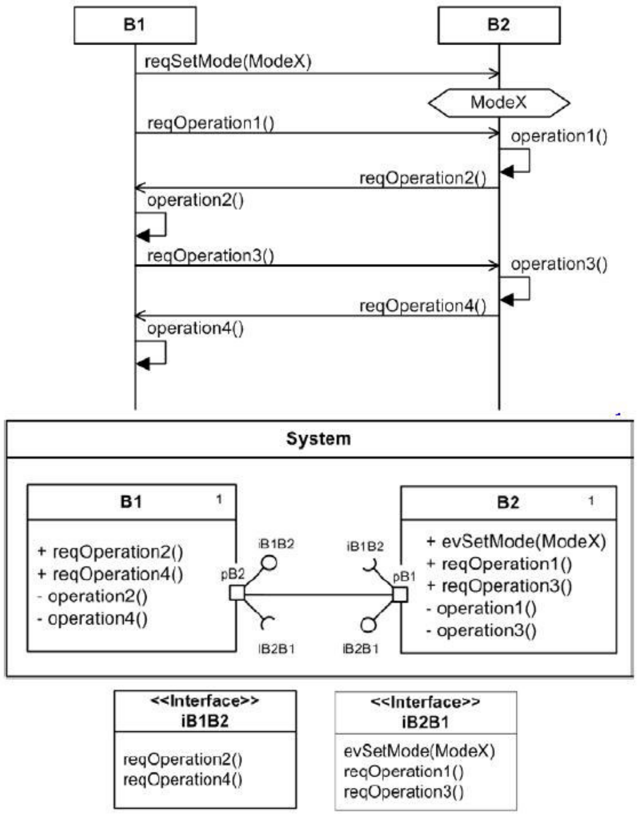


Figure 3-2. OMG SysML™ Representation of Service Request-Driven Approach.

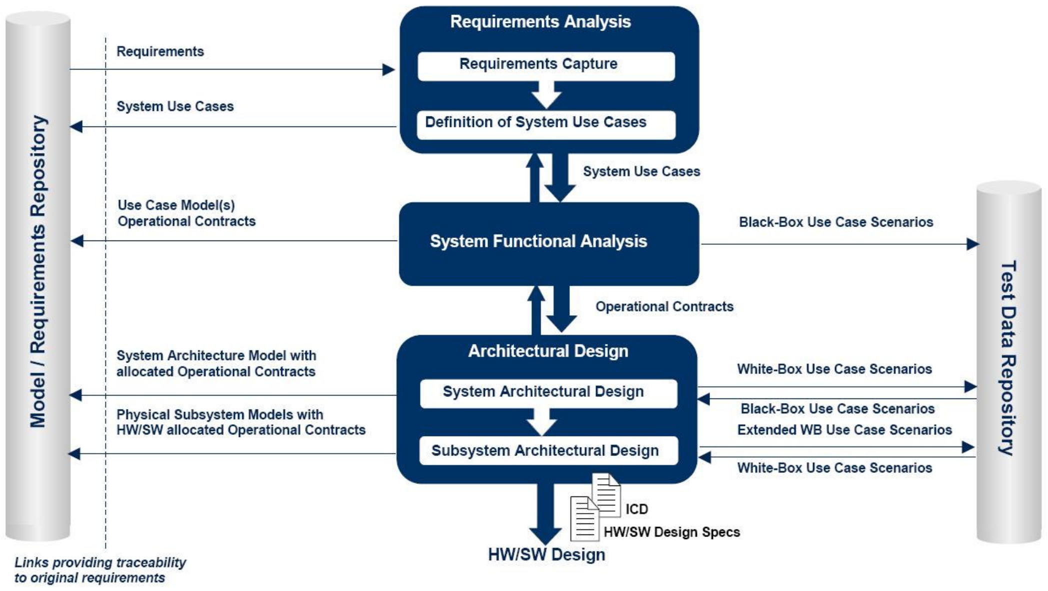

Model / Requirements Repository
Requirements
System Use Cases
Requirements Analysis
Requirements Capture
Definition of System Use Cases
System Use Cases
Use Case Model(s)
Operational Contracts
System Functional Analysis
Black-Box Use Case Scenarios
Operational Contracts
Architectural Design
System Architecture Model with allocated Operational Contracts
System Architectural Design
Physical Subsystem Models with HW/SW allocated Operational Contracts
Subsystem Architectural Design
White-Box Use Case Scenarios
Black-Box Use Case Scenarios
Extended WB Use Case Scenarios
White-Box Use Case Scenarios
Test Data Repository
ICD
HW/SW Design Specs
HW/SW Design
Links providing traceability to original requirements

Figure 3-3. Harmony-SE Process Elements.

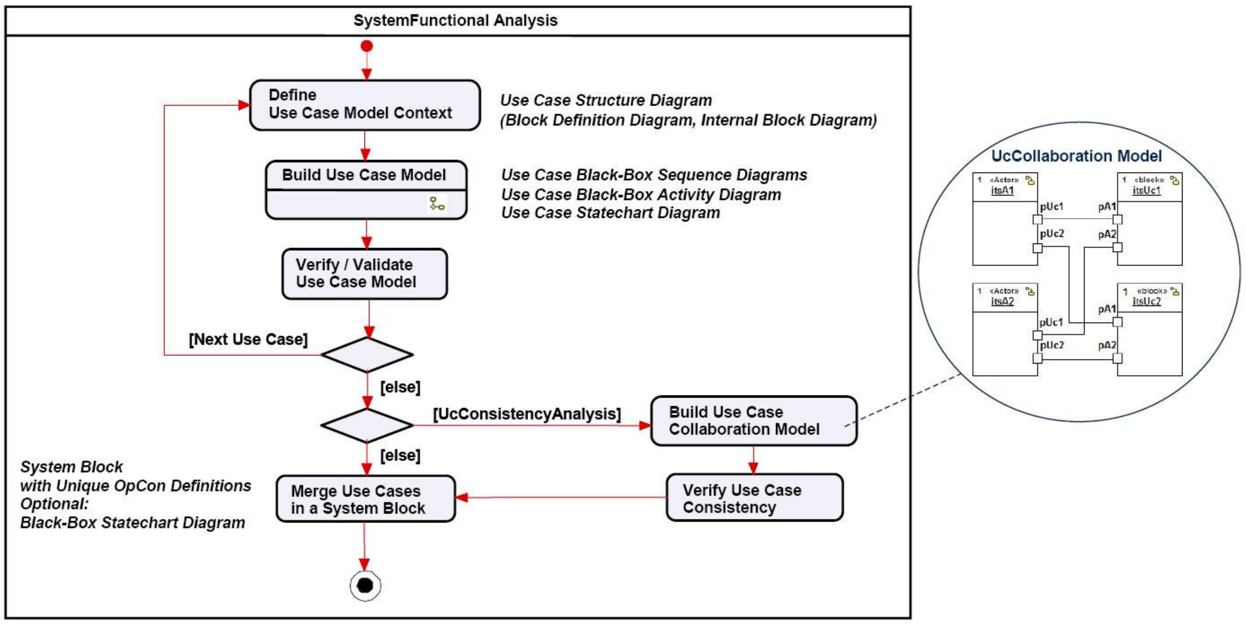

SystemFunctional Analysis
Define Use Case Model Context
Use Case Structure Diagram (Block Definition Diagram, Internal Block Diagram)
Build Use Case Model
Use Case Black-Box Sequence Diagrams
Use Case Black-Box Activity Diagram
Use Case Statechart Diagram
Verify / Validate Use Case Model
[Next Use Case]
[else]
[UcConsistencyAnalysis]
Build Use Case Collaboration Model
[else]
System Block with Unique OpCon Definitions
Optional: Black-Box Statechart Diagram
Merge Use Cases in a System Block
Verify Use Case Consistency
UcCollaboration Model
itsA1
itsUc1
itsA2
itsUc2
pUc1
pUc2
pA1
pA2

Figure 3-4. System Functional Analysis Task Flow and Work Products.

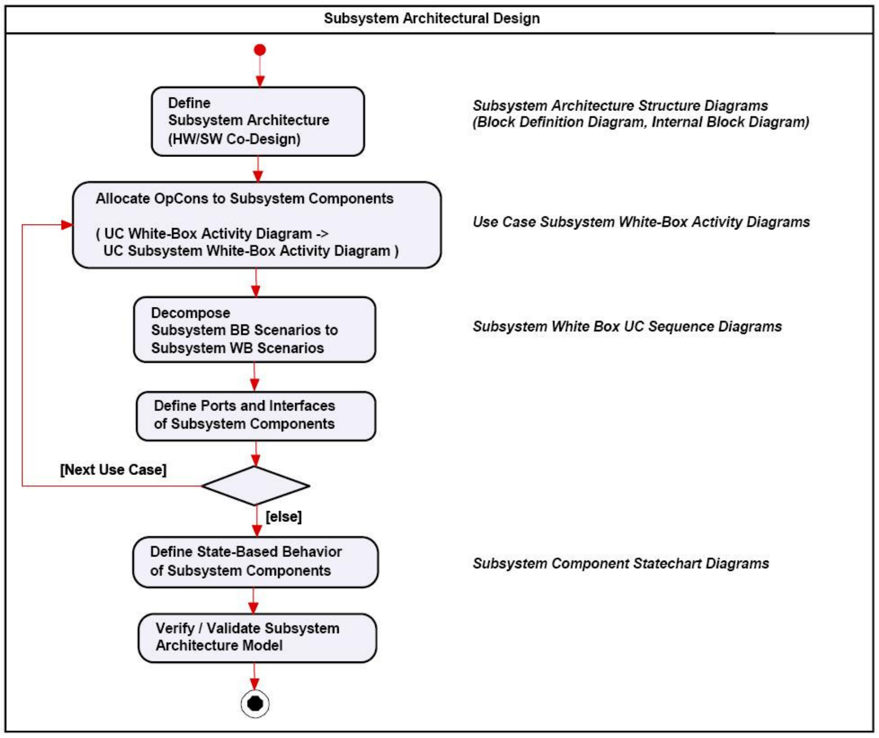

Subsystem Architectural Design
Define Subsystem Architecture (HW/SW Co-Design)
Subsystem Architecture Structure Diagrams (Block Definition Diagram, Internal Block Diagram)
Allocate OpCons to Subsystem Components (UC White-Box Activity Diagram -> UC Subsystem White-Box Activity Diagram)
Use Case Subsystem White-Box Activity Diagrams
Decompose Subsystem BB Scenarios to Subsystem WB Scenarios
Subsystem White Box UC Sequence Diagrams
Define Ports and Interfaces of Subsystem Components
[Next Use Case]
[else]
Define State-Based Behavior of Subsystem Components
Subsystem Component Statechart Diagrams
Verify / Validate Subsystem Architecture Model

Figure 3-5. Subsystem Architectural Design Task Flow and Work Products.

3.1.2. Tool Support

No process framework tool exists from IBM Telelogic or a third-party provider for Harmony-SE or the integrated systems and software engineering process, Harmony.

Recall that the Harmony-SE and Harmony were created as tool- and vendor-neutral, model-based methodologies. Tool support for MBSE that supports the methods specified by Harmony-SE and Harmony is, of course, provided by IBM Telelogic via the Telelogic Tau and Telelogic Rhapsody product offerings.

3.1.3. Offering/Availability

As stated earlier, a Harmony-SE/SysML Deskbook has been published to help guide the systems engineer and project manager through the entire MBSE methodology [22]. In addition, IBM Telelogic offers professional services to support methodology adoption.

3.2 INCOSE Object-Oriented Systems Engineering Method (OOSEM)

3.2.1. Overview

The Object-Oriented Systems Engineering Method (OOSEM) integrates a top-down, model-based approach that uses OMG SysML™ to support the specification, analysis, design, and verification of systems. OOSEM leverages object-oriented concepts in concert with more traditional top down systems engineering methods and other modeling techniques, to help architect more flexible and extensible systems that can accommodate evolving technology and changing requirements. OOSEM is also intended to ease integration with object-oriented software development, hardware development, and test.

OOSEM evolved from work in the mid 1990's at the Software Productivity Consortium (now the Systems and Software Consortium) in collaboration with Lockheed Martin Corporation.[7] The methodology was applied in part to a large distributed information system development at Lockheed Martin that included hardware, software, database, and manual procedure components. INCOSE Chesapeake Chapter established the OOSEM Working Group in November 2000 to help further evolve the methodology.[8] OOSEM is summarized in various industry and INCOSE papers [35],[36],[37], and more recently, in the forthcoming book entitled Practical Guide to SysML: Systems Modeling Language by Fridenthal, Moore, and Steiner [38]. An introduction to the methodology is also available as a full day tutorial [39].

The OOSEM objectives are the following:

- Capture and analysis of requirements and design information to specify complex systems.
- Integration with object-oriented (OO) software, hardware, and other engineering methods.
- Support for system-level reuse and design evolution.

As stated above, OOSEM is a hybrid approach that leverages object-oriented techniques and a systems engineering foundation. It also introduces some unique techniques as indicated in see Figure 3-6.

[7] Sanford Friedenthal, Lockheed Martin Corporation (private communication), Apr. 4, 2007.
[8] David Griffith, Northrop Grumman Corporation (private communication), Mar. 15, 2007.

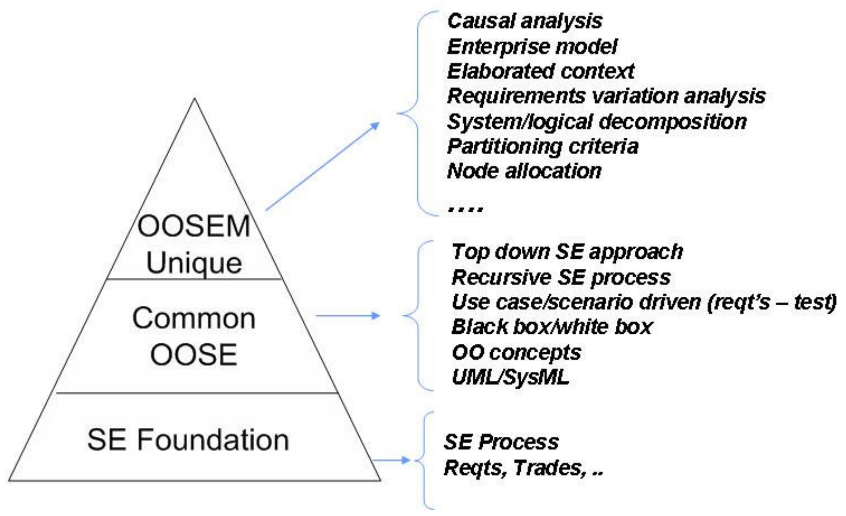


Figure 3-6. Foundation of OOSEM.

The OOSEM supports a SE process as illustrated in Figure 3-7.

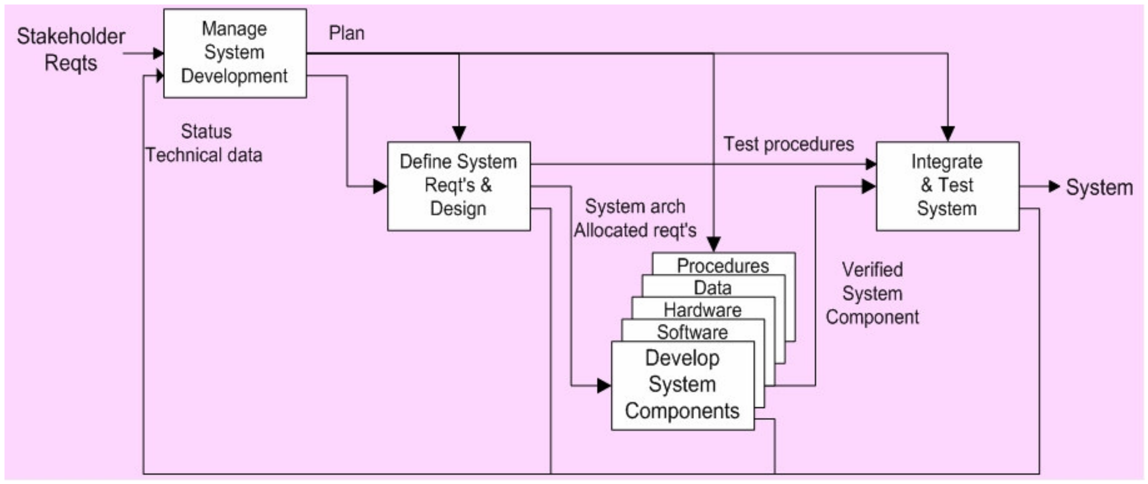


Figure 3-7. OOSEM Activities in the Context of the System Development Process.

The core tenets of OOSEM include recognized practices essential to systems engineering that include: 1) Integrated Product Development (IPD), essential to improve communications, and 2) a recursive "Vee" lifecycle process model that is applied to each multiple level of the system hierarchy.

As shown in Figure 3-8, OOSEM includes the following development activities:

- Analyze Stakeholder Needs
- Define System Requirements
- Define Logical Architecture
- Synthesize Candidate Allocated Architectures
- Optimize and Evaluate Alternatives
- Validate and Verify System

These activities are consistent with typical systems engineering "Vee" process that can be recursively and iteratively applied at each level of the system hierarchy. Fundamental tenets of systems engineering, such as disciplined management processes (i.e. risk management, configuration management, planning, measurement, etc.) and the use of multi-disciplinary teams, must be applied to support each of these activities to be effective.

OOSEM utilizes a model-based approach to represent the various artifacts generated by the development activities using OMG SysML as the predominant modeling language. As such, it enables the systems engineer to precisely capture, analyze, and specify the system and its components and ensure consistency among various system views. The modeling artifacts can also be refined and reused in other applications to support product line and evolutionary development approaches. A summary description of the activities and artifacts is provided on the following pages [37].

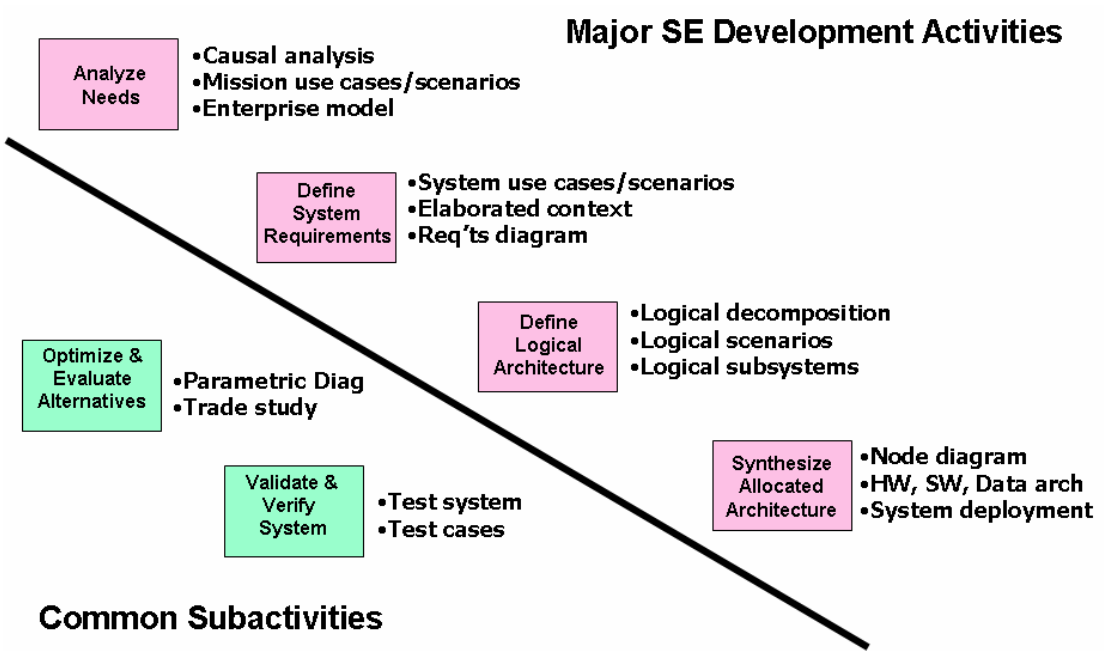


Figure 3-8. OOSEM Activities and Modeling Artifacts.

Analyze Stakeholder Needs

This activity captures the "as-is" systems and enterprise, their limitations and potential improvement areas. The results of the "as-is" analysis is used to develop the to-be enterprise and associated mission requirements. An enterprise model depicts the enterprise, its constituent systems, including the systems to be developed or modified, and enterprise actors (entities external to the enterprise). The as-is enterprise is analyzed using causal analysis techniques to determine its limitations, and used as a basis for deriving the mission requirements and to-be enterprise model. The mission requirements are specified in terms of the mission / enterprise objectives, measures of effectiveness, and top-level use cases. The use cases and scenarios capture the enterprise functionality.

Define System Requirements

This activity is intended to specify the system requirements that support the mission requirements. The system is modeled as a black box that interacts with the external systems and users represented in the enterprise model. The system-level use cases and scenarios reflect the operational concept for how the system is used to support the enterprise. The scenarios are modeled using activity diagrams with swim lanes that represent the black box system, users, and external systems. The scenarios for each use case are used to derive the black box system functional, interface, data, and performance requirements. The requirements management database is updated during this activity to trace each system requirement to the enterprise/mission level use case and mission requirements.

Requirements variation is evaluated in terms of the probability that a requirement will change, which is included in the risks, and later analyzed to determine how to design the system to accommodate the potential change. A typical example may be a system interface that is likely to change or a performance requirement that is expected to increase.

Define Logical Architecture

This activity includes decomposing and partitioning the system into logical components that interact to satisfy the system requirements. The logical components capture the system functionality. Examples may include a user interface that is realized by a web browser, or an environmental monitor that is realized by a particular sensor. The logical architecture/design mitigates the impact of requirements changes on the system design, and helps to manage technology changes.

OOSEM provides guidelines for decomposing the system into its logical components. The logical scenarios preserve system black box interactions with its environment. In addition, the logical component functionality and data are repartitioned based on partitioning criteria such as cohesion, coupling, design for change, reliability, performance, and other considerations.

Synthesize Candidate Allocated Architectures

The allocated architecture describes relationship among the physical components of the system including hardware, software, data and procedures. The system nodes define the distribution of resources. Each logical component is first mapped to a system node to address how the functionality is distributed. Partitioning criteria is applied to address distribution concerns such as performance, reliability, and security. The logical components are then allocated to hardware, software, data, and manual procedure components. The software, hardware, and data architecture are derived based on the component relationships. The requirements for each component are traced to the system requirements and maintained in the requirements management database.

Optimize and Evaluate Alternatives

This activity is invoked throughout all other OOSEM activities to optimize the candidate architectures and conduct trade studies to select the preferred architecture. Parametric models for modeling performance, reliability, availability, life-cycle cost, and other specialty engineering concerns, are used to analyze and optimize the candidate architectures to the level needed to compare the alternatives. The criteria and weighting factors used to perform the trade studies are traceable to the system requirements and measures of effectiveness. This activity also includes the monitoring of technical performance measures and identifies potential risks.

Validate and Verify System

This activity is intended to verify that the system design satisfies its requirements and to validate that the requirements meet the stakeholder needs. It includes the development of verification plans, procedures, and methods (e.g., inspection, demonstration, analysis, test). System-level use cases, scenarios, and associated requirements are primary inputs to the development of the test cases and associated verification procedures. The verification system can be modeled using the same activities and artifacts described above for modeling the operational system. The requirements management database is updated during this activity to trace the system requirements and design information to the system verification methods, test cases, and results.

The full description of each OOSEM activity and process flows are provided in the cited book by Friedenthal, Moore, and Steiner [38] and the referenced OOSEM tutorial [39].

3.2.2. Tool Support

A dedicated process framework tool for OOSEM does not exist; however, tool support for OOSEM can be provided by COTS-based OMG SysML tools and associated requirements management tools. Other tools required to support the full system lifecycle should be integrated with the SysML and requirements management tools, such as configuration management, performance modeling, and verification tools.

A more complete set of OOSEM tool requirements is provided in the referenced OOSEM tutorial [39].

3.2.3. Offering/Availability

The OOSEM tutorial and training materials can be made available by contacting the INCOSE OOSEM Working Group to gain access through the INCOSE Connect collaboration space. Unlike other industry-provided MBSE methodologies, OOSEM is not a formal offering that can be purchased from any specific vendor, including professional services. Support services may be available by contacting representatives of the INCOSE OOSEM Working Group.[9]

3.3 IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDSD)

3.3.1. Overview

The Rational Unified Process for Systems Engineering (RUP SE) is a derivative of the Rational Unified Process® (RUP®). RUP is a methodology that is both a process framework and process product from IBM Rational and has been used extensively in government and industry to manage software development projects [40].

RUP SE was created to specifically address the needs of systems engineering projects [41],[42]. The objective for its creation was to apply the discipline and best practices of the RUP for software development to the challenges of system specification, analysis, design, and development. Its goal is to help organizations save time, cut costs, reduce risk, and

[9] L. Mark Walker, Lockheed Martin Corporation (private communication), Apr. 19, 2007.

improve the quality of the systems they build. According to Cantor,[10] in current parlance, "RUP SE is the extension of the Rational Unified Process [RUP] to support Model-Driven Systems Development [MDSD]." The spirit of MDSD as envisioned by key IBM systems engineering leaders is documented in the cited reference by Balmelli et al. and will not be replicated here [16].

Before describing the guiding principles, methods, and architectural framework of RUP SE to support MDSD, it is helpful to familiarize the reader with the software development lifecycle focused RUP. RUP is based on a set of building blocks, or *content elements*, describing what is to be produced, the necessary skills required, and the step-by-step explanation describing how specific development goals are achieved. A graphical depiction of the RUP process framework is shown in Figure 3-9 [40] sometimes referred to in the industry as the "whale chart."

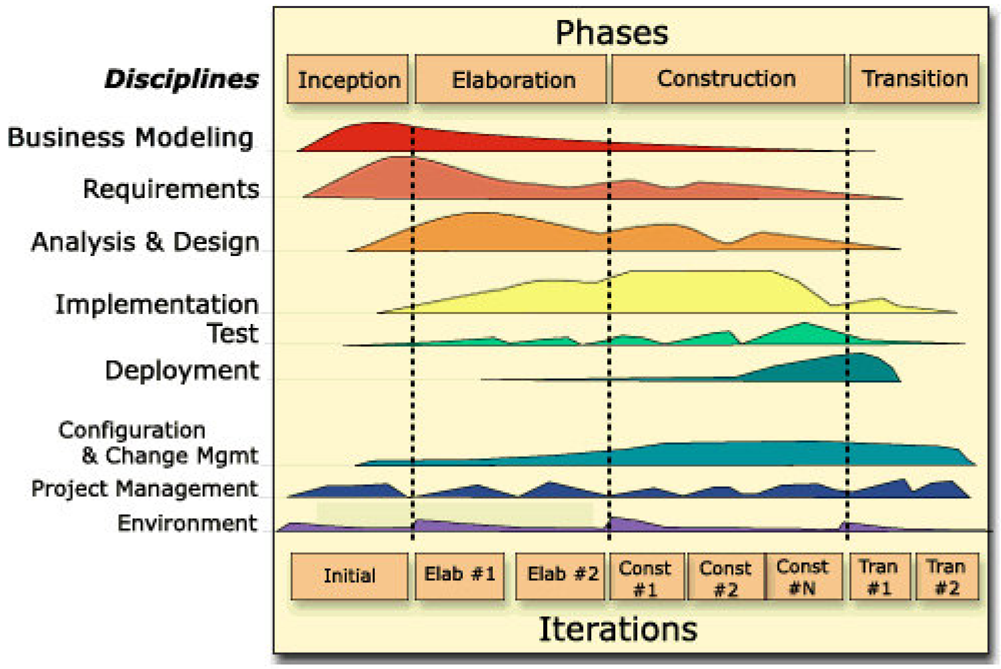


Figure 3-9. The Rational Unified Process® (RUP®) ("Whale Chart").

The main content elements of the RUP are the following:

- Roles ("WHO") – A *role* defines a set of related skills, competencies, and responsibilities.
- Work Products ("WHAT") – A *work product* represents something resulting from a task, including all the documents and models produced while working through the process.
- Tasks ("HOW") – A *task* describes a unit of work assigned to a role that provides a meaningful result.

Within each iteration, the tasks are categorized into a total of nine (9) *disciplines*:

[10] Murray Cantor, IBM Corporation (private communication), Feb. 27, 2007.

Engineering Disciplines:

1. Business modeling
2. Requirements
3. Analysis and design
4. Implementation
5. Test
6. Deployment

Supporting Disciplines:

7. Configuration and change management
8. Project management
9. Environment

The RUP lifecycle is an implementation of the spiral model for iterative and incremental development (cf., Section 2.2). It was created by assembling the content elements into semi-ordered sequences. Consequently the RUP lifecycle is available as a work breakdown structure (WBS), which can be customized to address the specific needs of a project. The RUP lifecycle organizes the tasks into phases and iterations.

A project has four phases:

- Inception
- Elaboration
- Construction
- Transition

A typical project profile showing the relative sizes of the four phases is shown in Figure 3-10 [40].

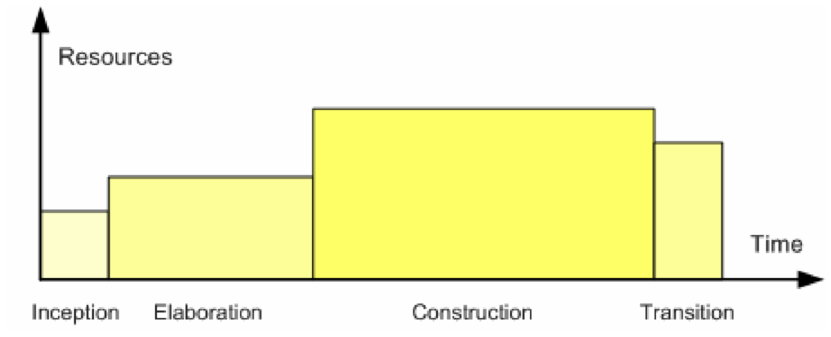


Figure 3-10. Typical Profile Showing Relative Sizes of the Four RUP Phases.

Because RUP SE is derived from RUP, it retains RUP's cornerstone principles, which have been refined and extended to enhance their utility for systems engineering efforts. RUP SE brings the RUP style of concurrent design and iterative development to systems engineering (as illustrated in Figure 3-11) [43]. In addition, it provides the highly configurable discipline (workflow) templates required to identify the hardware, software, and worker role components that comprise a systems engineering project.

RUP and RUP SE both are designed to help teams systematically define, organize, communicate, and manage requirements. Both methodologies support change control and quality initiatives. Without these capabilities, no systems engineering project is likely to be deemed a success relative to cost or business objectives.

Key elements in RUP SE that extend the RUP to systems engineering include the following:

- **New roles**. In RUP SE, the development team includes system engineers in addition to worker roles such as architects, developers, testers, etc. The role of the system engineer is primarily concerned with the specification of the overall system and deployment thereof, and to help address overall system requirements.

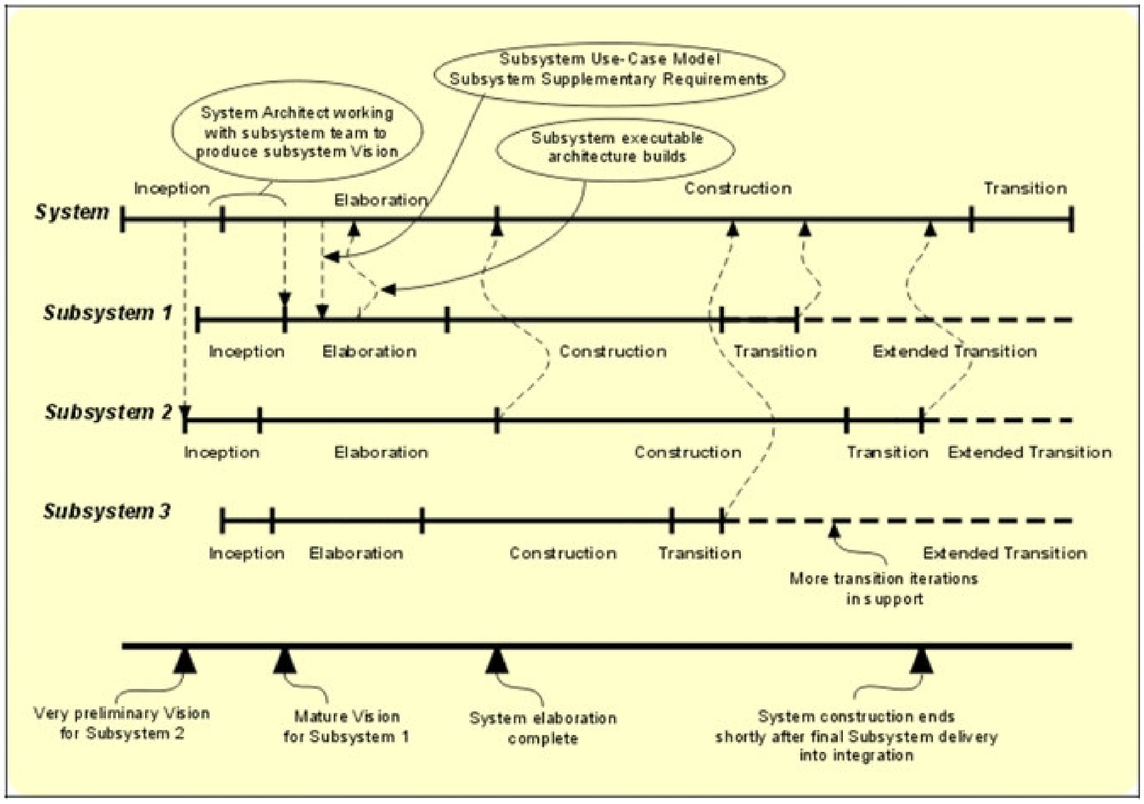

Subsystem Use-Case Model
Subsystem Supplementary Requirements
System Architect working with subsystem team to produce subsystem Vision
Subsystem executable architecture builds
System
Inception
Elaboration
Construction
Transition
Subsystem 1
Inception
Elaboration
Construction
Transition
Extended Transition
Subsystem 2
Inception
Elaboration
Construction
Transition
Extended Transition
Subsystem 3
Inception
Elaboration
Construction
Transition
Extended Transition
More transition iterations in support
Very preliminary Vision for Subsystem 2
Mature Vision for Subsystem 1
System elaboration complete
System construction ends shortly after final Subsystem delivery into integration

Figure 3-11. Illustration of RUP SE lifecycle.

- **New artifacts and workflows**. RUP includes full support for software system concerns, such as usability, maintainability, performance, and scalability. RUP SE adds artifacts and workflows that address additional concerns in the systems engineering domain, such as security, training, and logistics support.
- **An emphasis on business modeling**. Whatever kind of system being architected, it is important to understand the business purpose it will serve. Otherwise, system requirements will not accurately reflect business activities. RUP SE does not include changes to the business modeling features of RUP. However, RUP SE users are strongly encouraged to create business use cases with the associated identification of

business actors and the flow of business events, in order to adequately define system requirements. Furthermore, the RUP SE use-case flowdown activity is applied to derive system requirements from business requirements.

- **Viewpoints for systems engineering**. An architecture framework for RUP SE has been developed that contains the elements of model levels, viewpoints, and views (see Table 3-1). The concept of viewpoints and views used in the RUP SE architecture framework is consistent with industry standard definitions as articulated by the ISO/ITU 10746 standard *Reference Model for Open Distributed Processing (RM-ODP)* [44] and the ANSI/IEEE 1471-2000 standard *Recommended Practice for Architectural Description of Software-Intensive Systems* [45]. The cells in RUP SE architecture framework represent views.

  RUP SE supports domain-specific viewpoints common to system architectures, such as safety, security, and mechanical. Modeling levels are similar for most systems regardless of their complexity.

Table 3-1. The RUP SE architecture framework.

Model Levels	**Model Viewpoints**					
	Worker	**Logical**	**Information**	**Distribution**	**Process**	**Geometric**
Context	Role definition, activity modeling	Use case diagram specification	Enterprise data view	Domain-dependent views		Domain-dependent views
Analysis	Partitioning of system	Product logical decomposition	Product data conceptual schema	Product locality view	Product process view	Layouts
Design	Operator instructions	Software component design	Product data schema	ECM (electronic control media design)	Timing diagrams	MCAD (mechanical computer-assisted design)
Implementation	Hardware and software configuration					

Note: The Distribution viewpoint describes how the functionality of the system is distributed across physical resources. At the analysis level, it is necessary to describe a generalized view of resources, capturing the attributes needed to support the transformation from analysis and design. Cantor introduced the concept of *locality* to represent a generalized resource [41]. A locality is defined as a member of a system partition representing a generalized or abstract view of the physical resources. Localities can perform operations and have attributes appropriate for specifying physical designs. Localities are linked to each other with connections. Connections are defined as generalized physical linkages in RUP SE. *Connections* are characterized by what they carry or transmit and the necessary performance and quality attributes in order to specify their physical realization at the design level. A RUP SE distribution diagram showing two localities and a connection between them is illustrated in Figure 3-12.

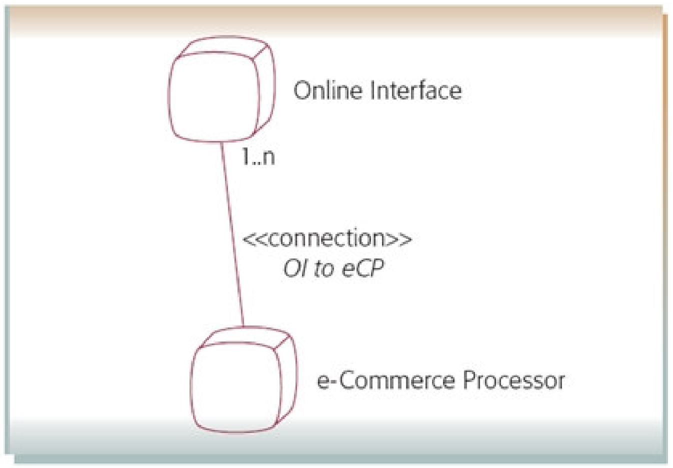


Figure 3-12. Two Localities and a Connection.

A *model level* is defined as a subset of the architecture model that represents a certain level of specificity (abstract to concrete); lower levels capture more specific technology choices. Model levels are not levels of abstraction; in fact, a model level may contain multiple levels of abstraction. Model levels are elements designed to group artifacts with a similar level of detail (see Table 3-2).

Table 3-2. Model levels in the RUP SE architecture framework.

Model Level	**Expresses**
Context	System black box—the system and its actors (through this is a black-box view for the system, it is a white-box view for the enterprise containing the system.
Analysis	System white box—initial system partitioning in each viewpoint that establishes the conceptual approach.
Design	Realization of the analysis level in hardware, software, and people
Implementation	Realization of the design model into specific configurations

- **Scalability enhancements**. Once design decisions have been captured in viewpoints and specified via model levels, the system architecture is captured in a set of OMG™ UML®/SysML™ diagrams; these further describe it from the various viewpoints and model levels. Although many of these artifacts are similar across RUP and RUP SE, there are a couple of important differences. In a nutshell, these new artifacts allow you to break the system down (1) by subsystems, and (2) by the localities where processing takes place. Each subsystem coupled with its locality has its own derived requirements in RUP SE, enabling the process to scale to meet the needs of even the largest and most complex projects.
- **Allocated versus derived requirements**. RUP SE encompasses two types of system requirements: use-cases, which capture functional requirements; and supplementary requirements, which cover non-functional (quality) attributes like reliability and maintainability (see Figure 3-13) [43]. With respect to the requirements associated with subsystems and localities, RUP SE makes a further distinction between those requirements that are allocated and those that are derived. A locality or subsystem requirement is allocated if a locality or subsystem is assigned sole responsibility for fulfilling a system requirement. A locality or subsystem requirement is derived if it is identified by studying how the subsystem or locality collaborates with others to meet a system requirement.

- **Subsystem-level flowdown activity**. RUP SE derives system requirements from business requirements via use-case flowdown activities. However, RUP SE departs from the RUP in that it also specifies a flow of events in a subsystem-level, "white box" view that references specific architectural elements.[11] This extra step is necessary in order to make decisions about where events are hosted, and to relate processes to events.
- **Support for designing additional components**. The design-level specification of system components with RUP SE is similar to its software-only counterpart in RUP. The key difference, as previously mentioned, is that systems engineering typically entails additional types of components than software engineering, such as hardware. Delineation of these components is supported via analysis of the RUP SE subsystem and locality use-case surveys that are generated prior to specifying component designs.

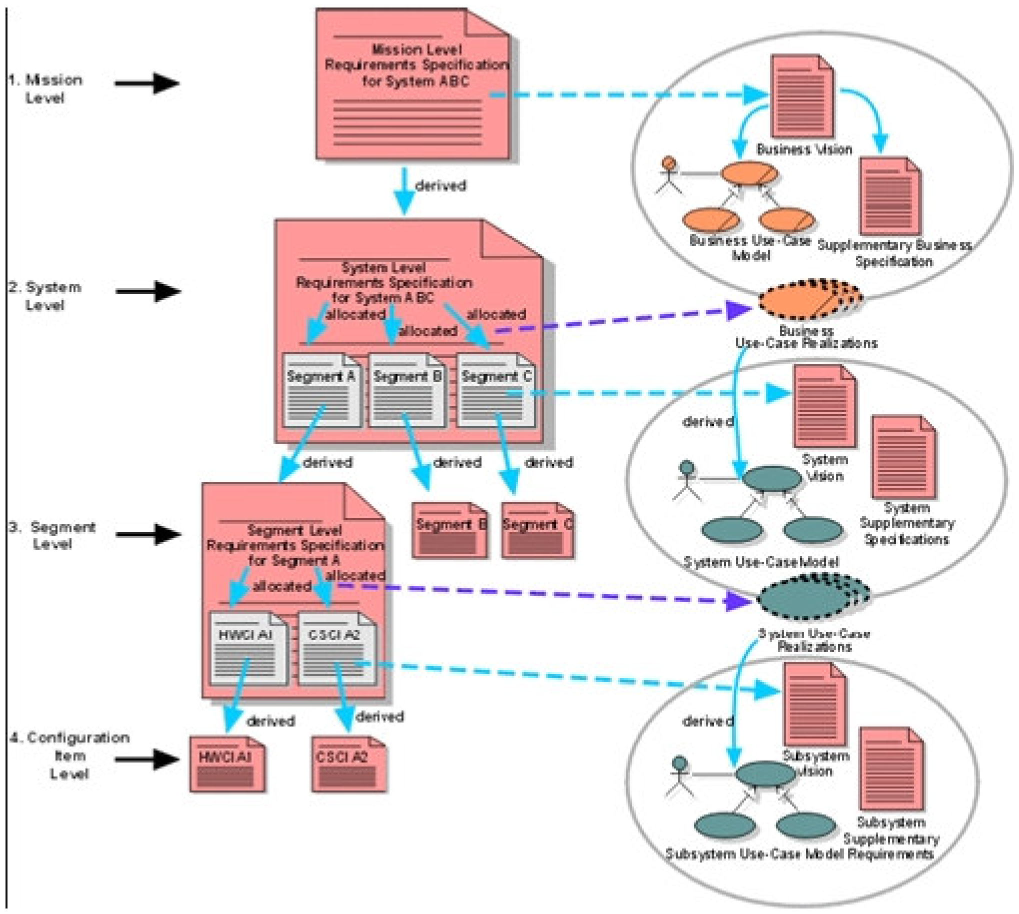

1. Mission Level
Mission Level Requirements Specification for System ABC
derived
2. System Level
System Level Requirements Specification for System ABC
allocated
allocated
allocated
Segment A
Segment B
Segment C
derived
derived
derived
3. Segment Level
Segment Level Requirements Specification for Segment A
allocated
allocated
HWCI A1
CSCI A2
derived
derived
4. Configuration Item Level
HWCI A1
CSCI A2
Business Vision
Business Use-Case Model
Supplementary Business Specification
Business Use-Case Realizations
derived
System Vision
System Supplementary Specifications
System Use-Case Model
System Use-Case Realizations
derived
Subsystem Vision
Subsystem Supplementary Requirements
Subsystem Use-Case Model

Figure 3-13. RUP SE Requirements Allocation/Derivation Method.

[11] The classical notion of a "white box" (*the elements or parts that make up a system*) and "black box" (*characteristics of the system as a whole: the services it provides, the requirements it meets*) characterization of a system is consistent with the IBM Model-Driven Systems Development (MDSD) approach and is described as part of the RUP SE methodology [16].

3.3.2. Tool Support

Unlike most of the other MBSE methodologies surveyed, a process framework tool does exist to support RUP SE and is available via the RUP SE *plugin* for the Rational Method Composer (RMC) product offering from IBM Rational® software. At the time of this writing, RUP SE V3.0 is included as part of RMC V7.2. A complete list of RMC plugins can be found at:
http://www.ibm.com/developerworks/rational/downloads/07/rmc_v7.2/

Direct MBSE tool support is provided by IBM through its Rational® suite of tool offerings that support analysis, modeling, design, and construction, albeit mostly with a software development focus; IBM Rational® has not historically been known as a provider of systems engineering tools per se. The Rational Rose product family, Rational Systems Developer (RSD), and Rational Software Modeler/Architect (RSM/RSA) offerings do support OMG™ UML®. Support for OMG™ SysML® is provided via the EmbeddedPlus SysML Toolkit, which is a third party offering from EmbeddedPlus Engineering.

Most of these tools mentioned, including RMC, are supported on the Eclipse™ open source platform managed under the auspices of the Eclipse Foundation, Inc.

3.3.3. Offering/Availability

As stated in Section 3.3.2, RUP SE tool support is provided by the RUP SE plugin for Rational® Method Composer (RMC); however, it is recommended that adoption and tailoring of the RUP SE methodology be supported through IBM professional services; specifically, IBM Software Services. A textbook by Kruchten exist for the baseline RUP® methodology that details core tenets and elements of the methodology and provides tailoring guidelines [40]. Such a companion text does not yet exist for RUP SE, at least not at the time of this survey report.

3.4 Vitech Model-Based System Engineering (MBSE) Methodology

3.4.1. Overview

Vitech Corporation, providers of the CORE® product suite, offer a MBSE methodology via a set of tutorials developed and offered by Vitech CEO and Chief Methodologist James ("Jim") E. Long [46]. A variation of the tutorial has been delivered at a number of INCOSE International Symposia as a half-day event [47]. Although the Vitech MBSE methodology is considered "tool-independent," there is a strong tie of the tutorial materials to the CORE tool set.

The Vitech MBSE methodology is based on four primary concurrent SE activities that are linked and maintained through a common *System Design Repository* (see Figure 3-14).

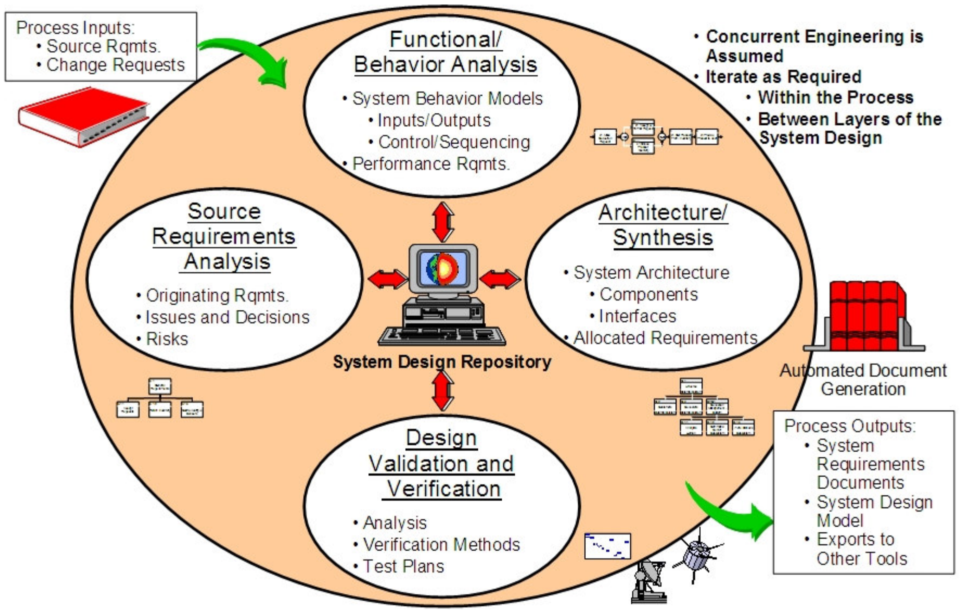


Figure 3-14. Vitech MBSE Primary SE Activities.

Each of these primary SE activities is linked within the context of associated "domains" as illustrated in Figure 3-15, where the SE activities are considered elements of a particular kind of domain known as the *Process Domain*.

In the Vitech MBSE methodology, it is stressed that a MBSE System Definition Language (SDL) is needed to manage model artifacts, which means that an agreed-upon information model in the form of a schema or ontology is necessary to manage the syntax (structure) and semantics (meaning) of the model artifacts [48],[49]. Such an "SDL" has a number of uses such as providing a structured, common, explicit, context-free language for technical communication serving as a guide for requirements analysts, system designers, and developers, and providing a structure for the graphic view generators, report generator scripts, and consistency checkers.[12] An example of a Vitech-specified MBSE SDL is illustrated in Table 3-3. Vitech MBSE System Definition Language (SDL). and based on and Entity-Relationship-Attribute (ERA) model.

[12] Many of these features of an MBSE SDL are targeted at the MBSE tool that interacts with or hosts the system design repository and is beyond the scope of other key elements of MBSE methodologies such as processes and methods. Nevertheless, the importance of specifying, owning, and utilizing an MBSE information model is acknowledged and a factor that is not explicitly called out in the literature of other MBSE methodologies surveyed in this study.

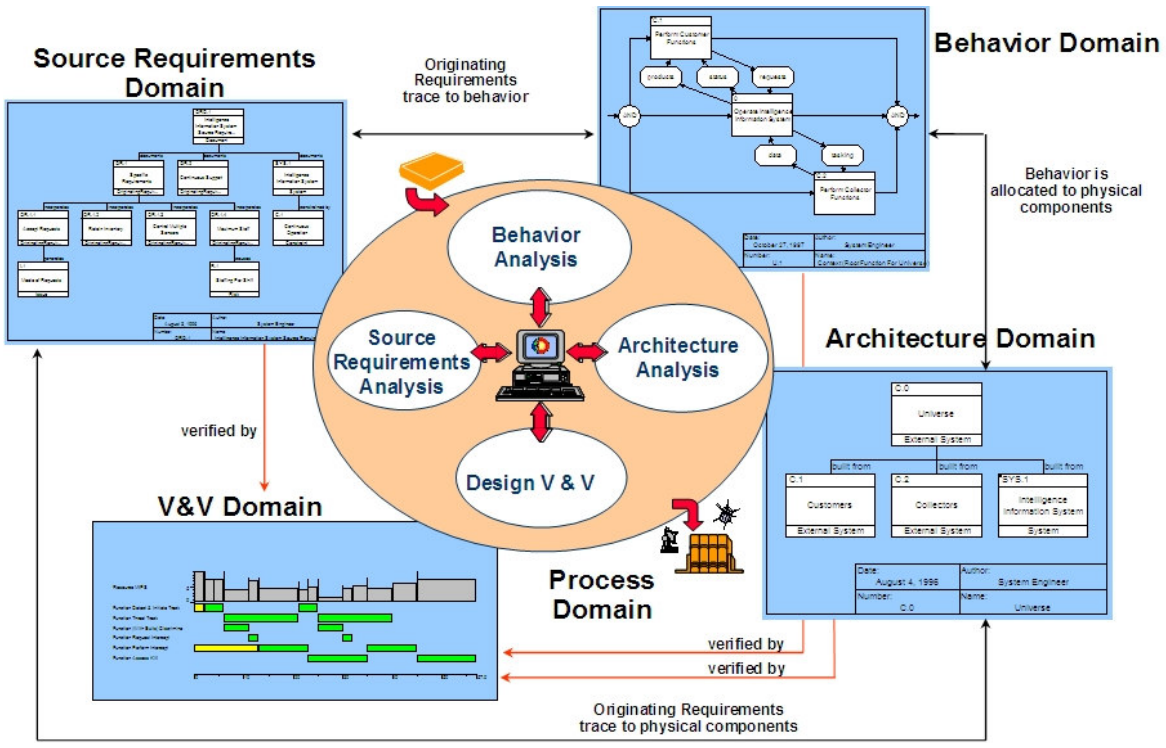

Source Requirements Domain
Originating Requirements trace to behavior
Behavior Domain
Behavior is allocated to physical components
Behavior Analysis
Source Requirements Analysis
Architecture Analysis
Design V & V
Architecture Domain
verified by
V&V Domain
Process Domain
verified by
verified by
Originating Requirements trace to physical components

Figure 3-15. Vitech MBSE Primary SE Domains.

Five core tenets help drive the Vitech MBSE methodology:

1. Model via modeling "language" the problem and the solution space; include semantically-meaningful graphics to stay explicit and consistent. This helps facilitate model traceability, consistent graphics, automatic documentation and artifacts, dynamic validation and simulation, and promotes more precise communication.
2. Utilize a MBSE system design repository.
3. Engineer the system horizontally before vertically, i.e., do it in complete, converging layers.
4. Use tools to do the "perspiration stuff" and your brain to do the "inspiration stuff."

To support tenet #3 above, the Vitech MBSE utilizes an incremental SE process known as the "Onion Model," which allows complete interim solutions at increasing levels of detail during the system specification process [50]. A visual representation of the Onion Model is illustrated in Figure 3-16.

Table 3-3. Vitech MBSE System Definition Language (SDL).

SDL Language*	English Equivalent	MBSE Example
Element	Noun	• **Requirement:** Place Orders • **Function:** Cook Burgers • **Component:** Cooks
Relationship	Verb	• **Requirement** basis of **Functions** • **Functions** are allocated to **Components**

Attribute	Adjective	• Creator • Creation Date • Description
Attribute of Relationship	Adverb	• **Resource** consumed by **Function** • Amount (of Resource) • Acquire Available (Priority)
Structure	N/A	• Viewed as Enhanced Function Flow Block Diagram (EFFBD) or FFBD
*Mapped to model element property sheets in Vitech CORE®		

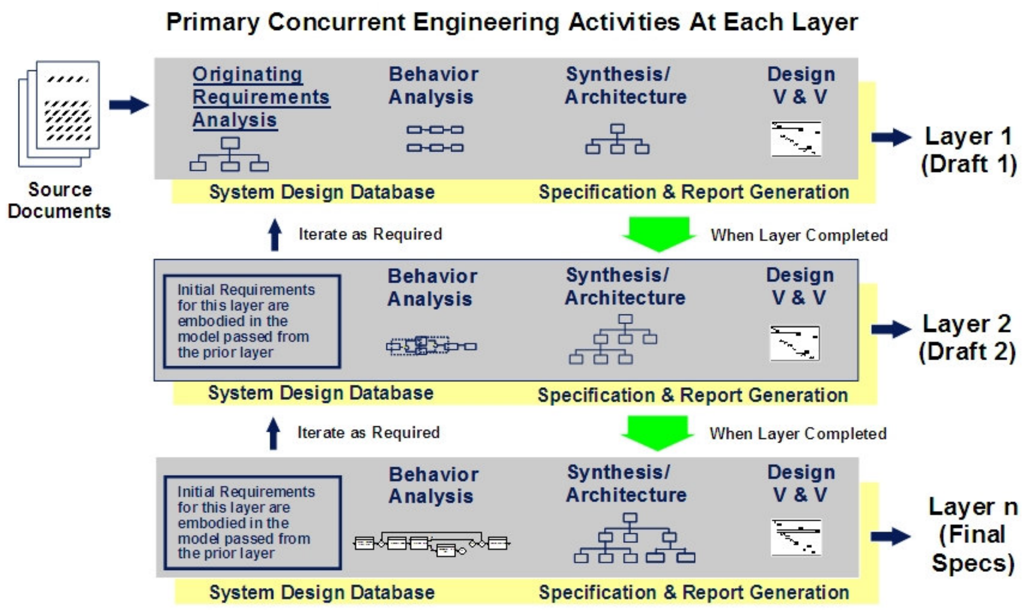

Primary Concurrent Engineering Activities At Each Layer
Source Documents
Originating Requirements Analysis
Behavior Analysis
Synthesis/ Architecture
Design V & V
System Design Database
Specification & Report Generation
Layer 1 (Draft 1)
Iterate as Required
When Layer Completed
Initial Requirements for this layer are embodied in the model passed from the prior layer
Layer 2 (Draft 2)
Layer n (Final Specs)

Figure 3-16. Vitech MBSE "Onion Model."

The Onion Model iterates the primary concurrent SE activities at each layer. According to Childers and Long [50], as the SE team successfully completes one level of system design, they "peel off a layer of the onion" and start to explore the next layer. When the team reaches the desired level of detail (the center), their design is complete. The primary benefit of the Onion Model over say more traditional waterfall SE approaches is that it provides a lower risk design approach since complete solutions at increasing levels of detail are available for early review and validation [50].

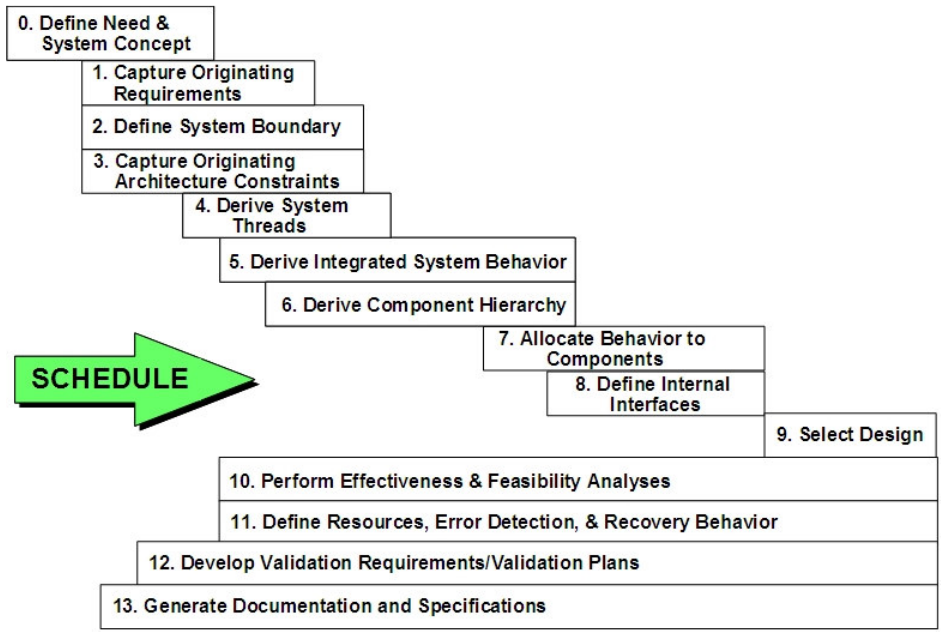
Completeness and *convergence* are essential principles of the Onion Model in that the SE team must complete a layer before moving to the next layer (completeness) and the team cannot iterate back more than one layer (convergence). If no valid, consistent solution can be found at any layer, the team must check if the system statement is overly constrained and may need to negotiate modifications such as modifications to the design implementation at the previous layer [50]. It is important to discover such constraints early as system design breakage that occurs in several layers lower in the iterative process can adversely impact cost and schedule. Guidance for determining completeness at each layer is provided in Table 3-4.

Table 3-4. Completion Criteria for Each Layer of the "Onion Model."

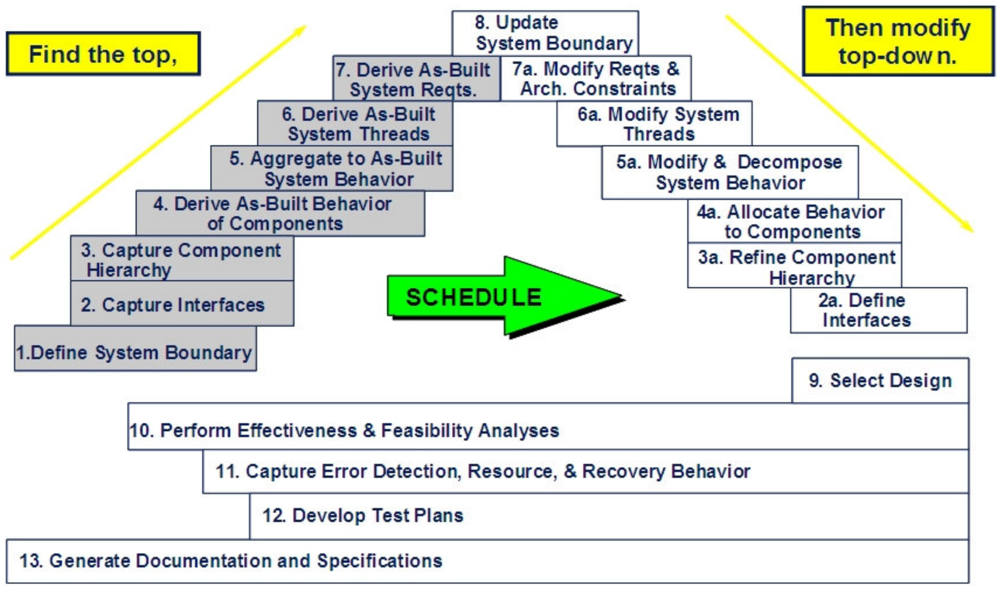
Process Element	Completion Criteria
1. Originating Requirements	1. Agreement on Acceptance Criteria.
2. Behavior/Functional Architecture	2. Each function is uniquely allocated to at most one component.
3. Physical Architecture Definition	3. Segment/component specs are complete requirements documents.
4. Qualification	4. V&V requirements have been traced to test system components.

The Onion Model is supported by two sets of SE activities timelines that are intend to apply to each layer of the "Onion;" one for a top down process (Figure 3-17a) and one for reverse engineering (Figure 3-17b).

Note that *schedule* is read as increasing in time from left to right in these SE activity timelines and the activity bars represent movement of the "center of gravity" of the SE team. Further, it is important to re-iterate that concurrent engineering is assumed.


0. Define Need & System Concept
1. Capture Originating Requirements
2. Define System Boundary
3. Capture Originating Architecture Constraints
4. Derive System Threads
5. Derive Integrated System Behavior
6. Derive Component Hierarchy
7. Allocate Behavior to Components
8. Define Internal Interfaces
9. Select Design
SCHEDULE
10. Perform Effectiveness & Feasibility Analyses
11. Define Resources, Error Detection, & Recovery Behavior
12. Develop Validation Requirements/Validation Plans
13. Generate Documentation and Specifications

(a)


Find the top,
Then modify top-down.
8. Update System Boundary
7. Derive As-Built System Reqts.
7a. Modify Reqts & Arch. Constraints
6. Derive As-Built System Threads
6a. Modify System Threads
5. Aggregate to As-Built System Behavior
5a. Modify & Decompose System Behavior
4. Derive As-Built Behavior of Components
4a. Allocate Behavior to Components
3. Capture Component Hierarchy
3a. Refine Component Hierarchy
2. Capture Interfaces
2a. Define Interfaces
1.Define System Boundary
SCHEDULE
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
11. Capture Error Detection, Resource, & Recovery Behavior
12. Develop Test Plans
13. Generate Documentation and Specifications

(b)

Figure 3-17. Vitech MBSE Activities Timeline - Top Down (a) and (b) Reverse Engineering.

According to Long [48], three models are necessary and sufficient to completely specify a system: (1) control (functional behavior) model, (2) interface (I/O) model, and (3) physical architecture (component) model. Performance requirements/resources are captured with parts or combinations of one of these three models. These three models provide a basis for knowing when the SE of the system has been completed, i.e., when—within projected technology—an achievable design specification for all system components has been reached, and the system V&V plans are defined and fully traced.

The Vitech MBSE methodology that is taught as part of the tutorial includes methods in support of a set of learning objectives for each of the four top-level SE activities areas articulated in Figure 3-14. Details of each method and tooling support will not be described here; however, as an example, the learning objectives associated with Source Requirements and Analysis and Architecture/Synthesis are shown in Table 3-5. Additional details on methods associated with the Vitech MBSE methodology are also described by Baker and Long [49], although described in the context to what the authors refer to as the "System Logic Modeling (SLM)" Process.

Table 3-5. Learning Objectives and Sub-Activities for Vitech MBSE Top-Level SE Activities of Source Requirements Analysis and Architecture/Synthesis.

	Source Requirements & Analysis	**Architecture/Synthesis**
Objective	Identify structure and analyze requirements from a source.	Expand our understanding of the system.
Activities	1. Identify and extract requirements 2. Organize requirements 3. Analyze requirements 3.1 Discover and identify issues	1. Define: 1.1 System boundaries 1.2 Potential interfaces 1.3 Preliminary physical architecture components 1.4 Preliminary functionality

	3.2 Discover and identify risks 4. Establish requirements relationships 5. View the requirements graphically 6. Generate the requirements and related information in a table	2. Maintain traceability to originating requirements 3. Identify performance factors 4. Identify constraints 5. Continue to mitigate issues and risks

Methods used in the Vitech MBSE methodology to support the Functional/Behavior Analysis top-level activity is based on a set of visual behavior models and constructs in an executable graphical language known as Enhanced Function Flow Block Diagrams (EFFBDs). Other supporting visual modeling languages to support Functional/Behavior Analysis include standard FFBDs, N2 charts, and Behavior diagrams; each of these modeling constructs is described in greater detail by Long [51]. Note that the Vitech MBSE tool CORE does not currently support the standard visual modeling language standards of the UML® or OMG SysML™. This contrast, particularly with respect to EFFBDs, is described in greater detail in Section 3.6. Although an assessment of use of the UML in support of the Vitech MBSE methodology was described by Skipper in 2003 [52], it is not yet clear that UML and/or SysML are on the Vitech CORE product roadmap for future support.

Methods associated with the Vitech MBSE methodology to support the Design Verification and Validation (V&V) top-level activity include test plan development and test planning with best practices emphasizing that test planning begins during the originating requirements extraction and analysis phase. Test threads are also described with test paths specified as derived from system behavior. Software testing methods are highlighted as well as system testing methods. The primary system testing methods described by the MBSE methodology are summarized in Table 3-6.

Table 3-6. System Testing Methods Defined in the Vitech MBSE Methodology.

Functional Testing	Test conditions are set up to ensure that the correct outputs are produced, based upon the inputs of the test conditions. Focus is on whether the outputs are correct given the inputs (also called "black box" testing).
Structural Testing	Examines the structure of the system and its proper functioning. Includes such elements as performance, recovery, stress, security, safety, availability. Some of the less obvious elements are described below.
Performance	Examination of the system performance under a range of nominal conditions, ensures system is operational as well.
Recovery	Various failure modes are created and the system's ability to return to an operational mode is determined.
Interface	Examination of all interface conditions associated with the system's reception of inputs and sending of outputs.
Stress Testing	Above-normal loads are placed on the system to ensure that the system can handle them; these above-normal loads are increased to determine the system's breaking point; these tests proceed for a long period of time in an environment as close to real as possible.

3.4.2. Tool Support

There is no process framework tool offered by Vitech Corporation or third party provider that supports the Vitech MBSE methodology. Vitech does offer an MBSE tool set via its CORE® product suite.

3.4.3. Offering/Availability

In the past, a half-day tutorial on the Vitech MBSE methodology was offered at various INCOSE International Workshops and Symposia [47]. This tutorial was entitled "H0D: Model Based Systems Engineering for Project Success: The Complete Process (PM)" and was taught by James ("Jim") E. Long. More detailed, multi-day courses are offered through the Vitech training services (see http://vitechcorp.com/services/).

3.5 JPL State Analysis (SA)

3.5.1. Overview

State Analysis (SA) is a JPL-developed MBSE methodology that leverages a model- and state-based control architecture (see Figure 3-18), where *state* is defined to be "a representation of the momentary condition of an evolving system," and *models* describe how state evolves [40].

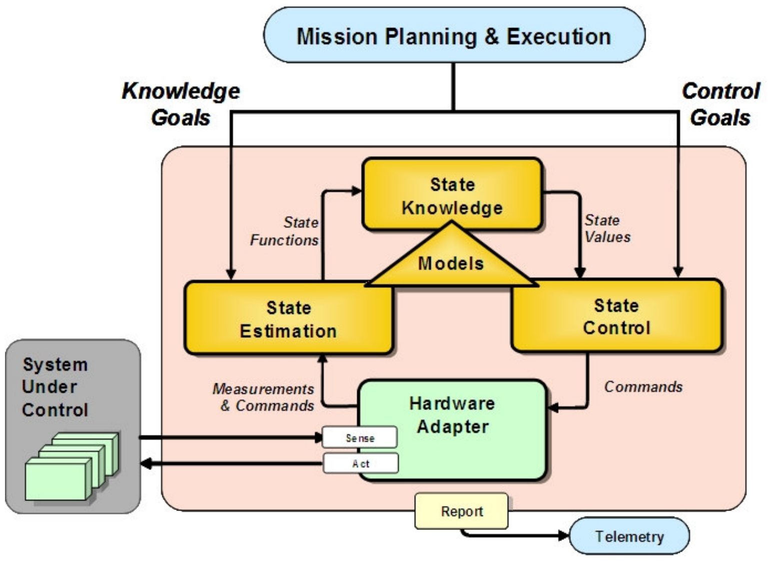


Figure 3-18. Model- and State-Based Control Architecture ("Control Diamond").

SA provides a process for capturing system and software requirements in the form of explicit models, thereby helping reduce the gap between the requirements on software specified by systems engineers and the implementation of these requirements by software engineers. Traditionally, software engineers must perform the translation of requirements into system behavior, hoping to accurately capture the system engineer's understanding of the system behavior, which is not always explicitly specified. In SA, model-based requirements map directly to software.

In SA, it is important to distinguish between the "state" of a system and the "knowledge" of that state. The real state may be arbitrarily complex, but ones knowledge of it is generally captured in simpler abstractions that one finds useful and sufficient to characterize the system state. These abstractions are called *state variables*. The known state of the system is the value of its state variables at the time of interest. Together, state and models supply what is needed to operate a system, predict future state, control toward a desired state, and assess performance.

Note: State defined in the context of SA extends the classical control theory definition of state (e.g., spacecraft position and attitude and corresponding rates) to include all aspects of the system that the system engineer is interested in for the purpose of control, and that might need to be estimated. This could include, for example, device operating modes and health, temperatures and pressures, resource levels (e.g., propellant; volatile and non-volatile memory) and any other "care abouts" for purposes of control [53].

Given the above definitions of state and state variables, it is useful to articulate the key features of the "Control Diamond" illustrated in Figure 3-18:

- **State is explicit**. The full knowledge of the state of the system under control is represented in a collection of state variables.
- **State estimation is separate from state control**. Estimation and control are coupled only through state variables. Keeping these two tasks separate promotes objective assessment of system state, ensures consistent use of state across the system, simplifies the design, promotes modularity, and facilitates implementation in software.
- **Hardware adapters provide the sole interface between the system under control and the control system**. They form the boundary of the state architecture, provide all the measurement and command abstractions used for control and estimation, and are responsible for translating and managing raw hardware input and output.
- **Models are ubiquitous throughout the architecture**. Models are used both for the execution (estimating and controlling state) and higher-level planning (e.g., resource management). SA requires that the models be documented explicitly, in whatever form is most convenient for the given application.
- **The architecture emphasizes goal-directed closed-loop operation**. Instead of specifying desired behavior in terms of low-level open-loop commands, SA uses *goals*, which are constraints on state variables over a time interval.
- **The architecture provides a straightforward mapping into software**. The control diamond elements can be mapped directly into components in a modular software architecture, such as Mission Data System (MDS).[13]

In addition to these features of the model- and state-based control architecture on which SA is based, there are a set of three core tenets that serve as guiding principles behind the SA methodology:

- Control subsumes all aspects of the system operation. It can be understood and exercised intelligently only through models of the system under control. Therefore, a clear distinction must be made between the *control system* and the *system under control*.[14]

[13] MDS is an embedded software architecture intended to provide multi-mission information and control architecture for robotic exploration spacecraft [54]. The regular structure of SA is replicated in the MDS architecture, with every SA product having a direct counterpart in the software implementation.

[14] A control system has cognizance over the system under control. This means that the control system is aware of the state of the system under control, and it has a model of how the system under control behaves. The premise of SA is that this knowledge of state and its behavior is complete, i.e., no other information is required to control a system [53].

- Models of the system under control must be explicitly identified and used in a way that assures consensus among systems engineers. Understanding state if fundamental to successful modeling. Everything we need to know and everything we want to do can be expressed in terms of the state of the system under control.
- The manner in which models inform software design and operation should be direct, requiring minimal translation.

The SA methodology defines an iterative process for state discovery and modeling, which allows the models to evolve as appropriate across the project lifecycle. (A tool known as the *State Database* compiles information that is traditionally documented in a variety of systems engineering artifacts [55].) In addition, mechanisms are specified by which the models are used to design software and operations artifacts. In summary then, SA provides a methodical and rigorous approach for the following three primary activities:

1. **State-based behavioral modeling**. Modeling behavior in terms of system state variables and the relationships between them.
2. **State-based software design**. Describing the methods by which objectives will be achieved.
3. **Goal-directed operations engineering**. Capturing mission objectives in detailed scenarios motivated by operator intent.

It should be noted that state-based behavior modeling directly influences/contributes to state-based software design and goal-directed operations engineering. This makes the SA approach to systems engineering ideally suited for application to complex embedded systems, autonomy, and closed-loop commanding. In fact, for JPL managed space missions with such characteristics, this author recommends that the SA methodology be fully exploited.

A detailed description of the methods ("HOWs") for each of the three aspects of SA identified above will not be described here as there are a myriad of published resources and training materials available to the JPL community (see [56] and [57] for example).

At first blush, SA appears to be a significant paradigm shift from document- and model-driven system design approaches that utilize the traditional functional analysis & decomposition approach to systems engineering. In actuality, SA is highly complementary to functional analysis; both approaches add value and reduce risk in the development of complex systems.

Relation to Functional Analysis & Decomposition

In support of a JPL internal research and development (R&D) activity entitled Model-Based Engineering Design (MBED) with the intent of demonstrating infusion of model-based concepts to engineering design applied to the formulation phase of a space system project lifecycle, Ingham [58] showed that SA could be synthesized with a functional analysis model-driven process as a more comprehensive and rigorous approach to system behavior modeling.

Figure 3-19 illustrates the result of the iterative decomposition process that is part of traditional functional analysis & decomposition that ultimately results in a hierarchy of functions, physical components (product breakdown structure) and requirements and the linkages between the functional, physical, and requirements hierarchies [58].

What Ingham and his colleagues showed as part of the MBED FY06 effort was that it was possible to augment the functional analysis schema used by the Vitech CORE® MBSE tool (which is patterned after the linkages and elements shown in Figure 3-19, consistent with traditional functional analysis) with the SA elements of *state variables*, *commands*, and *measurements* (see Figure 3-20).

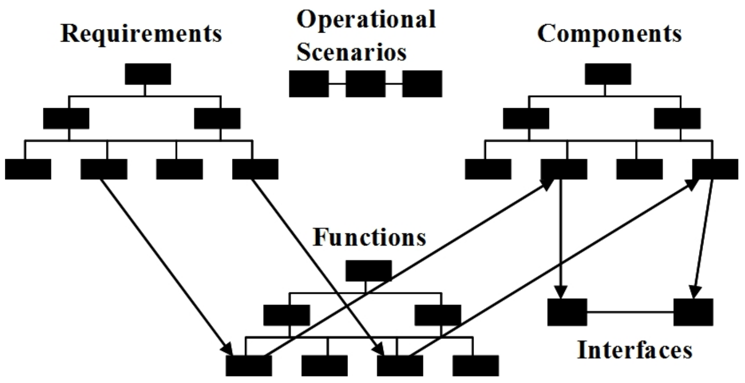
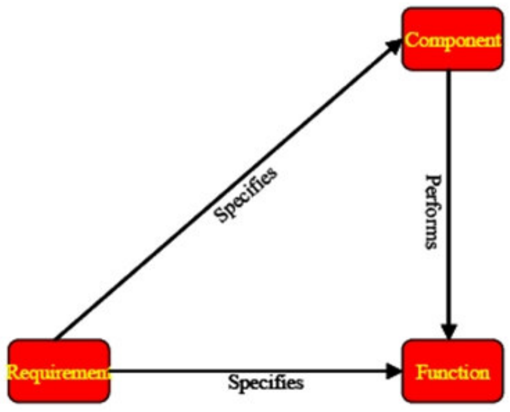
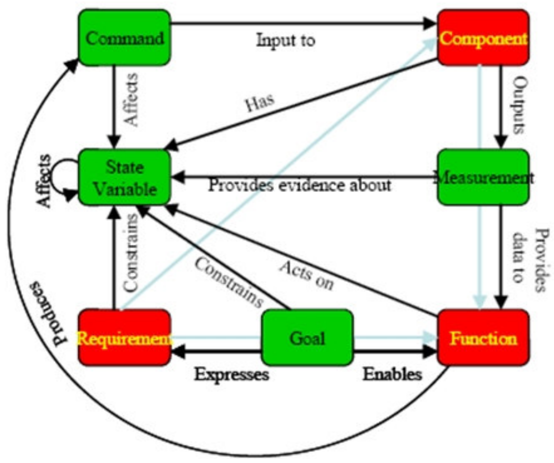


Figure 3-19. Conceptual Layout of Requirements, Functions, and Physical Components. (Operational Scenarios are also Shown.)





(a) (b)

Figure 3-20. (a) Functional Analysis Elements and Relationships, (b) Elements and Relationships of State Analysis Synthesized with Functional Analysis.[15]

This ability to synthesize functional and state analysis as demonstrated under the MBED R&D task for the FY06 year highlighted the complementary nature of these two MBSE methodologies and promises to yield significant benefits, including:

- Better understanding and documentation of designed behavior
- Earlier identification of unexpected design challenges
- Improved traceability to developed software
- More robust fault protection in the design system

[15] Although the SA element of *goals*, is shown as part of the integrated schema in Figure 3-20b, the goal-based operations engineering aspect of SA was not demonstrated.

Relation to Hazard Analysis

At the heart of the discipline of system safety is a practice known as *Hazard Analysis*, where *hazard* is defined as "a state or set of conditions that, together, with other conditions in the environment, may lead to an accident (loss event)" [59].[16] Hazard Analysis is used for developing safety-related requirements and design constraints, validating those requirements and constraints, preparing operational procedures and instructions, test planning, and management planning. Hazard analysis serves as a framework for design for safety and as a checklist to ensure management and technical responsibilities for safety are accomplished. All hazard analysis techniques rest on an underlying model of how accidents are assumed to be caused. Most traditional hazard analysis techniques are based on causal-chain accident models [60]. Fault Tree Analysis (FTA) and Failure Modes and Effects Criticality Analysis (FMECA) are examples of traditional, event-based hazard analysis techniques.

A new hazard analysis technique that is being pursued by Leveson and her colleagues is known as the STAMP-Based Hazard Analysis, or *STPA* for short [61]. *STAMP* stands for Systems Theoretic Accident Modeling and Process, and it is an accident model in which accidents are conceived as resulting not from component failures, but from inadequate control of safety-related constraints on the design, development, and operation of the system. The most basic concept in STAMP is not an event, but a constraint. In STAMP, safety is viewed as a control problem, i.e., accidents occur when component failures, external disturbances, and/or dysfunctional interactions among system components are not adequately handled. The control processes that enforce these constraints must limit system behavior to the safe adaptations implied by the constraints. It is this "controls"-based aspect of STAMP and the derived STPA methodology—together with the new technical standards being levied by NASA on software safety [62]—that have initiated a new task at JPL aimed at the possible harmonization of the controls-based STPA hazard analysis methodology with the controls-based SA MBSE methodology.

It is important to note that STPA does not supplant a traditional or model-based systems engineering process but rather augments it with a system safety process (see Figure 3-21) [59]. It is also important to acknowledge that hazard analysis is only one part of a comprehensive system safety process. Other important elements such as Intent Specifications [63] and component-based systems engineering [64]] used in concert with hazard analysis techniques such as STPA provide an *integrated* approach to system, software, and safety engineering for complex, safety-critical systems [59].

[16] An *accident*, as defined by Leveson [60], is "an undesired and unplanned (but not necessarily unexpected) [loss] event that results in (at least) a specified level of loss." *Safety*, in this context, is defined as "freedom from accidents or losses." System safety can be measured on a continuum where a system is considered to be either safe (no loss) or at some increasing level of loss.

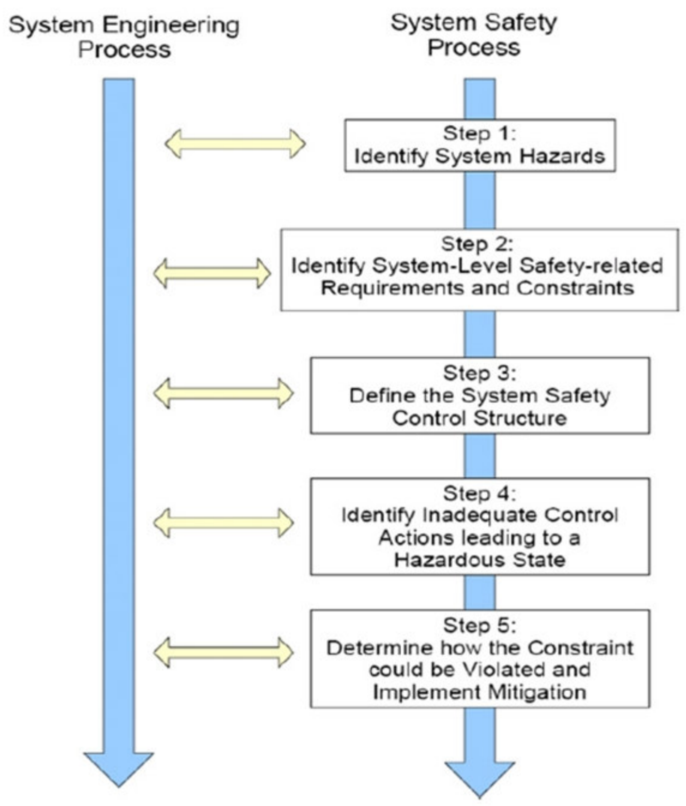

System Engineering Process
System Safety Process
Step 1: Identify System Hazards
Step 2: Identify System-Level Safety-related Requirements and Constraints
Step 3: Define the System Safety Control Structure
Step 4: Identify Inadequate Control Actions leading to a Hazardous State
Step 5: Determine how the Constraint could be Violated and Implement Mitigation

Figure 3-21. Integrated Approach to Design for Safety.

3.5.2. Tool Support

Tool support for State Analysis (SA) is provided by the State Database [55], which utilizes a Structured Query Language (SQL)-compliant relational database management system (RDBMS) such as Oracle® with a front end user interface. This tool supports developing, managing, inspecting, and validating system and software requirements capture as part of the SA process.

To support system safety engineering techniques such as Hazard Analysis and Intent Specifications, commercial tools such as Specifications Tool and Requirements Methodology (SpecTRM) and the formal requirements language used in that tool, SpecTRM-RL, as well as SpecTRM-GSC (Generic Spacecraft Component) are available from Safeware Engineering (see http://safeware-eng.com/).

3.5.3. Offering/Availability

State Analysis (SA) is a JPL-developed MBSE methodology and the offering is available by means of a series of courseware and tutorials offered by SA experts. These courses are offered through JPL Professional Development on a periodic, as-needed basis, or through reimbursable contract agreements with industry partners. As part of the hands-on exercises, access to the State Database tool and supporting training in use of the tool is provided. In March 2008, a full-day tutorial as well as a distilled evening session entitled "State Analysis for Systems Engineers" was offered to members of the Los Angeles Chapter of INCOSE in collaboration with JPL and the California Institute of Technology [65].

3.6 Dori Object-Process Methodology (OPM)

3.6.1. Overview

Dori defines the Object-Process Methodology (OPM) as a formal paradigm to systems development, lifecycle support, and evolution [66]. It combines formal yet simple visual models known as Object-Process Diagrams (OPDs) with constrained natural language sentences known as Object-Process Language (OPL) to express the function (what the system does or designed to do), structure (how the system is constructed), and behavior (how the system changes over time) of systems in an integrated, single model. Every OPD construct is expressed by a semantically equivalent OPL sentence or part of a sentence and vice versa. OPL is a dual-purpose language, oriented towards humans as well as machines [66].

The premise of OPM is that everything in the universe is ultimately either an *object* or a *process*. At the modeling level, OPM is built on top of three types of entities: objects, processes, and states, with objects and processes being the higher-level building blocks, collectively called *things*. OPM formally defines these entities as follows:

- An **object** is a thing that exists or has the potential of existence, physically or mentally.
- A **process** is a pattern of transformation that an object undergoes.
- A **state** is a situation an object can be at.

Objects exist, and processes transform the objects by generating, consuming, or affecting them. States are used to describe (stateful) objects, and are not standalone things. At any point in time, each stateful object is at some state. The symbol for objects and processes are rectangles and ellipses, respectively. The first letter of object and process names is always capitalized and process names are preferably expressed as gerunds (end in *-ing*) indicating they are active, dynamic things. The symbol for state is a "roundtangle" (rounded corner rectangle) [credited to D. Harel]. State names start with a lower-case letter. An elementary example that illustrates a simple OPD and OPL sentences, adapted from the OPM textbook [66], is illustrated in Figure 3-22 below.

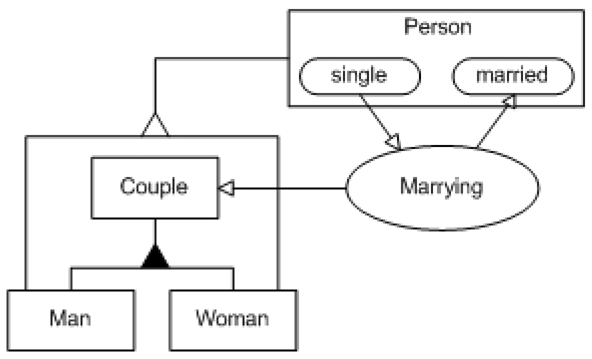
OPD	OPL
 Person single married Couple Marrying Man Woman	**Person** can be **single** or **married**. **Marrying** changes **Person** from **single** to **married**. **Man** and **Woman** are **Persons**. **Marrying** yields **Couple**. **Couple** consists of **Man** and **Woman**.

Figure 3-22. Simple OPD and OPL Modeling the Process of *Marrying*, which Yields *Couple*, which Consists of *Man* and *Woman* (in the Traditional Sense), Each Being a Specialization of *Person*.

In this simple visual OPD model, the open triangle that links **Person** to **Man** and **Woman** represents *specialization*, whereas the filled-in triangle that links **Couple** to **Man** and **Woman** represents *aggregation*. Lines with closed arrowheads represent an *input-output* link pair, a *consumption link*, or a *result link* depending on its context. In this example, the link from the state **single** to the process **Marrying** is an input link, the link from **Marrying** to **married** is an output link, and the link from **Marrying** to **Couple** is a result link. The same link with an inverse direction (from an object to a process) would be a consumption link. Tables A-1 through A-4 in the Appendix contain the OPM symbols as well as the core syntax and semantics of Object-Process Diagrams (OPDs)—the graphical part of OPM.

As Tables A-2 through A-4 show, links are further classified in OPM to be either *structural links*, which express persistent, long-term relations among objects or among processes in the system, or *procedural links*, which express the behavior of the system. In Figure 3-22, the input, output, and result links represent procedural links while the links exhibiting specialization and aggregation represent structural links. Since the structural and procedural links are expressed in the same diagram, they provide a complete picture of the system in a single graphical model, which is complemented by a textual one [66].

OPM supports a very rich set of modeling semantics, far beyond the simple example described above. For a complete description of the semantics of the graphical elements that make up OPDs as well as constructs for OPL sentences, the reader is referred to the OPM textbook [66]. Our interest for purposes of the MBSE methodology survey is not to focus on the semantics of the modeling constructs that describe OPDs and OPL sentences but rather the processes and methods to support lifecycle systems engineering; specifically, model-based systems engineering (MBSE).

Dori has shown that OPM can be used to model systems, natural systems as well as artificial systems [66].[17] In this respect, OPM is a holistic systems paradigm. It can be used to document functions of a system architecture, which of course is a key deliverable in a systems engineering process. (Recall that architecture prescribes the combination of the system's structure and behavior that attains its required functions under given constraints.) A major contribution of OPM to systems science and engineering is the precise semantics and syntax it ascribes to graphic symbols (used in OPDs) and the unambiguous association of graphic symbols with natural language constructs (i.e., OPL sentences).

In addition, OPM manages system complexity through three refinement/abstraction mechanisms: *Unfolding/folding*, which is used for refining/abstracting the structural hierarchy of a thing; *In-zooming/out-zooming*, which exposes/hides the inner details of a thing within its frame; and *state expressing/suppressing*, which exposes/hides the states of an object [66],[67]. Using these mechanisms, OPM enables specifying a system to any desired level of detail without losing legibility and comprehension of the resulting specification.

Dori notes that software developers often refer to the orderly development of software as *software engineering*, the *software process*, or shortly, *process* [66]. In OPM, the term "process" is a reserved word and has a very specific semantics of a thing that transforms and object; hence, OPM refers to the entire lifecycle of systems as *system evolution* rather than process. It has a name, however, that retains the word process—"OPM system

[17] Dori defines *system* as "an object that carries out or supports a significant function (as opposed to a non-significant function)."

process," which prescribes the development and lifecycle support of not just software systems, but systems in general, where software may be a component or a subsystem.

Chapter 11 of the OPM textbook describes an OPM model of system lifecycle phases, which sets the stage and provides a base metamodel on which certain aspects are further elaborated in an additional reference by Dori and Reinhartz-Berger [67]. Here, OPM is used to specify a generic OPM-based system development process. Dori and Reinhartz-Berger refer to the process of creating this type of methodology metamodel, described in the paper, as **reflective metamodeling** and to the methodology itself as a **reflective methodology** [67],[68]. The former models a methodology by the means and tools that the methodology itself provides; the latter is a self-contained approach that does not require auxiliary means or external tools to model itself. The distinction between these powerful techniques is subtle but important. The remainder of this section captures the essential threads of discussion directly adapted from the paper of Dori and Reinhartz-Berger paper [67]. They are included here for completeness to assist the reader and to complete our discussion of Dori OPM as a candidate MBSE methodology.

Author's Note: The following supplemental material is adapted essentially verbatim from Dori and Reinhartz-Berger [67] and used by permission.18 Only the figure and reference numbers have been altered to be consistent with the numbering scheme used by this survey report. Annotations captured as footnotes are added where deemed necessary to clarify certain concepts.

The System Diagram, which is labeled **SD** and shown in Figure 3-23, is the top-level specification of the OPM metamodel. It specifies **Ontology**, **Notation**, and the **System Developing** process as the major OPM features (characterizations). **Ontology** includes the basic elements in OPM, their attributes, and the relations among them. For example, objects, processes, states, and aggregations are all OPM elements. The **Notation** represents the **Ontology** graphically (by OPDs) or textually (by OPL sentences). For example, a process is represented graphically in an OPD by an ellipse, while an object is symbolized by a rectangle.

The **System Developing** process, also shown in **SD**, is handled by the **User**, who is the physical and external (environmental) object that controls (is the agent of) the process. This process also requires **Ontology** and **Notation** as instruments (inputs) in order to create a **System**.

[18] Dov Dori, Technion, Israel Institute of Technology (private communication), May 21, 2008.

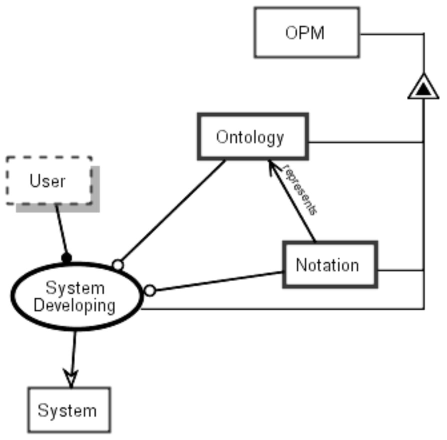
SD 	**OPM** exhibits **Ontology** and Notation, as well as **System Developing**. **Notation represents Ontology** **System Developing** requires **Notation** and **Ontology**. **System Developing** yields **System**. **User** is environmental and physical. **User** handles **System Developing**.

Figure 3-23. The Top Level Specification of the OPM Metamodel.

SD1

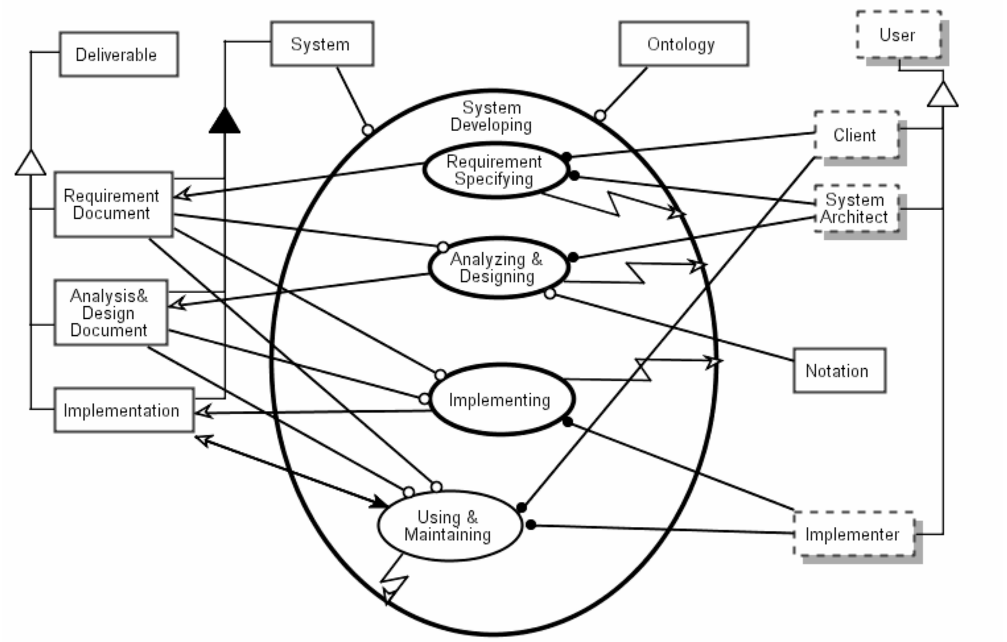


Figure 3-24. Zooming into *System Developing*.

Note: The thick ellipses used to model key processes (i.e., System Developing and its sub-processes Requirement Specifying, Analyzing & Designing, and Implementing) denote in-zoomed processes or processes which are in-zoomed in a lower level OPD.[19]

The OPL paragraph, which is equivalent to **SD**, is also shown in Figure 3-23. Since OPL is a subset of English, users who are not familiar with the graphic notation of OPM can validate their specifications by inspecting the OPL sentences. These sentences are automatically generated on the fly in response to the user's graphic input which creates the OPDs [69].

[19] Ibid.

Due to space limitations and the equivalence of OPM graphical and textual notations, we use only the OPD notation in the rest of the paper.

Zooming into **System Developing**, **SD1** (Figure 3-24) shows the common sequential[20] stages of system developing processes: **Requirement Specifying**, **Analyzing & Designing**, **Implementing**, and **Using & Maintaining**. All of these processes use the same OPM **Ontology**, a fact that helps narrowing the gaps between the different stages of the development process. **SD1** shows that the **Client** and the **System Architect**, which, along with the **Implementer**, specialize **User**, handle the **Requirement Specifying** sub-process. **Requirement Specifying** takes OPM **Ontology** as input and creates a new **System**, which, at this point, consists only of a **Requirement Document**. The termination of **Requirement Specifying** starts **Analyzing & Designing**, the next sub-process of **System Developing**.

The Requirements Specifying Stage

In SD1.1 (Figure 3-25), **Requirement Specifying** is zoomed into, showing its four subprocesses. First, the **System Architect** and the **Client** define the problem to be solved by the system (or project). This **Problem Defining** step creates the **Problem Definition** part of the current system **Requirement Document**. Next, through the **Requirement Reusing** sub-process, the **System Architect** may reuse requirements that fit the problem at hand and are adapted from any existing **System** (developed by the organization). Reuse helps achieve high quality systems and reduce their development and debugging time. Hence, when developing large systems, such as Web applications or real-time systems, it is important to try first to reuse existing artifacts adapted from previous generations, analogous systems, or commercial off-the-shelf (COTS) products that fit the current system development project. Existing, well-phrased requirements are often not trivial to obtain, so existing relevant requirements should be treated as a potential resource no less than code. Indeed, as the OPD shows, reusable artifacts include not only software or hardware components (which traditionally have been the primary target for reuse), but also requirements.

After optional reuse of requirements from existing systems (or projects), the **System Architect** and the **Client**, working as a team, add new **Requirements** or update existing ones. This step uses OPM **Ontology** in order to make the **Requirement Document** amenable to be processed by other potential OPM tools, and in particular to an OPL compiler. The bi-modal property of OPM, and especially the use of OPL, a subset of English (and potentially any natural language), enables the **Client** to be actively involved in the critical **Requirement Specifying** stage. Moreover, since the **System Architect** and the **Client** use OPM **Ontology** in defining the new requirements, the resulting **Requirement Document** is indeed expressed, at least partially, in OPL in addition to explanations in free natural English. Such structured OPM-oriented specification enables automatic translation of the **Requirement Document** to an OPM analysis and design skeleton (i.e., a skeleton of an OPD-set and its corresponding OPL script). Naturally, at this stage the use of free natural language beside OPM seems mandatory to document motivation, alternatives, considerations, etc.

[20] The time line in an OPD flows from the top of the diagram downwards, so the vertical axis within an in-zoomed process defines the execution order. The sub-processes of a sequential process are depicted in the in-zoomed frame of the process stacked on top of each other with the earlier process on top of a later one. Analogously, subprocesses of a parallel process appear in the OPD side by side, at the same height.

Finally, the **Requirement Adding** process results in the Boolean object "**Is Backtracking Required?**", which determines whether **System Developing** should be restarted. If so, **Development Process Backtracking** invokes the entire **System Developing**. Otherwise, **Requirement Specifying** terminates, enabling the **Analyzing & Designing** process to begin.

SD1.1

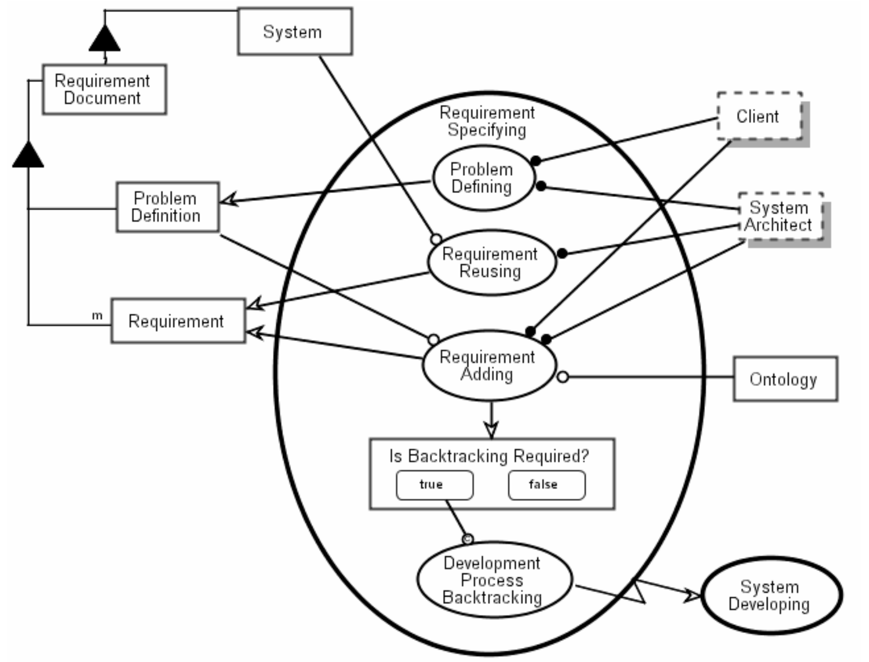


Figure 3-25. Zooming into *Requirement Specifying*.

The Analyzing and Designing Stage

During the **Analyzing & Designing** stage, shown in **SD1.2** (Figure 3-26), a skeleton of an **OPL Script** is created from the **Requirement Document** for the current system. As noted, in order to make this stage as effective and as automatic as possible, the **Requirement Document** should be written using OPM, such that the resulting OPL script can be compiled. The **System Architect** can then optionally reuse analysis and design artifacts from previous systems (projects), creating a basis for the current system analysis and design. Finally, in an iterative process of **Analysis & Design Improving** (which is in-zoomed in **SD1.2.1**), the **System Architect** can engage in **OPL Updating**, **OPD Updating**, **System Animating**, **General Information Updating**, or **Analysis & Design Terminating**.

Any change a user makes to one of the modalities representing the model triggers an automatic response of the development environment software to reflect the change in the complementary modality. Thus, as **SD1.2.1** (Figure 3-27) shows, **OPD Updating** (by the **System Architect**) affects the **OPD-set** and immediately invokes **OPL Generating**, which changes **OPL Script** according to the new **OPD-set**. Conversely**, OPL Updating** (also by the **System Architect**) affects the **OPL Script**, which invokes **OPD Generating**, reflecting the OPL changes in the **OPD-set**.

SD1.2

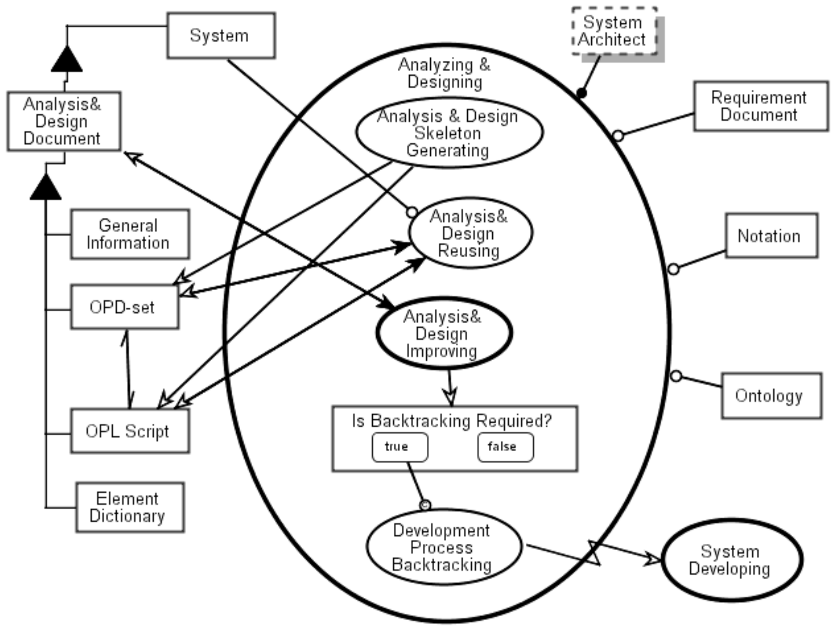


Figure 3-26. Zooming into Analyzing & Designing.

SD1.2.1

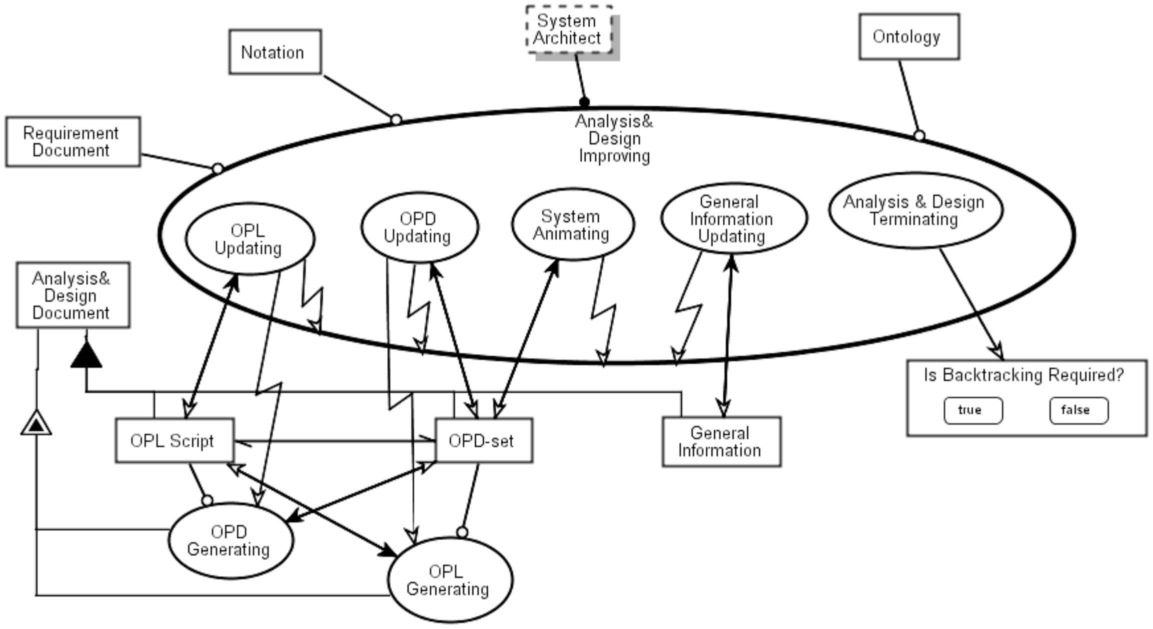


Figure 3-27. Zooming into *Analysis & Design Improving*.

Since OPM enables modeling system dynamics and control structures, such as events, conditions, branching, and loops, **System Animating** simulates an **OPD-set**, enabling **System Architects** to dynamically examine the system at any stage of its development. Presenting live animated demonstrations of system behavior reduces the number of design errors percolated to the implementation phase. Both static and dynamic testing help detect discrepancies, inconsistencies, and deviations from the intended goal of the system. As part of the dynamic testing, the simulation enables designers to track each of the system scenarios before writing a single line of code. Any detected mistake or omission is corrected at the model level, saving costly time and efforts required within the implementation level. Avoiding and eliminating design errors as early as possible in the system development process and keeping the documentation up-to-date contribute to shortening the system's delivery time ("time-to-market").

Upon termination of the **Analysis & Design Improving** stage, if needed, the entire **System Developing** process can restart or the **Implementing** stage begins.

The Implementing Stage

The **Implementing** stage, in-zoomed in **SD1.3** (Figure 3-28), begins by defining the **Implementation Profile**, which includes the target **Language** (e.g., Java, C++, or SQL) and a default **Directory** for the artifacts. Then, the **Implementation Skeleton Generating** process uses the **OPL Script** of the current system and inner **Generation Rules** in order to create a skeleton of the **Implementation**. A **Generation Rule** saves pairs of OPL sentence types (templates) and their associated code templates in various target **Languages**.

The initial skeleton of the **Implementation**, which includes both the structural and behavioral aspects of the system, is then modified by the **Implementer** during the **Implementation Reusing** and **Implementation Improving** steps. In the **Testing & Debugging** stage, the resulting **Implementation** is checked against the **Requirement Document** in order to verify that it meets the system requirements defined jointly by the **Client** and the **System Architect**. If any discrepancy or error is detected, the **System Developing** process is restarted, else the system is finally delivered, assimilated and used. These sub-processes are embedded in the **Using & Maintaining** process at the bottom of **SD1** (Figure 3-24). While **Using & Maintaining** takes place, the **Client** collects new requirements that are eventually used when the next generation of the system is initiated. A built-in mechanism for recording new requirements in OPM format while using the system would greatly facilitate the evolution of the next system generation [66].

SD1.3

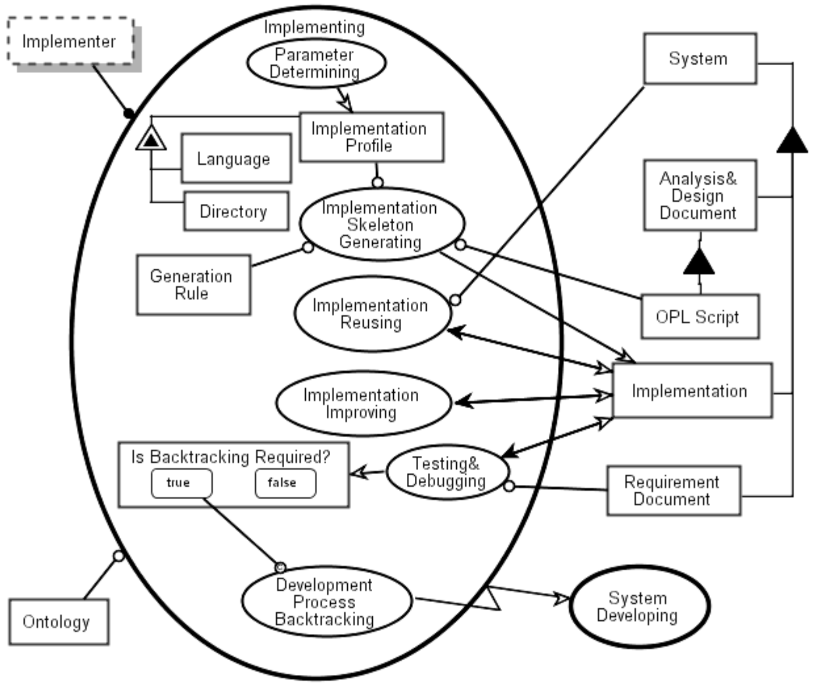


Figure 3-28. Zooming into *Implementing*.

3.6.2. Tool Support

The latest revision of the commercial version at the time of this writing is OPCAT v3.0. This product supports the concepts described in this section related to the OPM metamodel for the system development process, including modeling support of the System Diagram (SD). In addition the commercial OPCAT has a configurable template for all kinds of document artifacts, including but not limited to System Overview, The Current State, Future Goals, Business or Program Constraints, and Hardware and Software Requirements. OPCAT has facilities for animated simulation, requirements management, and many other advanced features.

A restricted version of OPCAT 3 for evaluation and academic use only can be downloaded from the official OPCAT website at:
http://www.opcat.com/downloads/restricted/

Information about the commercial version of OPCAT as well as product documentation and support and contact information can be found under the official OPCAT website at:
http://www.opcat.com/

3.6.3. Offering/Availability

Commercial OPCAT software for OPM systems modeling, systems engineering and lifecycle support, as well as professional services, and education & training can be obtained via: http://www.opcat.com/

4. Role of OMG™ UML®/SysML™

The Unified Modeling Language™ (UML®) and Systems Modeling Language™ (OMG SysML™) are visual modeling language standards managed under the auspices of the Object Management Group™ (OMG™); an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications [71],[72].

UML and OMG SysML are intended to be complementary. SysML was developed in partnership between the OMG and INCOSE and is based on the UML, specifically, UML 2.0 (see Figure 4-1) [38],[72]. SysML provides an additional set of modeling diagrams to model complex systems that include hardware, software, data, procedures and other system components. Together, UML and SysML go a long way to help unify what has historically been a communication chasm between the systems and software engineering communities.

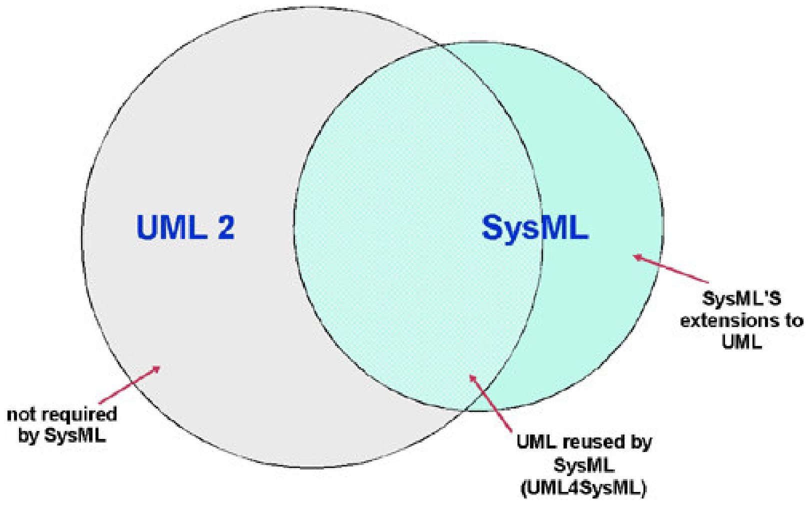


Figure 4-1. UML 2 and OMG SysML.

It is important to note that the UML and SysML are not software or systems methodologies but rather visual modeling languages that are agnostic to any one specific methodology. Several commercially-offered model-based systems and software engineering methodologies, including most of the MBSE methodologies surveyed in this study, incorporate the UML and/or SysML into specific methods and artifacts produced as part of the methodology.

Use of industry-standard visual modeling languages allows members of a systems and software lifecycle development activity to communicate in an unambiguous fashion. Further, such industry standards facilitate the ability to leverage commercial visual modeling tools as well as education and training provided by industry and academia. It is expected that more-and-more junior engineers in the discipline of systems and software engineering will be versed in one or both of these visual modeling standards in the coming years.

Additional information about UML and OMG SysML can be found at the following web sites:

- http://www.uml.org/
- http://www.omgsysml.org/

Relevance to State Analysis

With respect to State Analysis (SA), the UML component diagram is a modeling artifact that is currently used to describe the static structure of software components, although in somewhat a non-normative fashion. The term "non-normative" means that the visual models are not fully compliant with the standard; this is not a problem provided the models are clearly labeled as non-normative. It is recommended that SA component diagrams utilize the UML 2 standard and clearly delineate non-normative models.

UML and OMG SysML are extensible by means of *profiles* and the use of *stereotypes*, which allows these visual modeling standards to be tailored to specific domain areas or areas of application. Because of this extension/tailoring mechanism, it is recommended that, in addition to UML component diagrams, other UML/SysML structure and behavior diagrams be explored for adoption in the SA MBSE methodology, for example, state effects diagrams and state timelines.

Relevance to EFFBDs (or other "de-facto" MBSE visual modeling standards)

A recommendation for the MBSE tool vendors that currently do not support UML and/or OMG SysML is to add this capability to the product roadmap as soon as possible; particularly, SysML. The advantage of using industry standard visual modeling languages over vendor-specific modeling languages is clear and does not warrant debate. Some MBSE tools, for example, only support the Enhanced Function Flow Block Diagram (EFFBD) visual modeling capability, which in some cases (e.g., Vitech CORE/COREsim) support executable modeling constructs that allows the systems engineer to run discrete-event simulations based on EFFBD models. This is a very powerful capability and there is no technical reason that such an executable capability could not be added to OMG SysML diagrams such as activity diagrams and state diagrams. Bock [73],[74] has documented how both UML (UML 2 specifically) and SysML can be used for activity modeling and how these standards can extended to fully support EFFBDs.

5. Role of OMG™ MDA® and Executable UML Foundation

5.1 Model-Driven Architecture

An area of active research in the software architecture modeling community is the OMG's Model-Driven Architecture® (MDA®) initiative. MDA reflects OMG's approach to using models in software development to help achieve the vision of integrated systems and applications that can be deployed, maintained and integrated with far less cost and overhead than traditional approaches [75],[76],[77].

The primary goals of MDA are portability, interoperability, and reusability through architectural separation of concerns. Separation of concerns represents an established best practice of separating the specification of operation of a system from the details of the way that system uses the capabilities of its platform.

As an architectural framework, MDA prescribes certain kinds of models to be used, how those models may be prepared and the relationships of different kinds of models. To be more precise, these models are actually *viewpoint models* or *views* that represent a system from the perspective of a chosen viewpoint. The MDA framework specifies three viewpoints on a system, a computation independent viewpoint, a platform[21] independent viewpoint, and a platform specific viewpoint [75]. The three viewpoint models (views) that are associated with these viewpoints are briefly described below:

- Computation Independent Model (CIM) – A view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification.
- Platform Independent Model (PIM) – A view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.
- Platform Specific Model (PSM) – A view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

A graphic depiction of a typical software lifecycle process in applying MDA and how these models relate is depicted in Figure 5-1 [78].

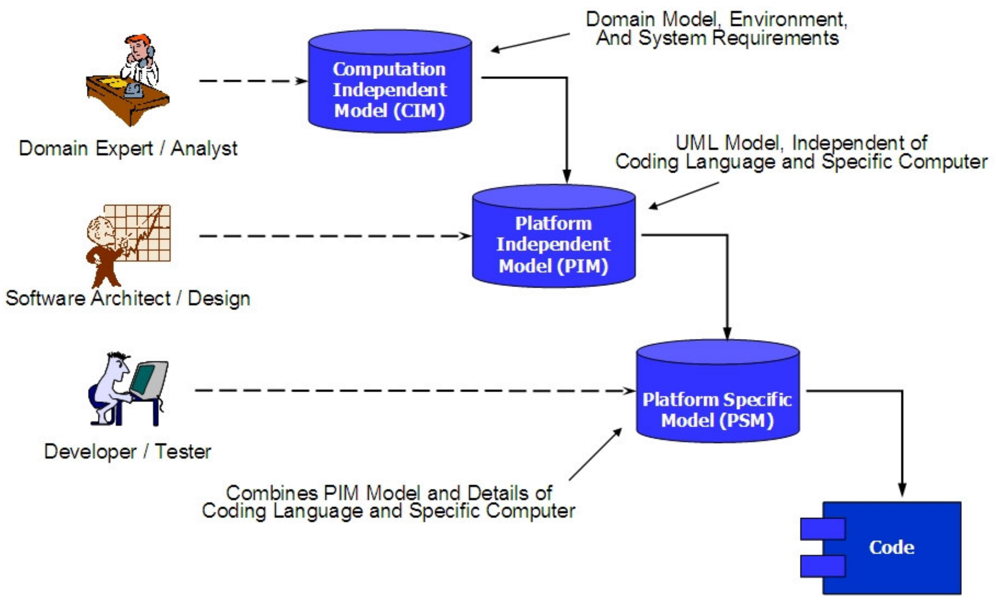

Domain Model, Environment, And System Requirements
Computation Independent Model (CIM)
Domain Expert / Analyst
UML Model, Independent of Coding Language and Specific Computer
Platform Independent Model (PIM)
Software Architect / Design
Platform Specific Model (PSM)
Developer / Tester
Combines PIM Model and Details of Coding Language and Specific Computer
Code

Figure 5-1. Applying MDA: Typical (Software) Process.

As an architectural approach, MDA provides for, and enables tools to be provided for:

- Specifying a system independently of the platform that supports it
- Specifying platforms
- Choosing a particular platform for the system
- Transforming the system specification into one for a particular platform

[21] OMG MDA defines a *platform* as a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.

There is a great deal more to OMG MDA including metamodeling, model transformation, and pervasive services. The interested reader is also encouraged to review plethora of white papers, presentations, online materials, and textbooks on the subject.

A link to the official OMG MDA website can be found at:

- http://www.omg.org/mda/

More recently, there has been an effort led by Cloutier [78],[79] to investigate the applicability of OMG MDA to the discipline of systems engineering; specifically, model-based systems engineering (MBSE). Cloutier hypothesizes that MDA may provide the types of productivity gains in systems engineering efforts on par with the kinds of productivity gains that have been demonstrated in the software engineering community. It is suggested that perhaps as much as 10-20% efficiency improvement may be realized on existing systems engineering projects once the methodology is understood and adapted for systems engineering [79].

A graphical depiction of applying OMG MDA to a typical systems engineering lifecycle process is shown in Figure 5-2 along with the various artifacts and deliverables associated with each MDA view, and the applicability of a common model repository and configuration and release tools that would comprise a MBSE tool suite [78].

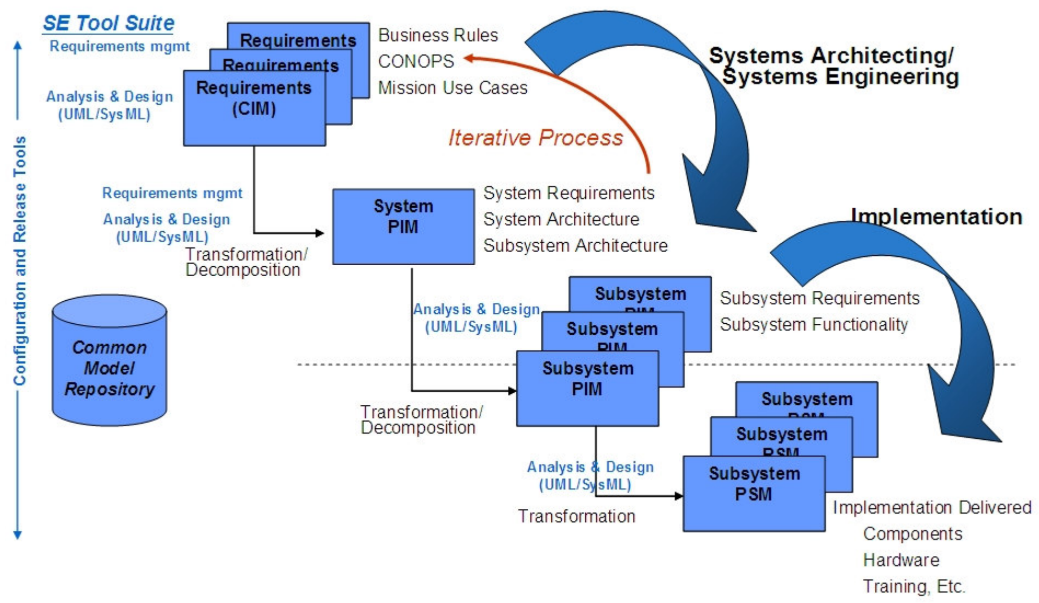


Figure 5-2. Applying MDA: Systems Engineering Process.

In the MDA MBSE approach, the CIM view plays a critical role in the lifecycle process. It is used to capture and model the concept of operations (CONOPS) for the system, taking a black-box view, and detailing how the system will interact with other systems and external actors. This CIM view is analogous to an operational architecture [79].

Also to be considered as part of the CIM view are the system goals, system requirements (in context), system stakeholder needs, and business rules that impact the system. This is not an exhaustive list of models that comprise the MBSE CIM view, but some of the more notable ones. It is expected that an OMG SysML-compliant MBSE tool would be used capture use case diagrams, and sequence diagrams or activity diagrams that model the mission use cases and mission scenarios.

These diagrams would then be "transformed" (in an MDA context) to the next lower level of detail with greater specificity into the system PIM view. Where the CIM view defines WHAT the system would do as it interacts with other systems and external actors, the PIM view defines HOW it performs those capabilities or functions at the system level [79]. It is expected that in the near term, this would be a manual transformation process. As future tooling becomes more mature that provides full support of MDA in an MBSE context, then greater automation of the transformation process could be introduced.

The PIM level of the system model will represent the system architecture, and the allocation of customer requirements to the system requirements. Unlike the PIM view for software systems that represents the computing platform, the PIM view for MBSE represents the complete deployment platform (e.g., tank, aircraft, ship, etc.). For MDA applied to MBSE, it may be necessary to create a multi-level PIM view to capture the necessary levels of detail to represent the complexity of the entire system while not providing too much detail in any single model [79]. In the PIM view, the MBSE process decomposes the business rules and system requirements and they are transformed into a more specific detailed model of the system. Here, the major parts of the system begin to take form as the capabilities of the system are more thoroughly defined and derived. Common capability groups (allocated from the system specifications) and logical subsystems begin to emerge from these groupings.

Cloutier and his colleagues are also investigating emerging patterns for systems and systems architecture that will enable the application of a high level of system pattern to the CIM view, and the decomposition of that pattern applied at the PIM view. References to that body of work are cited in the Cloutier paper [79].

5.2 Executable UML Foundation

Another area of active research is Executable UML; technically, Executable UML Foundation [80],[81]. At the time of this writing, this work is managed under the auspices of the OMG's Analysis and Design (A&D) Platform Task Force, an OMG Platform Technology Committee (TC) Work in Progress. This particular TC Work in Progress is chaired by Stephen J. Mellor and focused on managing a single, joint submission to the Executable UML Foundation RFP. The original RFP was issued on April 15, 2005 and formally entitled "Semantics of a Foundation Subset for Executable UML Models" RFP [80]. It is expected that the work of this TC will yield a public release of a second submission to the OMG for review at the end of June 2008 and a third submission to be formally voted on at the end of September 2008.[22]

Quoting directly from the Executable UML Foundation RFP, "*The objective of this RFP is to enable a chain of tools that support the construction, verification, translation, and execution of computationally complete executable models.*" The RFP objective statement goes on to state that proposals are solicited for the definition of a computationally complete and compact subset of UML 2.0 to be known as an "Executable UML Foundation," along with a full specification of the execution semantics of this subset. Here, "computationally complete" means that the subset shall be sufficiently expressive to allow definition of models that can be executed on a computer either through interpretation or as equivalent

[22] Nicolas Rouquette, Jet Propulsion Laboratory, California Institute of Technology (private communication), Apr. 18, 2008.

computer programs generated from models through some kind of transformations. And "compact subset" means that the selected metamodel should include as small a subset of the UML concepts as is practicable to achieve computational completeness.

Given the scope of the objectives for an Executable UML Foundation stated in the RFP, it not hard to see how Executable UML models could provide not only a solid foundation for the OMG MDA paradigm described earlier (see Section 5.1) but also a unified paradigm for precisely specifying structural and behavioral properties for requirements, verification, compliance, acceptance and simulation purposes, to name but a few.

Why does this body of work have relevance to the MBSE community? Because should a formal standard for Executable UML models emerge that is based on UML 2.0 semantics, it would be natural to extend this capability to OMG SysML, which, of course, builds on UML 2.0. This could result in a standards-based Executable SysML models as opposed to today's environment in which [UML and OMG SysML] executable models are tightly coupled to a particular vendor solution offering. A combined Executable UML/SysML specification could, for example, enable users to specify the minimum structural and behavioral requirements that an analysis model must fulfill in order to support a range of simulation techniques for domain-specific engineering analysis.

Progress on this emerging OMG specification and eventual standard for Executable UML Foundation will be monitored and reported in a future revision of this MBSE methodology survey report.

6. References

Additional information regarding the content of this report, including resources that describe the various candidate MBSE methodologies described herein, can be found in this section.

[1] Friedenthal, Sanford, Greigo, Regina, and Mark Sampson, INCOSE MBSE Roadmap, in "INCOSE Model Based Systems Engineering (MBSE) Workshop Outbrief" (Presentation Slides), presented at INCOSE International Workshop 2008, Albuquerque, NM, pg. 6, Jan. 26, 2008.

[2] Martin, James N., *Systems Engineering Guidebook: A Process for Developing Systems and Products*, CRC Press, Inc.: Boca Raton, FL, 1996.

[3] Jonah Z. Lavi and Joseph Kudish, "Systems Modeling & Requirements Specification Using ECSAM: An Analysis Method for Embedded & Computer-Based Systems," *Innovations in Systems and Software Engineering*, Vol. 1, No. 2, Springer: London, England, pp. 100-115, Sept. 2005.

[4] Jonah Z. Lavi and Joseph Kudish, *Systems Modeling and Requirements Specification Using ECSAM: An Analysis Method for Embedded and Computer-Based Systems*, Dorset House Publishing Company, Inc.:New York, NY, 2004.

[5] Boehm, Barry and Dan Port, "When Models Collide: Lessons from Software Systems Analysis," *IT Professional*, IEEE Computer Society, Institute of Electrical and Electronics Engineers, Vol. 1, Issue 1, Jan/Feb 1999.

[6] MBASE home page, Center for Systems and Software Engineering, University of Southern California, Los Angeles, CA, Aug. 13, 2003.

http://sunset.usc.edu/research/MBASE/index.html

[7] Bloomberg, Jason and Ronald Schmelzer, *Service Orient or Be Doomed!*, John Wiley & Sons: Hoboken, New Jersey, 2006.

[8] Royce, Winston W., "Managing the Development of Large Software Systems," *Proceedings of IEEE WESCON 26*, pp. 1-9, Aug. 1970.

[9] Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," *Computer*, pp. 61-72, May 1988.

[10] Forsberg, Kevin and Harold Mooz, "The Relationship of Systems Engineering to the Project Cycle," *Engineering Management Journal*, **4**, No. 3, pp. 36-43, 1992.

[11] Forsberg, Kevin and Harold Mooz, "Application of the "Vee" to Incremental and Evolutionary Development," *Proceedings of the Fifth Annual International Symposium of the National Council on Systems Engineering,* St. Louis, MO, July 1995.

[12] Department of Defense Directive (DoDD) Number 5000.1, "The Defense Acquisition System," Undersecretary of Defense for Acquisition and Technology, U.S. Department of Defense, May 12, 2003.

[13] Department of Defense Instruction (DoDI) Number 5000.2, "Operation of the Defense Acquisition System," U.S. Department of Defense, May 12, 2003.

[14] NASA Policy Directive (NPD) 7120.4C, "Program/Project Management," National Aeronautics and Space Administration, Washington, D.C., Dec. 6, 1999.

[15] NASA Procedural Requirement (NPR) 7120.5D, "NASA Space Flight Program and Project Management Requirements," National Aeronautics and Space Administration, Washington, D.C., Mar. 6, 2007.

[16] Balmelli, L., Brown, D., Cantor, M. and M. Mott, "Model-Driven Systems Development," *IBM Systems Journal*, **45**, No. 3, pp. 569-585, 2006.

[17] Roedler, Garry, "What is ISO/IEC 15288 and Why Should I Care?" (presentation slides), ISO/IEC JTC1/SC7/WG7, Geneva: International Organization for Standardization, Sept. 23, 2002.

[18] ANSI/EIA 632, *Processes for Engineering a System*, American National Standards Institute/Electronic Industries Alliance, 1999.

[19] IEEE Std 1220-1998, *IEEE Standard for Application and Management of the Systems Engineering Process*, Institute for Electrical and Electronic Engineers, Dec. 8, 1998.

[20] ISO/IEC 15288:2002, *Systems Engineering – System Life Cycle Processes*, International Organization for Standardization/International Electrotechnical Commission, Nov. 15, 2003.

[21] IEEE Std 15288™-2004, *Systems Engineering – System Life Cycle Processes*, Institute for Electrical and Electronic Engineers, June 8, 2005.

[22] Haskins, Cecilia (ed.), *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, *v. 3*, INCOSE-TP-2003-002-03, International Council on Systems Engineering, June 2006.

[23] NASA Procedural Requirement (NPR) 7123.1A, "NASA Systems Engineering Processes and Requirements," National Aeronautics and Space Administration, Washington, D.C., Mar. 26, 2007.

[24] Baker, Loyd, Clemente, Paul, Cohen, Bob, Permenter, Larry, Purves, Byron and Pete Salmon, "Foundational Concepts for Model Driven System Design," white paper, INCOSE Model Driven System Design Interest Group, International Council on Systems Engineering, Jul. 15, 2000.

[25] Wymore, A. Wayne, *A Mathematical Theory of Systems Engineering: The Elements*, John Wiley & Sons: New York, NY, 1967.

[26] Wymore, A. Wayne, *Model-Based Systems Engineering*, CRC Press, Inc.: Boca Raton, FL, 1993.

[27] Wymore, A. Wayne, "Contributions to the Mathematical Foundations of Systems Science and Systems Engineering," Systems Movement: Autobiographical Retrospectives, The University of Arizona, Tucson, AZ, 2004.

[28] Chapman, William L., Bayhill, A. Terry, and A. Wayne Wymore, *Engineering Modeling and Design*, CRC Press, Inc.: Boca Raton, FL, 1992.

[29] Wymore, A. Wayne, "Model-Based Systems Engineering: A Quick Overview" (Presentation Slides), tutorial presented at INCOSE International Workshop 2007, Albuquerque, NM, Jan. 25, 2007.

[30] Klir, George, "Review of *Model-Based Systems Engineering*," International Journal of General Systems, George Klir (Ed.), Vol 25, No. 2, Gordon and Breach: Amsterdam, Holland, pp 179-180, 1996.

[31] Bahill, A. Terry, Alford, Mark, Bharathan, K., Clymer, John R., Dean, Douglas L., Duke, Jeremy, Hill, Greg, LaBudde, Edward V., Taipale, Eric J., and A. Wayne Wymore, "The Design-Methods Comparison Project," *IEEE Transactions on Systems, Man, and Cybernetics,* Part C: Applications and Reviews, Vol. 28, No. 1, pp. 80-103, Feb. 1998.

[32] Douglass, Bruce P., "The Harmony Process," I-Logix white paper, I-Logix, Inc., Mar. 25, 2005.

[33] Hoffmann, Hans-Peter, "SysML-Based Systems Engineering Using a Model-Driven Development Approach," *Proceedings of INCOSE 2006 International Symposium*, Orlando, FL, Jul. 12, 2006.

[34] Hoffmann, Hans-Peter, "Harmony-SE/SysML Deskbook: Model-Based Systems Engineering with Rhapsody," Rev. 1.51, Telelogic/I-Logix white paper, Telelogic AB, May 24, 2006.

[35] Lykins, Howard, Friedenthal, Sanford and Abraham Meilich, "Adapting UML for an Object-Oriented Systems Engineering Method (OOSEM)," *Proceedings of the INCOSE 2000 International Symposium*, Minneapolis, MN, Jul. 2000.

[36] Friedenthal, Sanford, "Object Oriented Systems Engineering," *Process Integration for 2000 and Beyond: Systems Engineering and Software Symposium.* New Orleans, LA, Lockheed Martin Corporation, 1998.

[37] "Object Oriented System Engineering Method", OOSEM Descriptive Outline for INCOSE SE Handbook Version 3, Annotated Update, Sect. 6.4.2, pp. 6-1 to 6-6, Mar. 14, 2006.

[38] Friedenthal, Sanford, Moore, Alan, and Rick Steiner, *Practical Guide to SysML: Systems Modeling Language*, Morgan Kaufmann Publishers, Inc.: San Francisco, CA, 2008 (in press).

[39] "Object-Oriented Systems Engineering Method (OOSEM) Tutorial," Ver. 02.42.00, Lockheed Martin Corporation and INCOSE OOSEM Working Group, Apr. 2006.

[40] Kruchten, Philippe, *The Rational Unified Process: An Introduction*, *Third Edition*, Addison-Wesley Professional: Reading, MA, 2003.

[41] Cantor, Murray, "RUP SE: The Rational Unified Process for Systems Engineering," *The Rational Edge*, Rational Software, Nov. 2001.

[42] Cantor, Murray, "Rational Unified Process® for Systems Engineering, RUP SE® Version 2.0," IBM Rational Software white paper, IBM Corporation, May 8, 2003.

[43] Viljoen, Jaco, "RUP SE: The Rational Unified Process for Systems Engineering," INCOSE SA [South Africa] Newsletter, Issue 01: Q4, 2003.

[44] ISO/IEC 10746-1:1998(E), *Information technology – Open Distributed Processing – Reference model: Overview,* International Organization for Standardization/International Electrotechnical Commission, Nov. 15, 2003.

[45] ANSI/IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, American National Standards Institute/Institute for Electrical and Electronic Engineers, Sep. 21, 2000.

[46] Long, James E., "Systems Engineering (SE) 101," CORE®: Product & Process Engineering Solutions, Vitech training materials, Vitech Corporation, Vienna, VA, 2000.

[47] "Vitech Announces Participation in INCOSE's 17th Annual International Symposium, Delivering Five Key Systems Engineering Presentations, Papers and Panels," Vitech News Release, Vitech Corporation, Vienna, VA, Feb. 23, 2007.

[48] Long, James E., "MBSE in Practice: Developing Systems with CORE," Vitech briefing slides, Vitech Corporation, Vienna, VA, Mar. 2007.

[49] Baker, Loyd Jr. and James E. Long, "Role of System Engineering Across The System Life Cycle," Vitech white paper, Vitech Corporation, Vienna, VA, Jul. 15, 2000.

[50] Childers, Susan Rose and James E. Long, "A Concurrent Methodology for the System Engineering Design Process," Vitech white paper, Vitech Corporation, Vienna, VA, May 20, 2004.

[51] Long, Jim, "Relationships between Common Graphical Representations in Systems Engineering," Vitech white paper, Vitech Corporation, Vienna, VA, Aug. 6, 2002.

[52] Skipper, J. F., "A Systems Engineer's Position on the Unified Modeling Language," Vitech white paper, Vitech Corporation, Vienna, VA, Jul. 15, 2003.

[53] Ingham, Michel D., Rasmussen, Robert D., Bennett, Matthew B. and Alex C. Moncada, "Generating Requirements for Complex Embedded Systems Using State Analysis," *Acta Astronautica*, **58**, Iss. 12, pp. 648-661, Jun. 2006.

[54] Dvorak, Dan, Rasmussen, Robert, Reeves, Glenn and Allan Sacks, "Software Architecture Themes in JPL's Mission Data System," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, paper AIAA-99-4553, 1999.

[55] Bennett, Matthew B., Rasmussen, Robert D. And Michel D. Ingham, "A Model-Based Requirements Database Tool for Complex Embedded Systems," *Proceedings of the INCOSE 2005 International Symposium*, Rochester, NY, Jul. 2005.

[56] Rasmussen, Bob, "Session 1: Overview of State Analysis," (internal document), State Analysis Lite Course, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 2005.

[57] Ingham, Mitch, "State Analysis Overview: What PSEs ought to know," Briefing Slides (internal document), Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Nov. 2, 2006.

[58] Kordon, Mark, Wall, Steve, Stone, Henry, Blume, William, Skipper, Joseph, Ingham, Mitch, Neelon, Joe, Chase, James, Baalke, Ron, Hanks, David, Salcedo, Jose, Solish, Benjamin, Postma, Mona, and Richard Machuzak, "Model-Based Engineering Design Pilots at JPL," *2007 IEEE Aerospace Conference Proceedings*, 1-4244-0524-4/07, IEEEAC paper #1678, Institute of Electrical and Electronic Engineers, 2005.

[59] Weiss, Kathryn A., "Engineering Spacecraft Mission Software using a Model-Based and Safety-Driven Design Methodology," *Journal of Aerospace Computing, Information, and Communication*, **3**, pp. 562-586, Nov. 2006.

[60] Leveson, Nancy G., *Safeware: System Safety and Computers*, Addison-Wesley: Reading, MA, pp. 171-184 and 313-358, 1995.

[61] Leveson, Nancy G., "A New Approach to Hazard Analysis for Complex Systems," *International Conference of the System Safety Society*, Ottawa, Canada, Aug. 2003.

[62] NASA Software Safety Standard, NASA Technical Standard, NASA-STD-8719.13B w/Change 1, National Aeronautics and Space Administration, Washington, D.D., Jul. 8, 2004.

[63] Leveson, Nancy G., "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Transactions on Software Engineering*, **26**, No. 1, Jan. 2000.

[64] Weiss, Kathryn A., Ong, Elwin C. and Nancy G. Leveson, "Reusable Specification Components for Model-Driven Development," *Proceedings of the INCOSE 2003 International Symposium*, Crystal City, VA, Jul. 2003.

[65] Piccorillo, Sallie (Ed.), INCOSE Los Angeles Chapter Newsletter, Vol. 6, Issue No. 2, March 2008.

[66] Dori, Dov, *Object-Process Methodology: A Holistic Systems Paradigm*, Springer-Verlag: Berlin Heidelberg, Germany, 2002.

[67] Dori, Dov and Iris Reinhartz-Berger, "An OPM-Based System Development Process," I.-Y. Song et al. (Eds.): ER 2003, LNCS 2813, Springer-Verlag: Berlin Heidelberg, Germany, pp. 105-117, 2003.

[68] Reinhartz-Berger, Iris and Dov Dori, "A Reflective Metamodel of Object-Process Methodology: The System Modeling Building Blocks," *Business Systems Analysis with Ontologies*, P. Green and M. Rosemann (Eds.), Idea Group: Hershey, PA, USA, pp. 130-173, 2005.

[69] Dori, Dov, Reinhartz-Berger, Iris, and Arnon Sturm, "OPCAT – A Bimodal Case Tool for Object-Process Based System Development," 5th International Conference on Enterprise Information Systems (ICEIS 2003), pp. 286-291, 2003.

[70] Dori, Dov and Mordechai Choder, "Conceptual Modeling in Systems Biology Fosters Empirical Findings: The mNRA Lifecycle," Proceedings of the Library of Science ONE (PLoS ONE), Sep. 12, 2007.

[71] Object Management Group, *Unified Modeling Language: Superstructure*, Ver. 2.1.1, OMG Adopted Specification, OMG document formal/2007-02-05, Needham, MA, Feb. 2007.

[72] Object Management Group, *OMG SysML Specification*, OMG Adopted Specification, OMG document ptc/06-05-04, Needham, MA, May 2006.

[73] Bock, Conrad, "UML 2 for Systems Engineering," Briefing Slides, National Institute of Standards and Technology, Mar. 27, 2003.

[74] Bock, Conrad, "SysML and UML 2 Support for Activity Modeling," *Systems Engineering*, **9**, No. 2, pp. 160-186, 2006.

[75] Object Management Group, *MDA Guide Version 1.0.1*, OMG document omg/2003-06-01, Needham, MA, Jun. 12, 2003.

[76] Kleppe, Anneke, Warmer, Jos, and Wim Bast, *MDA Explained: The Model-Driven Architecture: Practice and Promise*, The Addison-Wesley Object Technology Series, Addison-Wesley Professional: Boston, MA, 2003.

[77] Mellor, Stephen J., Scott, Kendall, Uhl, Axel, and Dirk Weise, *MDA Distilled*, The Addison-Wesley Object Technology Series, Addison-Wesley Professional: Boston, MA, 2004.

[78] Cloutier, Robert, "Model Driven Architecture for Systems Engineering" (Presentation Slides), Stevens Institute of Technology, presented at INCOSE International Workshop 2008, Albuquerque, NM, Jan. 25, 2008.

[79] Cloutier, Robert, "Model Driven Architecture for Systems Engineering," Paper #220, *Proceedings of the Conference on Systems Engineering Research (CSER)*, CSER 2008, University of Southern California, Los Angeles, CA, Apr 4-5, 2008.

[80] Object Management Group, *Semantics of a Foundational Subset for Executable UML Models*, Initial Submission, OMG document ad/06-05-02, Needham, MA, May 27, 2006.

[81] Mellor, Stephen J. and Marc J. Balcer, *Executable UML: A Foundation for Model-Driven Architecture*, The Addison-Wesley Object Technology Series, Addison-Wesley Professional: Boston, MA, 2002.

7. Acronyms and Abbreviations

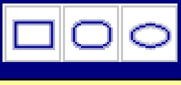
ANSI	American National Standards Institute
CAE	Computer Aided Engineering
CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-The-Shelf
DoD	Department of Defense
DoDD	Department of Defense Directive
DoDI	Department of Defense Instruction
EFFBD	Extended Function Flow Block Diagram
EIA	Electronic Industries Alliance
FDD	Functional Design Description
FDIS	Final Draft International Standard
FMECA	Failure Modes and Effects Criticality Analysis
FOC	Full Operational Capability
FTA	Fault Tree Analysis
IEC	International Electrotechnical Commission
IEEE	Institute for Electrical and Electronic Engineers
INCOSE	International Council on Systems Engineering
IOC	Initial Operational Capability
IPD	Integrated Product Development
IPPD	Integrated Product and Process Development

ISO	International Organization for Standardization
ITU	International Telecommunication Union
JPL	Jet Propulsion Laboratory
KSA	Knowledge, Skills, and Abilities
MBED	Model-Based Engineering Design
MBSE	Model-Based Systems Engineering
MDS	Mission Data System
MDSD	Model-Driven Systems Development
NASA	National Aeronautics and Space Administration
NPD	NASA Policy Directive
NPR	NASA Procedural Requirement
OMG	Object Management Group
OOSEM	Object-Oriented Systems Engineering Method
OPM	Object-Process Methodology
PMTE	Process, Methods, Tools, and Environment
RMC	Rational Method Composer
RUP	Rational Unified Process
RUP SE	Rational Unified Process for Systems Engineering
SA	State Analysis
SE	Systems Engineering
SEMP	Systems Engineering Management Plan
STAMP	Systems Theoretic Accident Modeling and Process
STPA	STAMP-based Hazard Analysis
SysML	Systems Modeling Language
UML	Unified Modeling Language
V&V	Verification and Validation
WBS	Work Breakdown Structure

Appendix

Main Object-Process Methodology (OPM) concepts, their symbols, and their meaning (adapted from [70]).

Table A-1. ENTITIES



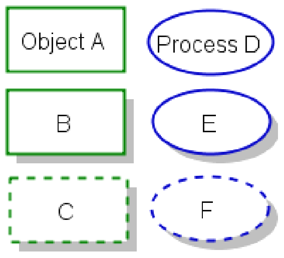
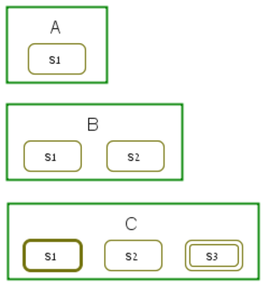
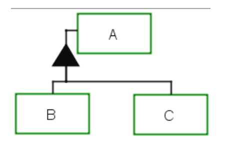
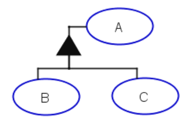
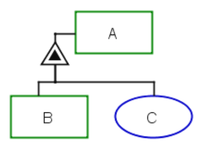
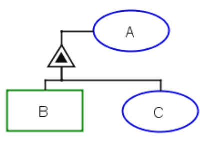
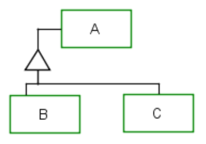
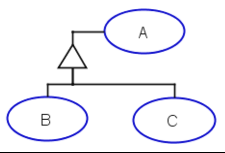
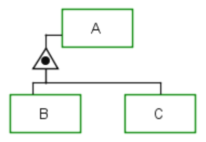
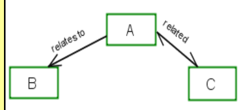
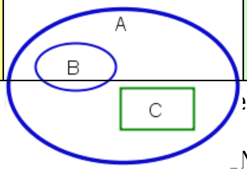
Name		Symbol	OPL	Definition
Things	**Object** / **Process**		**B** is physical. (shaded rectangle) **C** is physical and environmental. (shaded dashed rectangle) **E** is physical. (shaded ellipse) **F** is physical and environmental. (shaded dashed ellipse)	An **object** is a thing that exists. A **process** is a thing that transforms at least one object. Transformation is object generation or consumption, or effect—a change in the state of an object.
State			**A** is **s1**. **B** can be **s1** or **s2**. **C** can be **s1**, **s2**, or **s3**. **s1** is initial. **s3** is final.	A **state** is situation an object can be at or a value it can assume. States are always within an object. States can be initial or final.

Table A-2. STRUCTURAL LINKS AND COMPLEXITY MANAGEMENT

Name		Symbol	OPL	Semantics
Fundamental Structural Relations	Aggregation-Participation		**A** consists of **B** and **C**.	A is the whole, B and C are parts.
			**A** consists of **B** and **C**.	
	Exhibition-Characterization		**A** exhibits **B**, as well as **C**.	Object B is an attribute of A and process C is its operation (method). A can be an object or a process.
			**A** exhibits **B**, as well as **C**.	
	Generalization - Specialization		**B** is an **A**. **C** is an **A**.	A specializes into B and C. A, B, and C can be either all objects or all processes.
			**B** is **A**. **C** is **A**.	
	Classification-Instantiation		**B** is an instance of **A**. **C** is an instance of **A**.	Object A is the class, for which B and C are instances. Applicable to processes too.
Unidirectional & bidirectional tagged structural links			**A** relates to **B**. (for unidirectional) **A** and **C** are related. (for bidirectional)	A user-defined textual tag describes any structural relation between two objects or between two processes.
In-zooming			**A** exhibits **C**. **A** consists of **B**. **A** zooms into **B**, as well as	Zooming into process A, B is its part and C is its attribute.

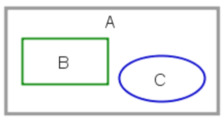
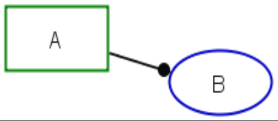
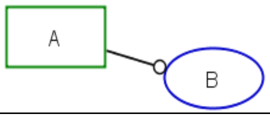
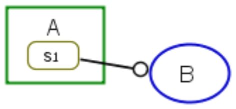
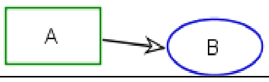
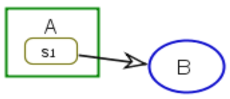
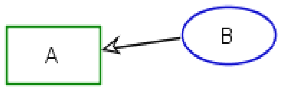
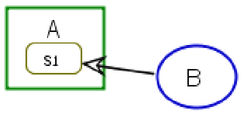
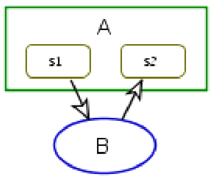
		C.	
		**A** exhibits **C**. **A** consists of **B**. **A** zooms into **B**, as well as **C**.	Zooming into object A, B is its part and C is its operation.

Table A-3. ENABLING AND TRANSFORMING PROCEDURAL LINKS

	Name	Symbol	OPL	Semantics
Enabling links	Agent Link		**A** handles **B**.	Denotes that the object is a human operator.
	Instrument Link		**B** requires **A**.	"Wait until" semantics: Process B cannot happen if object A does not exist.
	State-Specified Instrument Link		**B** requires **s1 A**.	"Wait until" semantics: Process B cannot happen if object A is not at state s1.
Transforming links	Consumption Link		**B** consumes **A**.	Process B consumes Object A.
	State-Specified Consumption Link		**B** consumes **s1 A**.	Process B consumes Object A when it is at State s1.
	Result Link		**B** yields **A**.	Process B creates Object A.
	State-Specified Result Link		**B** yields **s1 A**.	Process B creates Object A at State s1.
	Input-Output Link Pair		**B** changes **A** from **s1** to **s2**.	Process B changes the state of Object A from State s1 to State s2.

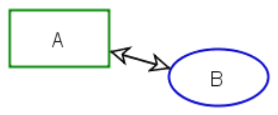
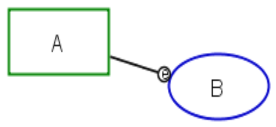
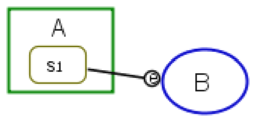
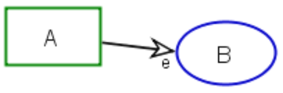
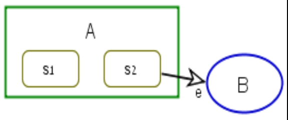
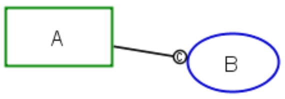
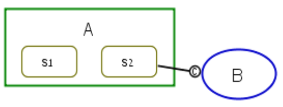
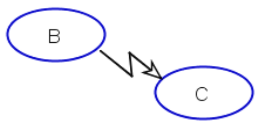
	Effect Link		**B** affects **A**.	Process B changes the state of Object A; the details of the effect may be added at a lower level.

Table A-4. EVENT, CONDITION, AND INVOCATION PROCEDURAL LINKS

Name	Symbol	OPL	Semantics
Instrument Event Link		**A** triggers **B**. **B** triggers **A**.	Existence or generation of object A will attempt to trigger process B once. Execution will proceed if the triggering failed.
State-Specified Instrument Event Link		A triggers **B**. when it enters **s1**. **B** requires **s1** **A**.	Entering state s1 will attempt to trigger the process once. Execution will proceed if the triggering failed.
Consumption Event Link		**A** triggers **B**. **B** consumes **A**.	Existence or generation of object A will attempt to trigger process B once. If B is triggered, it will consume A. Execution will proceed if the triggering failed.
State-Specified Consumption Event Link		**A** triggers **B** when it enters **s2**. **B** consumes **s2** **A**.	Entering state s2 will attempt to trigger the process once. If B is triggered, it will consume A. Execution will proceed if the triggering failed.
Condition Link		**B** occurs if **A** exists.	Existence of object A is a condition to the execution of B. If object A does not exist, then process B is skipped and regular system flow continues.
State-Specified Condition Link		**B** occurs if **A** is **s1**.	Existence of object A at state s2 is a condition to the execution of B. If object A does not exist, then process B is skipped and regular system flow continues.
Invocation Link		**B** invokes **C**.	Execution will proceed if the triggering failed (due to failure to fulfill one or more of the conditions in the precondition set).

Acknowledgments

The author would like to thank Joseph Skipper, Mark Kordon, Ross Jones, Kenny Meyer, Michel Ingham, Kathryn Weiss, and Nicolas Rouquette all of JPL for their valuable feedback and review of this survey report. In addition, the author would like thank Hans-Peter Hoffmann of IBM Telelogic, Sanford ("Sandy") Friedenthal of Lockheed Martin Corporation, Murray Cantor of IBM Rational, James ("Jim") Long of Vitech Corporation, and Prof. Dov Dori of Technion, Israel Institute of Technology for their valuable feedback of the Harmony-SE, OOSEM, RUP-SE, Vitech MBSE, and OPM methodologies, respectively.

The work carried out in this report was carried out at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

Document Change Record

Date	Revision	Description	Author(s)
May 23, 2007	A	Initial revision.	Jeff A. Estefan
May 23, 2008	B	1. Updated scope section. 2. Added references to ECSAM and MBASE methodologies to scope section. 3. Added new section dedicated to Wymore work. 4. Added new section describing Dori OPM methodology. 5. Added new section on role of OMG MDA and Executable UML Foundation. 6. Minor editorial updates.	Jeff A. Estefan