

Escaping the Software Tar Pit: Model Clashes and How to Avoid Them

Barry Boehm, Dan Port, USC

1. Introduction

“No scene from prehistory is quite so vivid as that of the mortal struggles of great beasts in the tar pits... Large system programming has over the past decade been such a tar pit, and many great and powerful beasts have thrashed violently in it...

“Everyone seems to have been surprised by the stickiness of the problem, and it is hard to discern the nature of it. But we must try to understand it if we are to solve it.”

Fred Brooks, 1975



Several recent books and reports have confirmed that the software tar pit is at least as hazardous today as it was in 1975. Our research into several classes of models used to guide software development (product models, process models, property models, success models), has convinced us that the concept of model clashes among these classes of models helps explain much of the stickiness of the software tar-pit problem.

We have been developing and experimentally evolving an approach called MBASE -- Model-Based (System) Architecting and Software Engineering -- which helps identify and avoid software model clashes. Section 2 of this paper introduces the concept of model clashes, and provides examples of common clashes for each combination of product, process, property, and success model. Sections 3 and 4 introduce the MBASE approach for endowing a software project with a mutually supportive set of models, and illustrate the application of MBASE to an example corporate resource scheduling system. Section 5 summarizes the results of applying the MBASE approach to a family of small digital library projects. Section 6 presents conclusions to date.

2. Model Clashes and the Need to Avoid Them

2.1 Model Types and Sources

When we participate in the definition of a new or revised software-intensive system, we employ various models to help us visualize or reason about the prospective system and its likely effects. Here, we use simplified versions of two of Webster's definitions of "model:" (1) a pattern of something to be made; (2) a description or analogy used to help visualize something. Analysis is included as a form of visualization.

Some of these models deal with the product or system we are defining. Others deal with the process of developing and evolving the system; with properties of the system such as cost, performance, or dependability; or with what it means for the system to be successful (mission performance, return on investment, stakeholder win-win).

The models we use come from various sources. Some models may be imposed by laws or regulations, such as Government acquisition processes. Some models may be imposed by business conventions, such as return on investment. Some models may come from hard-won experience: the IKIWISI (I'll know it when I see it) success model emerged after many experiences confirmed that the use of prototypes generally led to better user interfaces than relying just on written requirement specifications (Note, though, that models based on outdated experience can become sources of problems). Some models may be buried in our early education, such as the Golden Rule success model (Do unto others as you would have others do unto you).

2.2 Nature of Model Clashes

Models such as the above are very powerful and deep-rooted. When you follow them, you generally feel that you must be doing everything right. When they conflict with each other to produce a troubled software project, it is hard for the project participants to understand what is going wrong. Frequently, they try surface remedies (baseline the requirements, fire the manager, add more quality assurance people, buy some tools, impose more standards), and are surprised when things don't get better – or frequently get worse.

What is happening deep below is a model clash: a conflict among the underlying assumptions of the models the project has unsuspectingly adopted. This model clash is usually not affected by the surface remedies. It surfaces in ways which cause combinations of confusion, mistrust, frustration, rework, and throwaway systems. It leaves the participants feeling that making progress on this software project is much like slogging through a tar pit.

2.3 An Example: The Golden Rule

For example, what could be more right than the Golden Rule? Yet when applied to software, this success model has been the cause of a tremendous amount of confusion, mistrust, frustration, rework, and throwaway software.

For software-intensive systems, the system developers are often computer scientists, who invoke the Golden Rule to say [Boehm, 1981, p.720]:

Do unto others	Develop a computer system to serve users and operators,
as you would have others do unto you	assuming that the users and operators like to write computer programs, and know a good deal about computer science.

Such systems are likely to have user interfaces with terse, powerful, but often unforgiving and obscure command languages. They are likely to put the user in direct contact with a powerful, but often unforgiving and obscure operating system or network management system. If such user interfaces are developed for programmers, they will love them. If they are developed for doctors, financial analysts, or real estate agents, they will “know it when they see it” that the system is not for them.

The usual project result is a late discovery that the system is not user-supportive or user-friendly. This causes either a throwaway system or a lot of confusion, mistrust, frustration, and rework trying to fix the system – after its original budget and schedule have already been spent.

The main cause is the underlying assumption of the Golden Rule: “Everyone is like me.” This causes a conflict with the underlying assumption of the stakeholder win-win success model that the users are implicitly expecting: “My win conditions may be different from yours. When these conflict, we need to resolve the conflicts before going forward.”

How can we provide guidelines for avoiding such conflicts in the future? It is best to try to retain the strengths of such deeply-held principles as the Golden Rule, rather than trying to replace them. This suggests the following Modified Golden Rule: “Do unto others as you would have others do unto you – if you were like them.”

2.4 A Taxonomy of Model Clashes

Table 1 shows that serious model clashes may arise from any combination of product, process, property, and success models adopted by a project. The Golden Rule vs. stakeholder win-win conflict above is an example of a model clash among software success models.

Table 1. Examples of Model Clashes

	Product Model	Process Model	Property Model	Success Model
Product Model	<ul style="list-style-type: none"> • Structure clash • Traceability clash • Architecture style clash 	<ul style="list-style-type: none"> • COTS-driven product vs. Waterfall (requirements-driven) process 	<ul style="list-style-type: none"> • Interdependent multiprocessor product vs. linear performance scalability model 	<ul style="list-style-type: none"> • 4GL-based product vs. low development cost and performance scalability
Process Model		<ul style="list-style-type: none"> • Multi-increment development process vs. single-increment support tools 	<ul style="list-style-type: none"> • Evolutionary development process vs. Rayleigh-curve cost model 	<ul style="list-style-type: none"> • Waterfall process model vs. “I’ll know it when I see it” (IKIWISI) prototyping success model
Property Model			<ul style="list-style-type: none"> • Minimize cost and schedule vs. maximize quality (“Quality is free”) 	<ul style="list-style-type: none"> • Fixed-price contract vs. easy-to-change, volatile requirements
Success Model				<ul style="list-style-type: none"> • Golden Rule vs. stakeholder win-win

Model clashes among software product models have been the ones most frequently addressed to date. Examples are structure clashes between a project’s input and output structures [Jackson, 1975]; traceability clashes among a product’s requirements, design and code [Rosove, 1967]; and architectural style clashes in integrating commercial off-the-shelf (COTS) or other reusable software components [Garlan et al., 1995].

Garlan's group tried to integrate four COTS products into their Aesop system: the OBST object management system, the Mach RPC Interface Generator, the SoftBench tool integration framework, and the InterViews user interface manager. They found a number of architectural style clashes among the underlying assumptions of the products. For example, three of the four products were event-based, but each had different event semantics, and each assumed it was the sole owner of the event queue. Resolving these model conflicts escalated an original two-person, six-month project into a five-person, two-year project: a factor of four in schedule and a factor of five in effort.

There is also an increasing appreciation of the need to avoid clashes between a system's product model and its development process model. A good example is the clash between a COTS-driven product model, in which the available COTS capabilities largely determine the system's "requirements," and the use of a waterfall process model, in which a set of predetermined requirements are supposed to determine the system's capabilities.

This model clash is most serious when the waterfall-model requirements are made the basis of a legal contract between the customer and developer organizations. In a recent example, the contract performance requirements specified a response time of less than one second. When it was subsequently determined that no commercial database systems could not meet this requirement, it was not the technical people who led the effort to resolve the issue. It was the lawyers. In the end, everyone agreed that it was in no one's best interest to insist on the one-second requirement – but only after months of confusion, frustration, and mistrust.

Continuing along the top row of Table 1, an example of a model clash between a product model and a property model is to use a multiprocessor product model and a throughput property model which assumes that system throughput scales linearly with the number of processors. For most multi-processor applications, data dependencies, control dependencies, or resource contention problems will cause the product strategy, "If we run out of throughput, just add another processor," to actually yield a decrease in throughput beyond a certain number of processors.

This strategy was employed on some early satellite control center projects. When the limited scalability of the architecture was discovered late in the development process, a tremendous amount of rework was necessary to re-architect and rebuild the system.

The classic New Jersey Department of Motor Vehicles system failure [Babcock, 1985] was a good example of a clash between product model (build the product using a fourth generation language) and a success model involving two properties (low development cost and good throughput). The 4GL system was developed at low cost, but its throughput was so poor that at one point over a million New Jersey automobiles were driving around with unprocessed license renewals--again, yielding a lot of confusion, frustration and mistrust.

2.5 Model Clash Patterns

We have been working on a set of model clash patterns to help people to recognize the most common model clashes, to appreciate and communicate the problems they will cause, and to determine what to do about them.

Table 2 shows two example patterns from the discussions in Sections 2.3 and 2.4, and a further example from Table 1: the conflict between the waterfall process model and the IKIWISI (I'll know it when I see it) success model.

A key underlying assumption of the sequential waterfall model is that the requirements are knowable in advance of the system's implementation. This is frequently not the case, as the new system's automated capabilities will impact existing operations in ways that are hard for users to articulate or for analysts to work out via abstract task models.

In the example shown in Table 2, a large commercial organization found that the most convenient source of detailed requirements came from their existing operational procedures. This "automating the existing system" approach caused so much unnecessary user input workload and such obtuse user outputs that the system had to be scrapped after about three years and \$20 million worth of effort.

Note that the two model clashes in Table 2 involving the waterfall model involve similar recommended solutions: concurrently evolving the requirements and other key system artifacts such as COTS choices, architectural choices, concepts of operation, and prototypes. In Section 3, we will show how the MBASE approach enables people to do this.

Table 2. Example Model Clash Patterns

Model Types	Models	Underlying Assumptions	Likely Problems: Examples	Problem Avoidance
Success	Golden Rule	Everybody is like me	Programmer-friendly systems unfriendly to non-programmers: Numerous medical systems	Modified Golden Rule: Do unto others as you would have others do unto you -- if you were like them
Success	Stakeholder Win-Win	People have different win conditions		
Process	Waterfall	The requirements and resources are fixed in advance. They have no high-risk implications.	Requirements are found to exclude COTS and/or to be expensive or impossible to achieve. Lawyers get together to fix things. Numerous government contracts.	Concurrently evolve requirements and risk management of COTS-based architecture.
Product	COTS-driven	The requirements are driven by the available COTS capabilities.		
Process	Waterfall	The requirements are knowable in advance of implementation.	Using existing operations as a convenient source of requirements. Major user overloads and task mismatches with new automated system. Numerous commercial systems.	Concurrently evolve requirements, prototypes, scenarios, task models, concepts of operation.
Success	IKIWISI	Users are best able to articulate requirements via interaction with working examples.		

Model Assumptions

However, note also that the two waterfall model underlying assumptions shown in Table 2 are different. In general, models will have several underlying assumptions which may conflict with those of other models. As another help in avoiding model clashes, we are working on identifying the assumptions a project will inherit when it adopts a given model. Table 3 shows example lists of key underlying assumptions for the waterfall and evolutionary development process models.

Table 3. Examples of Key Underlying Assumptions

<p>Waterfall Model Assumptions</p> <ol style="list-style-type: none">1. The requirements are knowable in advance of implementation.2. The requirements have no unresolved, high-risk implications (e.g., risks due to COTS choices, cost, schedule, performance, safety, security, user interfaces, organizational impacts).3. The nature of the requirements will not change very much (during development; during evolution).4. The requirements are compatible with all the key system stakeholders' expectations (e.g., users, customers, developers, maintainers, investors).5. The right architecture for implementing the requirements is well understood.6. There is enough calendar time to proceed sequentially.
<p>Evolutionary Development Assumptions</p> <ol style="list-style-type: none">1. The initial release is sufficiently satisfactory to key system stakeholders that they will continue to participate in its evolution.2. The architecture of the initial release is scalable to accommodate the full set of system life cycle requirements (e.g., performance, safety, security, distribution, localization).3. The operational user organizations are sufficiently flexible to adapt to the pace of system evolution.4. The dimensions of system evolution are compatible with the dimensions of evolving-out the legacy systems it is replacing.

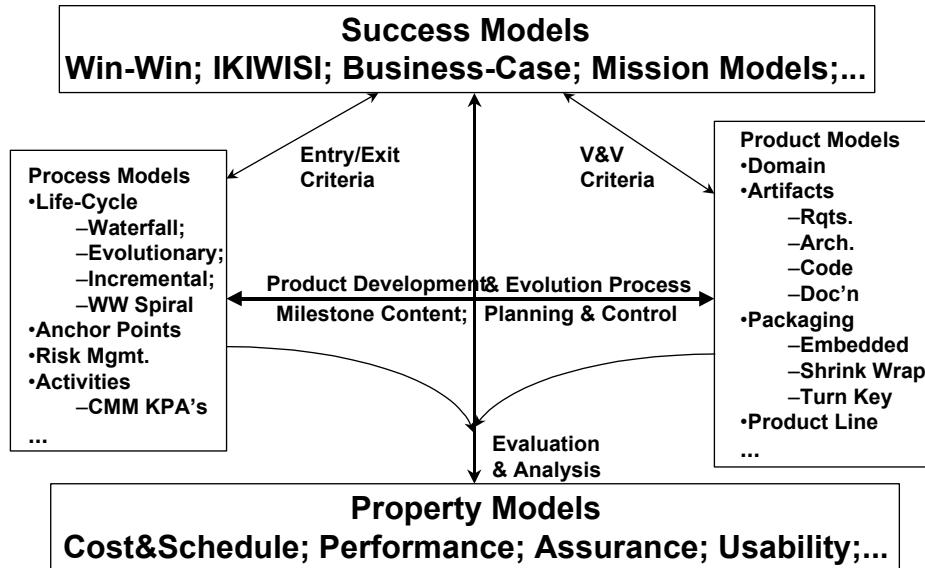
3. MBASE Overview

3.1 MBASE Model Integration Framework

Figure 1 summarizes the overall model integration framework used in the MBASE approach to ensure that a project's success, product, process and property models are consistent and well integrated. At the top of Figure 1 are success models, illustrated with several examples, whose priorities and consistency should be considered first.

Thus, if the overriding top-priority success model is to “Demonstrate a competitive agent-based electronic commerce system on the floor of COMDEX in 9 months,” this constrains the ambition level of other success models (e.g., one may not insist on provably correct code or a fully documented system). The 9-month schedule constraint is most critical because the system will lose most of its value if it is not available to compete for early market share at COMDEX.

Figure 1. MBASE Model Integration Framework



This schedule also determines many aspects of the product model (architected to easily shed lower-priority features if necessary to meet schedule), the process model (design-to-schedule), and various property models (only portable and reliable enough to achieve a successful demonstration). The achievability of the success model needs to be verified with respect to the other models. And conversely, the choices of process and product models need to be validated by having the success model provide verification and validation (V&V) criteria for the product milestone artifacts; preconditions and postconditions for the process milestones.

In the 9-month demonstration example, a cost-schedule estimation model would relate various product characteristics (sizing of components, reuse, product complexity), process characteristics (staff capabilities and experience, tool support, process maturity), and property characteristics (required reliability, cost constraints) to determine whether the product capabilities achievable in 9 months would be sufficiently competitive for the success models. Thus, as shown at the bottom of Figure 1, a cost and schedule property model would be used for the evaluation and analysis of the consistency of the system's product, process, and success models.

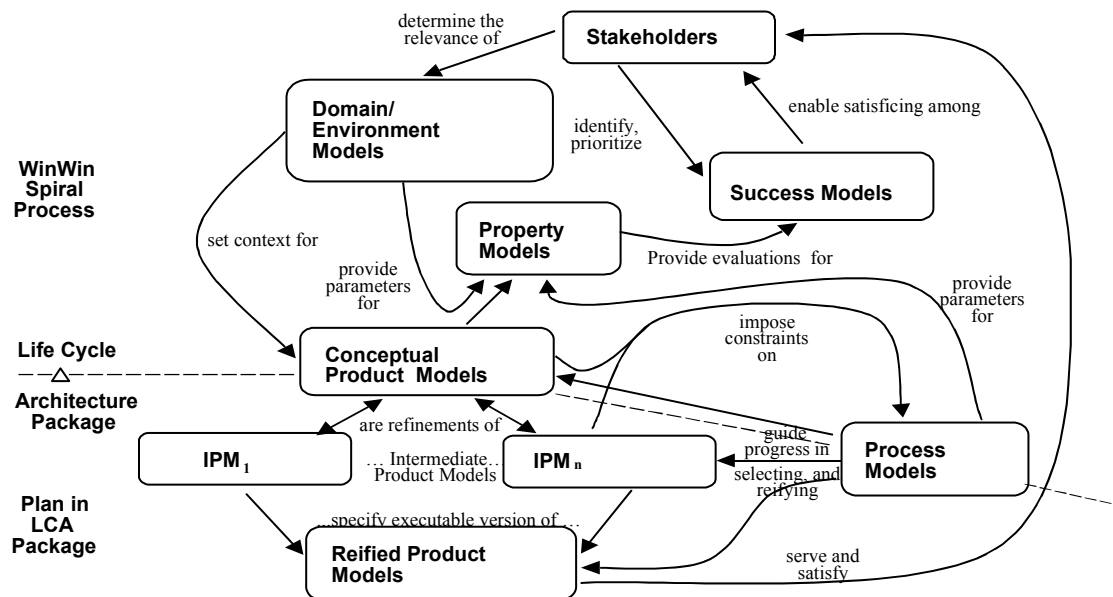
In other cases, the success model would make a process model or a product model the primary driver for model integration. An IKIWISI (I'll know it when I see it) success model would initially establish a prototyping and evolutionary development process model, with most of the product features and property levels left to be determined by the process. A success model focused on developing a product line of similar products would initially focus on product models (domain models, product line architectures), with process models and property models subsequently explored to perform a business-case analysis of the most appropriate breadth of the product line and the timing for introducing individual products.

3.2 MBASE Process Framework

Figure 2 provides an overall process framework for the MBASE approach. The primary drivers for any system's (or product line's) characteristics are its key stakeholders. These generally include the system (taken below to mean "system or product-line") users, customers, developers, and maintainers. Key stakeholders can also include strategic partners, marketers, operators of closely coupled systems, and the general public for such issues as safety, security, privacy, or fairness.

The critical interests of these stakeholders determine the priorities, desired levels, and acceptable levels of various system success criteria. These are reflected in the success models for the system, such as stakeholder win-win, business case, operational effectiveness models, or IKIWISI. These in turn determine which portions of an applications domain and its environment are relevant to consider in specifying the system

Figure 2. MBASE Process Framework



and its development and evolution process. The particular objective is to determine the system boundary, within which the system is to be developed and evolved; and outside of which is the system environment.

For example, in our 9-month COMDEX electronic commerce demo application, the system boundary would be determined by the most cost-effective set of capabilities that could be developed in 9 months. Thus, a credit card verification capability might be considered outside the initial system boundary, although it would be needed later.

The domain scope for the demo system would be very much determined by the available COTS products which could be tailored, integrated, and built upon. Determining the appropriate combination of COTS products and extensions could take several iterative cycles of experimental prototyping and risk resolution, in concert with cost-schedule modeling to determine how much capability would be feasible to develop in 9 months.

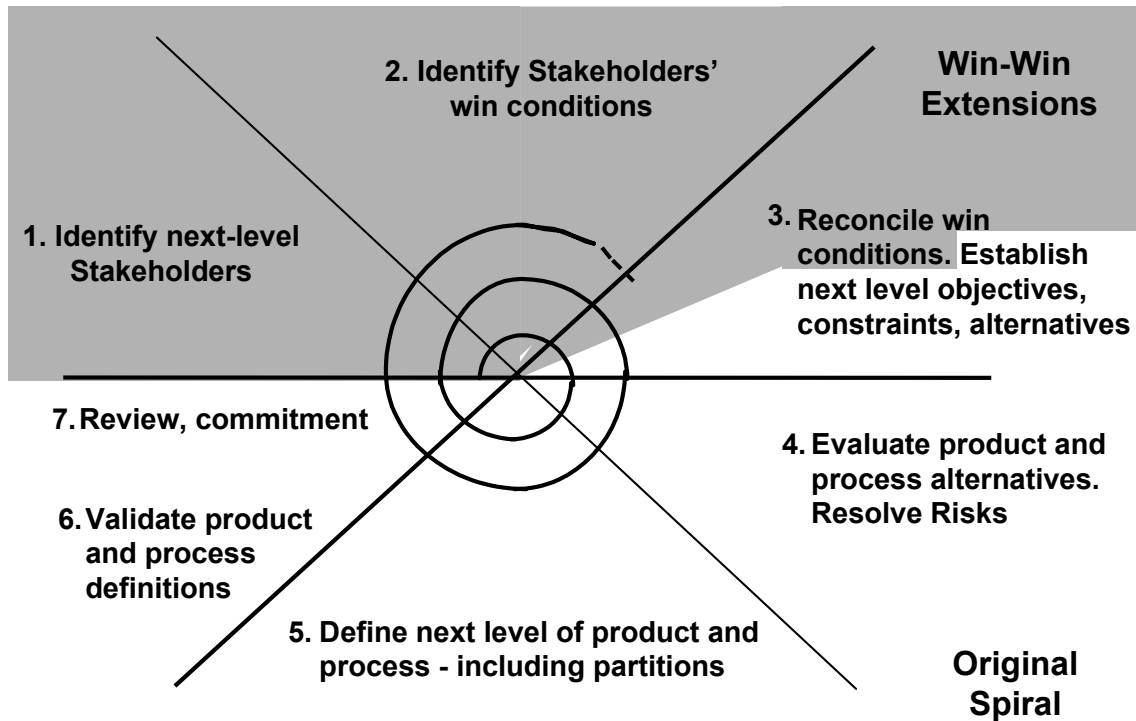
3.3 The Win-Win Spiral Process

These iterative cycles would follow the structure shown in the upper part of Figure 2. As indicated there, the cycles can be scoped and executed via the Win-Win Spiral Process. This process enhances the original risk-driven spiral model [Boehm, 1988] in two main ways.

- Using a stakeholder win-win approach to determine the objectives, constraints, and alternatives for each cycle [Boehm-Bose, 1994]; see Figure 3.
- Using a set of life cycle anchor points [Boehm, 1996] as critical management decision points.

These anchor points are described further in Section 3.4. As shown in Figure 2, one of them is the Life Cycle Architecture milestone. It includes a product definition, a process definition, and a feasibility rationale ensuring the compatibility of the systems product, process, property, and success models. From this base, the project can continue to construct the system by refining the product models into an executing product. This process will be elaborated in Section 4.

Figure 3. The Win-Win Spiral Model



3.4 Anchor Point Milestones

The specific content of the first two anchor point milestones are summarized in Table 4. They include increasingly detailed, risk-driven definitions of the system's operational concept, prototypes, requirements, architectures, life cycle plan, and feasibility rationale. For the feasibility rationale, property models are invoked to help verify that the project's success models, product models, process models, and property levels or models are acceptably consistent.

The first milestone is the Life Cycle Objectives (LCO) milestone, at which management verifies the basis for a business commitment to proceed at least through an architecting stage. This involves verifying that there is at least one system architecture and choice of COTS/reuse components which is shown to be feasible to implement within budget and schedule constraints, to satisfy key stakeholder win conditions, and to generate a viable investment business case.

Table 4. LCO and LCA Anchor Points

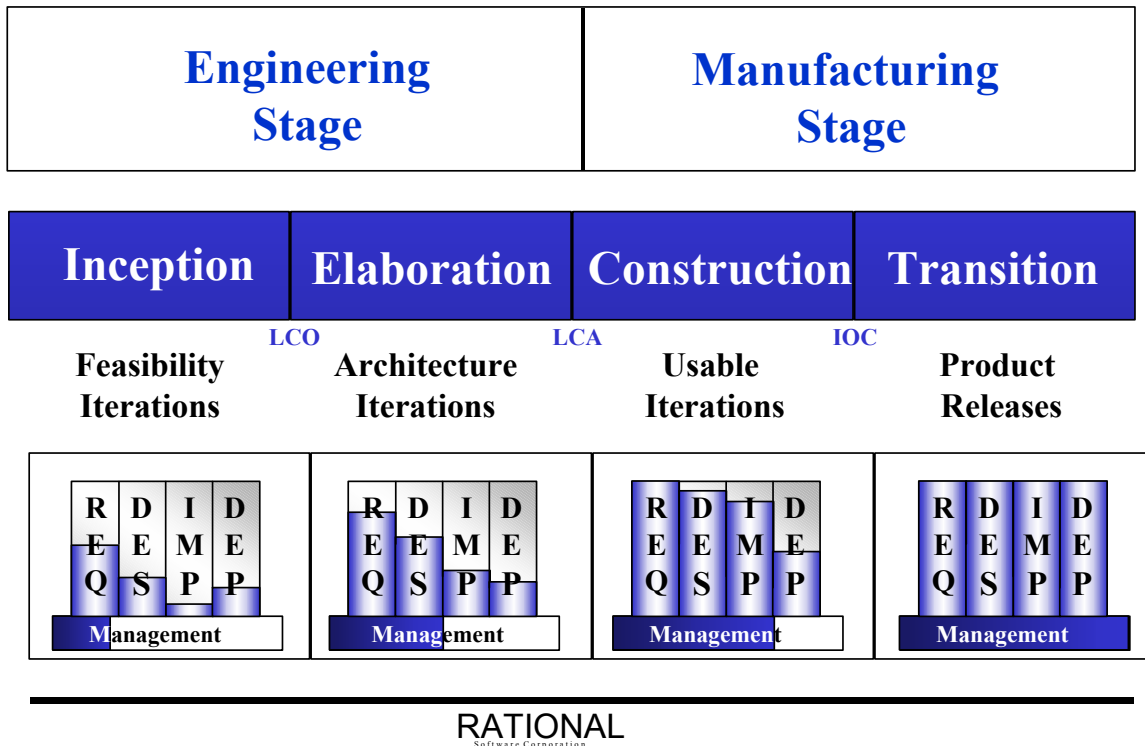
Milestone Element	Life Cycle Objectives (LCO)	Life Cycle Architecture (LCA)
Definition of Operational Concept	<ul style="list-style-type: none"> • Top-level system objectives and scope <ul style="list-style-type: none"> - System boundary - Environment parameters and assumptions - Evolution parameters • Operational concept 	<ul style="list-style-type: none"> • Elaboration of system objectives and scope by increment • Elaboration of operational concept by increment
System Prototype(s)	<ul style="list-style-type: none"> • Exercise key usage scenarios • Resolve critical risks 	<ul style="list-style-type: none"> • Exercise range of usage scenarios • Resolve major outstanding risks
Definition of System Requirements	<ul style="list-style-type: none"> • Top-level functions, interfaces, quality attribute levels, including: <ul style="list-style-type: none"> - Growth vectors - Priorities • Stakeholders' concurrence on essentials 	<ul style="list-style-type: none"> • Elaboration of functions, interfaces, quality attributes by increment <ul style="list-style-type: none"> - Identification of TBDs (to-be-determined items) • Stakeholders' concurrence on their priority concerns
Definition of System and Software Architecture	<ul style="list-style-type: none"> • Top-level definition of at least one feasible architecture <ul style="list-style-type: none"> - Physical and logical elements and relationships - Choices of COTS and reusable software elements • Identification of infeasible architecture options 	<ul style="list-style-type: none"> • Choice of architecture and elaboration by increment <ul style="list-style-type: none"> - Physical and logical components, connectors, configurations, constraints - COTS, reuse choices - Domain-architecture and architectural style choices • Architecture evolution parameters
Definition of Life-Cycle Plan	<ul style="list-style-type: none"> • Identification of life-cycle stakeholders <ul style="list-style-type: none"> - Users, customers, developers, maintainers, interpreters, general public, others • Identification of life-cycle process model <ul style="list-style-type: none"> - Top-level stages, increments • Top-level WWWWWHH* by stage 	<ul style="list-style-type: none"> • Elaboration of WWWWWHH* for Initial Operational Capability (IOC) <ul style="list-style-type: none"> - Partial elaboration, identification of key TBDs for later increments
Feasibility Rationale	<ul style="list-style-type: none"> • Assurance of consistency among elements above <ul style="list-style-type: none"> - Via analysis, measurement, prototyping, simulation, etc. - Business case analysis for requirements, feasible architectures 	<ul style="list-style-type: none"> • Assurance of consistency among elements above • All major risks resolved or covered by risk management plan

* WWWWWHH: Why, What, When, Who, Where, How, How Much

The second milestone is the Life Cycle Architecture (LCA) milestone, at which management verifies the basis for a sound commitment to product development and evolution (a particular system architecture with specific COTS and reuse commitments which is shown to be feasible with respect to budget, schedule, requirements, operations concept and business case; identification and commitment of all key life-cycle stakeholders; and elimination of all critical risk items). The AT&T/Lucent Architecture Review Board technique [Marenzano, 1995] is an excellent management verification approach involving the LCO and LCA milestones.

The third anchor point is the system's Initial Operational Capability (IOC), defined further in [Boehm, 1996] The LCO, LCA, and IOC have become the key milestones in Rational's Objectory or Unified Management Process [Rational, 1997; Royce, 1998] ; see Figure 4. Informally, the LCO milestone is the equivalent of getting engaged, the LCA milestone is the equivalent of getting married, and the IOC milestone is the equivalent of having your first child.

Figure 4. Objectory Information Set Evolution



3.5 MBASE Relations to Other Approaches

MBASE's expansion, Model-Based (System) Architecting and Software Engineering, acknowledges a strong dependence on the System Architecting approach developed in [Rechtin, 1991] and [Rechtin-Maier, 1997]. MBASE implements the System Architecting concepts of multi-criteria satisficing vs. optimizing (via stakeholder win-win negotiations); concurrent development of a system's operational concept, requirements, architecture, and life cycle plans (via the anchor point milestones); and placing inductive, synthetic, heuristic approaches ahead of deductive, analytic, formal approaches (via the risk-driven exploration of alternatives in the spiral model and the invention of win-win options for mutual gain [Fisher-Ury, 1981]).

The MBASE approach is applicable to systems engineering as well as software engineering. [Rechtin-Maier, 1997] indicates that hardware processes often need a waterfall-type approach to accommodate manufacturing facility preparation, tooling, and ordering of long-lead-time components. However, since for these sources of risk, the waterfall is a special case of the spiral model, the MBASE approach can adapt to such hardware concerns.

MBASE differs from Model-Based Systems Engineering (MBSE) [Fisher et al., 1998], in that MBSE concentrates almost exclusively on product models (and their associated property models). This is also the case for the Software Engineering Institute's Model-Based Software Engineering [Gargaro-Peterson, 1996] and Honeywell's Model-Based Software Development [Honeywell, 1998] approaches.

4. MBASE Object-Oriented Development: Overview and IT Example

4.1 Overview

The particular MBASE object-oriented approach to developing the executing software product spans both the upper and lower cycles of the MBASE process framework in Figure 2. In the early Inception and Elaboration stages preceding the LCO and LCA milestones (Figure 4), the MBASE spiral approach synthesizes an object-oriented system architecture from available COTS and infrastructure choices and early versions (entities, responsibilities, components, behaviors) of the classic OO artifacts such as objects, classes, and operations.

Figure 5 provides an overview of how the synthesis proceeds during the MBASE spiral cycles in the Inception, Elaboration, and Construction stages. It is presented in terms of the template used in the original spiral model paper [Boehm, 1988]. Figure 5 shows how risk identification and resolution activities guide the filtering of alternative COTS, architecture, and OO alternatives through the spiral cycles in each stage. It also shows how the more abstract OO predecessor artifacts such as entities and components are elaborated into OO classes and objects during the progress through the Inception, Elaboration, and Construction stages.

Of course, this overview figure is necessarily oversimplified, as the need to develop risk-driven portions of the operational software during the earlier stages will require some elaboration into classes, objects, operations, and OO code during the early Inception and Elaboration stages.

As discussed below, classical build-from-scratch OO approaches have encountered significant difficulties in dealing with challenges offered by emerging information technology (IT) applications. These include COTS and reuse, legacy systems, distributed collaboration support, and rapidly changing technology. We will illustrate how the MBASE OO approach addresses these challenges in the context of an emerging class of IT applications: the development of Web and Inter-/intranet-based collaborative systems. We will use a simple but representative and real example: a Web-

based system for scheduling common corporate resources such as conference facilities, auditoriums, training capabilities, or test facilities. The specific example we use below covers the scheduling of the Seaver Science Auditorium, a resource administered by the USC Seaver Science Library.

Figure 5. Spiral Approach to MBASE Iterations

-Synthetic, inductive vs. analytic, deductive

-Risk-driven level of detail; artifacts and stages overlap

	Inception - Stage Cycles	Elaboration-Stage Cycles	Construction-Stage Cycles
Objectives, Constraints	<ul style="list-style-type: none"> • Cost, schedule, performance, other goals • Top-level, most critical <ul style="list-style-type: none"> -activity use cases -workflows -entities -responsibilities 	<ul style="list-style-type: none"> • Top-level, most critical requirements <ul style="list-style-type: none"> -project, functional, interface, quality • Mid-level, mid-criticality <ul style="list-style-type: none"> -components -behavior use cases -entity/class relations 	<ul style="list-style-type: none"> • Essential requirements • Detailed specs <ul style="list-style-type: none"> -components -objects -operations -classes -data model -logical, layered, deployment views
Alternatives	<ul style="list-style-type: none"> • Major COTS, reusable components • Major architecture options <ul style="list-style-type: none"> -physical, logical • Major operational options <ul style="list-style-type: none"> -scenarios, task models 	<ul style="list-style-type: none"> • Mid-level COTS, reuse, architecture, operational options • Major data structure, control structure, algorithm options 	<ul style="list-style-type: none"> • Detailed tailoring of COTS, reuse, architecture, operational options • Detailed data structure, control structure, algorithm options
Evaluation, Risk ID	<ul style="list-style-type: none"> • Filter out unsatisfactory alternatives • Iterate objectives, constraints • Feature importance vs. difficulty prioritization • Identify key risks, uncertainties 	<ul style="list-style-type: none"> • Filter out unsatisfactory mid-level alternatives • Iterate objectives, constraints • Converge on likely best architecture • Identify all major risks 	<ul style="list-style-type: none"> • Evaluate detailed alternatives • Iterate objectives, constraints • Converge on likely best implementation • Monitor risk items
Risk Resolution	<ul style="list-style-type: none"> • Buy information <ul style="list-style-type: none"> -prototypes, models, analyses, interviews... • Choose resolution options <ul style="list-style-type: none"> -risk avoidance, transfer, management 	<ul style="list-style-type: none"> • Mid-level continuation of earlier risk resolution activities 	<ul style="list-style-type: none"> • Detailed continuation of earlier risk resolution activities
Results	LCO Package (Table 4)	LCA Package (Table 4)	IOC elements
Plan For Next Phase	<ul style="list-style-type: none"> • Detailed elaboration stage WWWWHH <ul style="list-style-type: none"> -mini spiral cycles • Residual-risk management plan 	<ul style="list-style-type: none"> • Detailed construction-Stage WWWWHH <ul style="list-style-type: none"> -mini-spiral cycles • Residual-risk management plan 	<ul style="list-style-type: none"> • Detailed transition-stage WWWWHH • Residual-risk management plan
Commitment	<ul style="list-style-type: none"> • LCO ARB review • Stakeholders commit to necessary resources, levels of participation 	<ul style="list-style-type: none"> • LCA ARB review • Stakeholders commit to necessary resources, levels of participation 	<ul style="list-style-type: none"> • IOC readiness review • Stakeholders commit to necessary resources, levels of participation

4.2 MBASE Object Orientation and Product Model Layers

The object oriented paradigm suggests viewing a system as a collection of interacting “things” known as objects. The specific definition of an object varies, however a fundamental characteristic is that an object is an embodiment of both memory and functionality that is capable of maintaining “state” or a notable collection of attributes that has significant effects on functionality. A primary purpose for using an object oriented approach is that it enables the representation of a complex system at a level of abstraction that is closer to the domain in which the system is conceived. This has many direct benefits such as expanded coverage of functionality per model element (i.e. “chunking”), complexity management through encapsulation, extensibility and flexibility from high modularity, and robustness and maintainability from “natural” abstractions that are more closely aligned with the way people think [Booch, 1991]. The price for this is that object designed systems tend to be less observable and controllable [Weide et. al., 1996] and the approach is not always suitable for all systems. In particular, highly optimized embedded and/or real-time systems are often easier to model as flows of data rather than interacting objects. Generally sophisticated management information systems (MIS) with a non-trivial integration with other systems are well served by the object oriented approach. This is because such systems are generally more “people” rather than “technology” focused [Port, 1999]. The classic object oriented conundrum here is “how do you find the objects?” More specifically, given the concept of a system, how do you determine what objects are needed to faithfully represent that system in software? By faithfulness we mean there exists a well-defined, sound relationship between the software and the original system concept it represents.

The problems described above are exacerbated by the ever-increasing economic need to incorporate COTS, legacy systems, and general re-use of technology into the development process. The difficulty here is that pre-existing entities such as COTS exist at a higher level of abstraction than objects. As such, entities like COTS can not be dealt with (i.e. modeled) in the same way as objects. For example, COTS products are usually considered earlier than objects, often by non-developer kinds of stakeholders, and are less flexible in that they come with pre-set constraints with potential product model classes such as architectural style commitments and fixed interfaces, exacerbated by providing no access to source code. This problem is actually more general in that model elements at widely varying levels of abstraction are constantly being introduced throughout development cycle. This is particularly challenging on large systems with multiple stakeholders, who will introduce potential model clashes via win conditions on COTS compatibility with their existing systems or on COTS/system performance, dependability or portability objectives. Later we will see how MBASE architecting deals with this problem within the object identification problem through model evolution.

The object determination problem has the two, not necessarily orthogonal, dimensions of orientation and specificity (see Figure 6). Orientation, as indicated earlier in the differences between MIS and embedded systems, covers the required qualities for “people” concepts at one extreme and “technology” on the other. Specificity covers the

level of abstraction of the concepts, ranging from “general” to “specific”. In this light the problem can be stated as a problem of translating a conceptual systems general-people oriented abstractions into faithful specific-technology focused representations. The dimensions at a crude level indicate four broad, yet distinguishable development layers that fit well into a WWHD (Why/where, What, How, Do) spiral process. Although the recursive hierarchical nature of why-what-how relationships has been well-known for some time [Ross-Schoman, 1977], within the context of Figure 6 the quadrants can be considered to relatively address the questions of why/where; what; how; and "do;" in the context of one’s current spiral cycle.

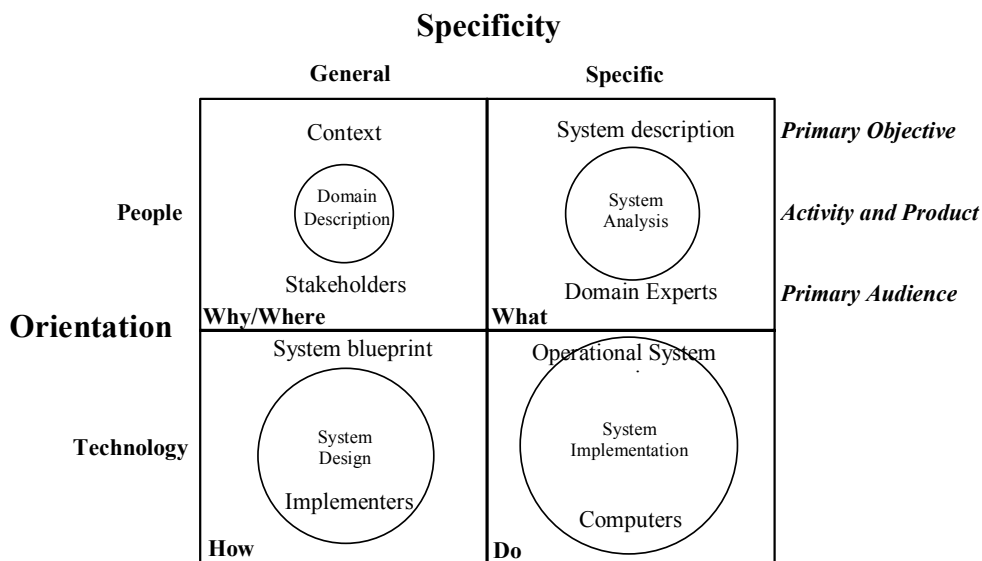


Figure 6. MBASE Product Model Relationships

Although the dimensions of orientation and specificity are not entirely orthogonal, they are independent enough to provide natural software development layers in terms of the validators of the model elements (to whom we refer as the audience for the layer) such as stakeholders, domain experts, implementers, and of course computers. The process of translating general-people kinds of abstractions into specific-technology oriented abstractions indicate four development layers that must be concurrently elaborated and continuously reconciled in order to reify a software product. These layers are supported by environments to create model elements using a functional decomposition process. The layers are:

General-People: Abstractions created here describe parts of the overall domain or environment of the system. This brings the Domain/Environment Models into the Product Model creation process as a natural starting point and relevance tool by setting forth the "why" a project is being done, and "where" it's starting from. The focus and scope of these models is determined during the exploration of all the alternative concepts for improving the system. As shown in Figure 6, the Domain/Environment Models are validated by the stakeholders. The activity and product of this layer is referred to as the

"Domain Description." For our resource-scheduling example IT application, the Domain Description includes the background of the Seaver Science Library; its mission; goals such as "Provide equal and efficient access to Library facilities"; and descriptions of the current manner that the library auditorium is scheduled, descriptions of the basic entities and their activities related to the Auditorium that are relevant to the resource-scheduling system, such as "Auditorium Calendar" and "Manage use of Seaver Science Auditorium".

Specific-People: Concurrently, a determination must be made for people to understand "what" software product is to be developed in order to provide equal and efficient access to the use of the Auditorium. The model elements which describe this are best validated (to reduce risk of unfaithfulness) by domain experts, or people who understand the system concept well and can make decisions on what should or should not be represented in software. This involves careful consideration of particular domain elements and exactly what people require from the system. In this regard, the term "System Analysis" is an appropriate description of the overall activities performed and product developed in this layer. Continuing with the IT example, the entity "Auditorium Calendar" and the activity "Manage use of Seaver Science Auditorium" in the Domain Description is elaborated into the components "Seaver Auditorium Reservation Schedule," "Auditorium Reservation", and system responsibilities such as "Manage Auditorium reservations" within the System Analysis. Since all the qualities an entity and activity possesses may not be relevant to the proposed system, elaboration specifies precisely what with respect to the domain we do wish to have the system represent.

General-Technology: The issues in this layer resolve "how" the high-level system description can be realized as a software system. As such, it must describe the overall design of the software system and how it represents the system concept. The activity and product in this layer is called the system and software Design. The technical feasibility of actually implementing the blueprint it contains can only be validated by the implementers; thus they are the primary audience. Here, for example, the component "Seaver Auditorium Reservation Schedule" in the System Analysis is elaborated into such objects such as "Reservation Schedule," "Reservation," and "Reservation Scheduler," and "Reservation User Interface." The system responsibility, "Manage Auditorium reservations" gives rise to the requirements, "GUI selection of reservations," and "Administrate reservations database," the latter in turn gives rise to the operations, "make reservation," "modify reservation," "delete reservation" in the System Design.

Specific-Technology: For this layer, the implementers reify the System Design into computer configuration directives and software code. The software can then be targeted to and validated by the software system's computer configuration (often including operating system, DBMS, compiler, etc.). This layer is where the actual software and system representation is accomplished, or the "do" part of development; the activity and product is given the name "System Implementation." In our example, the "Reservation User Interface" might be a Java GUI that directly manipulates a "Reservation" object residing as a row in an Oracle database. The operations "make reservation," "modify

reservation,” and “delete reservation” might be implemented as a combination of events in the Reservation User Interface that generate SQL queries to Oracle.

4.3 Object Oriented Architecting

Architecting is the process of constructing the product model layers in such a way that there exists a sound, well-defined mapping between the implementation’s specific computer and software technology configuration and the Domain Description’s general-people oriented abstractions. If successful, the resulting collection of models that constitute the model layers (which consist of representations of various kinds of entities, their relationships and constraints) are considered to be the system’s architecture conception and realization. This definition is compatible with and extends other definitions of software architecture, see [Shaw-Garlan, 1996] for example. The MBASE extension of the software architecture into the full Life Cycle Architecture package in Table 4 serves as a “bridge” over the gap formed between the concept of a system and a representation of that system in software (see [Boehm et al., 1999]). Figure 7 shows the general sequence in which information evolves through the model layers, the language used, and their relative size (with respect to number of model elements).

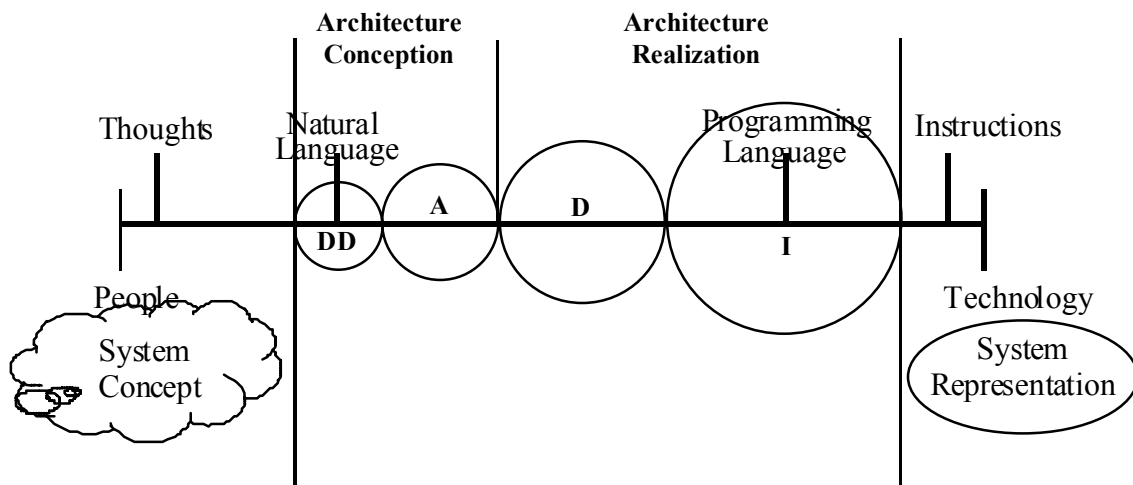


Figure 7: The Architectural Bridge

Note that the language used within each layer matches with the language of the primary audience for that model layer (as described above) and that there is an important distinction made between high-level architecture which addresses user and customer concerns (i.e. people) and low-level architecture that addresses technology. This distinction reduces risk of “losing the forest for the trees” while maintaining the ability to provide meaningful tracing between the layers. Note further that the progression through the LCO and LCA milestone packages in Table 4 and the Inception, Elaboration, and Construction stages in Figure 4 is not trying to bridge the gap sequentially, but it is creating a sequence of increasingly robust bridges across the gap. The relative height of the REQ, DES, and IMP bars in Figure 4 indicate the relative width of the Requirements,

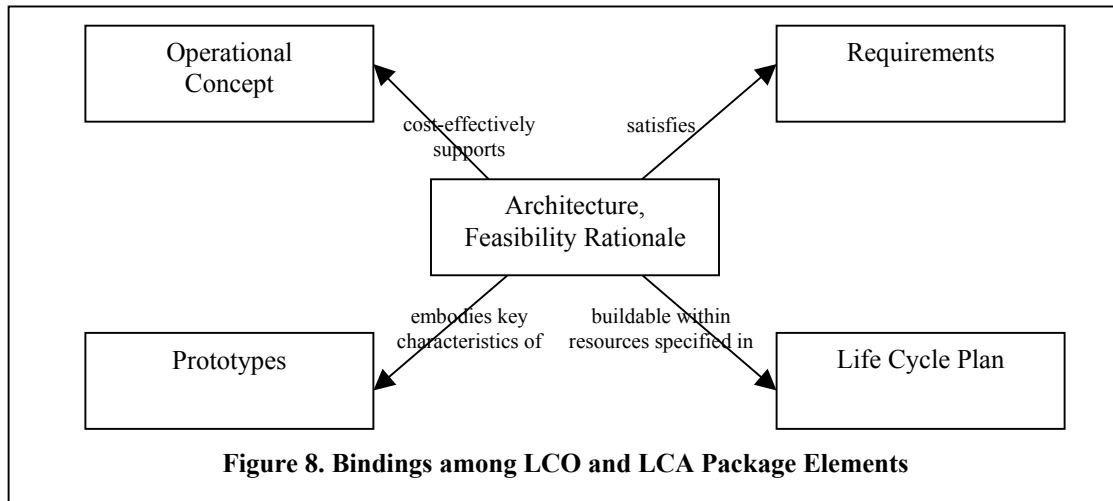
Design, and Implementation parts of the bridge at the LCO, LCA, and IOC milestones. For example, the LCO package for the scheduling system included 4 Project Requirements, 16 System Requirements, 16 Quality Attribute Requirements, 1 Interface Requirement, 7 Environment and Data Requirements, and 4 Evolution Requirements. In comparison there were 7 objects specified for 7 components, and 13 operations detailed in the Design. By LCA the number of objects and operations will greatly exceed the number of components and requirements.

4.4 Binding the Bridge Together

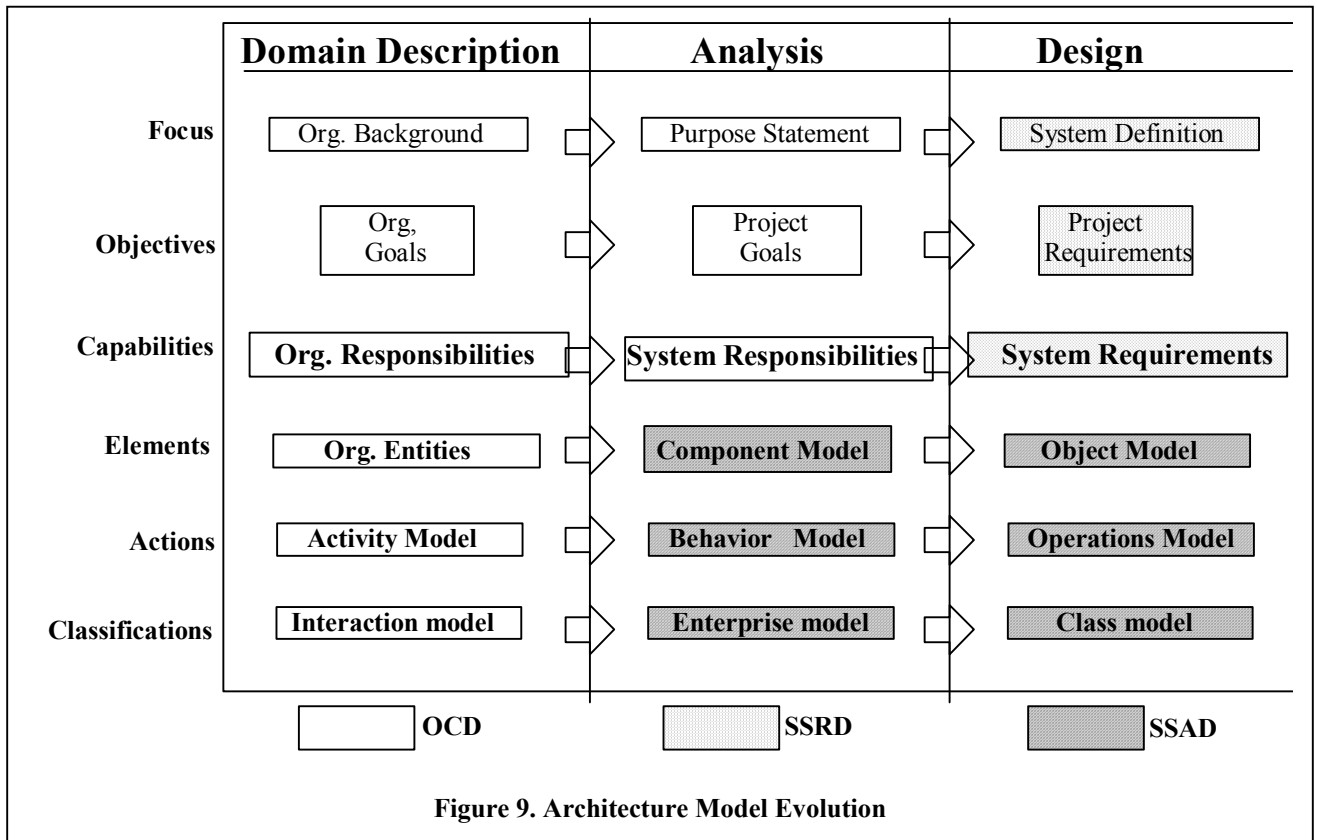
Bridging the gap between people with their system concepts and computers with their software infrastructure has been a major software and systems engineering challenge. For MBASE, it requires strong relationships among the various system and project views in the LCO and LCA packages, and strong relationships among the layers of system and software definition which become the Operational Concept Description (OCD), Requirements Description (RD), Architecture Description (AD), and ultimate system Implementation (I).

The binding relationships in the LCO and LCA packages are provided primarily via the AD and Feasibility Rationale (FR) elements (see figure 8). The main binding mechanism is the success criterion for the FR: it must demonstrate that a system built to the specifications in the Architecture Description

- will cost-effectively support the Operational Concept,
- will satisfy the Requirements,
- will embody the key characteristics of the Prototypes, and
- will be buildable within the budget and schedule in the Life Cycle Plan.



The bindings among the Domain Description, Analysis, Design and Implementation layers are a series of increasingly detailed representations of the software/system objectives, capabilities, elements, behaviors, and relationships (See Figure 9)



The overall bindings between the layers are formed through the specific evolutionary paths between analogous models as depicted in Figure 9. Along each path the models are increasingly reified in a well-defined and traceable way, ultimately translating General-People concepts into Specific-Technology structures. For example, the Behavior Model is derived by considering what specific activities from the Activity Model people desire the system to support (i.e. Specific-People from General-People). The next step is to detail the steps required by the software to carry out the behaviors. The basic steps that carry out system a system behavior are called operations. Following the examples from our IT resource scheduling system discussed earlier, one of the library organizational activities from the Domain Description is “Manage use of the Seaver Science Auditorium”. This activity gives rise to several behaviors within the System

Analysis which collectively carry out the activity with respect to the System Responsibility, “Manage auditorium reservations”. The behaviors are: “View auditorium reservations”, “Reserve auditorium”, and “Modify auditorium reservation”. The behaviors have sub-behaviors that detail what specific behaviors the system should have. For example the “Reserve auditorium” behavior had sub-behaviors “one time”, and “recurring”. Each behavior or sub-behavior is detailed within the Design as a sequence of operations that detail exactly how the behaviors are to be carried out by the system with respect to a system requirement (related to the system responsibility for the behaviors). The aforementioned behavior is related to the System Requirement, “GUI selection of reservations” and has the operations “select reservation begin time/date, select reservation end time/date, select recurrence rate: {daily, weekly, monthly, yearly}, select recurrence date range, check recurrence validity, identify schedule conflicts, suggest conflict resolutions, confirm reservation”.

These steps illustrate an overall evolutionary spiral cycle between the models within each layer. However there are mini-spirals that bind the models within each layer. Each model layer starts with a model that provides focus, objectives are then set, capabilities enumerated, elements supporting the capabilities are identified, actions the elements may take (with respect to the capabilities) are detailed, then finally classifications are made to collect elements and actions together and simplify the models. This sequence may be repeated several times producing increasingly refined and consistent models.

The result of the bindings across and within the model layers is a guided and flexible definition of consistent and traceable artifacts at various levels of abstraction. This provides a tangible means to access and assure desirable qualities such as maintainability, extensionality, scalability, COTS introduction, assured requirements satisfaction, and complexity reduction to name a few.

4.5 Comparison of MBASE and Other OO Approaches

In our view, the MBASE OO approach attempts to integrate and extend other object oriented approaches such as OMT, Booch, and Jacobson OOSE in several areas. First, most other approaches concentrate solely on one or two of the model layers. For example the Jacobson Use-Case OOSE approach focuses strongly on the Analysis layer. MBASE provides unified integration of the Domain Description as an equal and first class model layer whereby the entire development process adapts to the project’s specific domain. Domain specific approaches are not new and have been applied effectively in several related areas, see for example [Tracz, 1966] as an example.

A second distinction is that other approaches are less specific about how the model layers are applied within an evolutionary lifecycle process – specifying the general order in which the various models are created, by whom, and how the model elements evolve within and throughout the model layers. A third area is that MBASE embodies a reference model (from Capabilities) as indicated in [Balzer, 1997] that provides a basis from which model elements such as components and behaviors are drawn, modeled, and ultimately reconciled and classified in a traceable and consistent manner. This approach

provides a tangible means of obtaining assurance and soundness in requirements satisfaction.

A fourth difference, and perhaps a key differentiation, is that within MBASE classification is performed last as part of an independent “engineering” process. This is important for producing faithful models. Many OO approaches advocate identifying classes [Booch 1991, Rumbaugh, 1991] early on, however this presents some difficulty when attempting to create representations that are faithful to the domain as it is difficult to identify natural groupings of model elements, such as objects, without first having a set of objects that are known to be faithful to group (a variation on the chicken and egg problem). Attempts to identify classifications of model elements directly from the domain often leads to haphazard, redundant, unnecessary, unbalanced, and unnecessarily complex groupings with a large degree of inconsistency, incompleteness and redundancy, leading to unfaithful representations. For instance, when classification is done too early, it is common to create two classifications for the same element described in two different ways (or two components that are operationally equivalent), or a component in two different states or modes, or use different names for the same element. Such occurrences are a nuisance when constructing a system from scratch, but become truly confounding when dealing with components such as COTS and legacy systems.

The Kruchten 4+1 approach [Kruchten, 1999] is similar in spirit to the MBASE approach. MBASE tries to take this even further than 4+1 to provide capabilities for dealing with:

- integration of COTS, re-use, and legacy systems,
- risk identification and mitigation,
- economic concerns, including feasibility analysis,
- explicit entry and exit criteria (through success models)
- architecture and architecting,
- stakeholder concurrence,
- language and audience issues
- quality attributes and their tradeoffs,
- requirements documents,
- rationale capture and documentation

The next section summarizes our experience to date in trying to integrate all these considerations.

5. Example MBASE Applications

5.1 Digital Library Multimedia Archive Projects

The description above sounds somewhat complicated, and one can wonder how quickly and effectively it can be learned and applied. Our first opportunity to apply the MBASE approach to a significant number of projects came in the fall of 1996. We arranged with the USC Library to develop the LCO and LCA packages for a set of 12 digital library multimedia applications. The work was done by 15 6-person teams of students in our graduate Software Engineering I class, with each student developing one

of the 6 LCO and LCA package milestone elements shown in Table 4. Three of the 12 applications were done by two teams each. The best 6 of the LCA packages were then carried to completion in our Spring 1997 Software Engineering II class.

The multimedia archives covered such media as photographic images, medieval manuscripts, Web-based business information, student films and videos, video courseware, technical reports, and urban plans. The original Library client problem statements were quite terse, typically containing about 4-5 sentences. Our primary challenge was to provide a way for the student teams to work with these clients to go from these terse statements to an LCO package in 6 weeks and an LCA package in 11 weeks.

We enabled the students and clients to do this by providing them with a set of integrated MBASE models focused on the stakeholder win-win success model; the WinWin Spiral Model as process model; the LCO and LCA artifact specifications and a multimedia archive domain model as product models; and a property model focused on the milestones necessary for an 11-week schedule. Further details are provided in [Boehm et al., 1997] and [Boehm et al., 1998].

MBASE Model Integration for LCO Stage

The integration of these models for the LCO stage is shown in Figure 10. The end point at the bottom of Figure 10 is determined by the anchor point postconditions or exit criteria for the LCO milestone [Boehm, 1996]: having an LCO Rationale description which shows that for at least one architecture option, that a system built to that architecture would include the features in the prototype, support the concept of operation, satisfy the requirements, and be buildable within the budget and schedule in the plan.

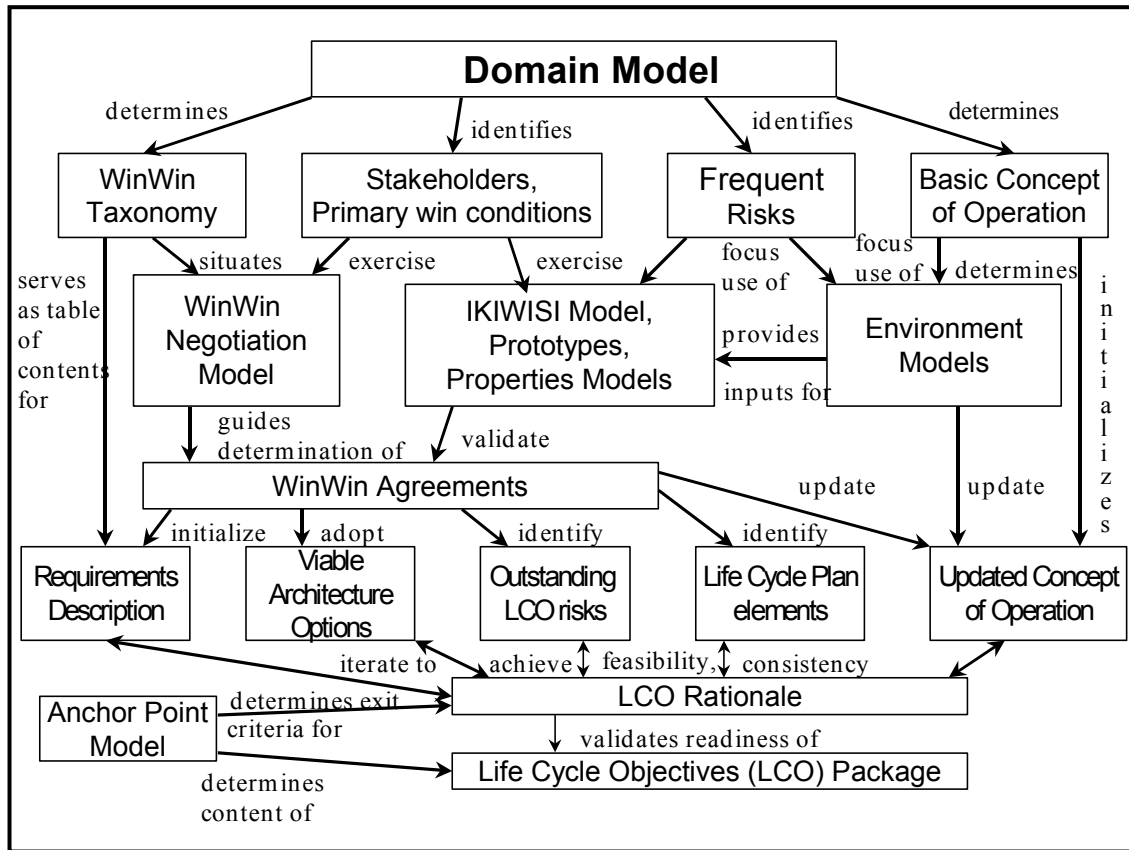


Figure 10. MBASE Model Integration: LCO Stage

The beginning point at the top of Figure 10 was a multimedia archive domain model furnished to the students. The domain model included the system boundary, its major interfaces, and the key stakeholders with their roles and responsibilities. The domain model also established a domain taxonomy used as a checklist and organizing structure for the WinWin requirements negotiation system furnished to the teams.

As shown at the left of Figure 10, this taxonomy was also used as the table of contents for the requirements description, ensuring consistency and rapid transition from WinWin negotiation to requirements specification. The domain model also indicated the most frequent risks involved in multimedia archive applications. This was a specialization of the list of 10 most frequent software risks in [Boehm, 1989], including

performance risks for image and video distribution systems; and risks that users could not fully describe their win conditions, but would need prototypes (IKIWISI).

The sequence of activities between the beginning point and the LCO end point were determined by the WinWin Spiral Model. As illustrated in Figure 3, this model emphasizes stakeholder win-win negotiations to determine system objectives, constraints and alternatives; and early risk identification and resolution via prototypes and other methods [Boehm-Bose, 1994].

Library Project Results

We were not sure how many of the 6-student teams would be able to work concurrently with each other and with their Library clients to create consistent and feasible LCO packages in 6 weeks and LCA packages in 11 weeks. With the aid of the integrated MBASE models, all 15 of the student teams were able to complete their LCO and LCA packages on time (3 of the applications were done separately by 2 teams). The Library clients were all highly satisfied, often commenting that the solutions went beyond their expectations. Using a similar MBASE and WinWin Spiral Model approach, 6 applications were selected and developed in 11 weeks in the Spring of 1997. Here also, the Library clients were delighted with the results, with one exception: an overambitious attempt to integrate the three photographic-image applications into a single product.

The projects were extensively instrumented, including the preparation of project evaluations by the librarians and the students. These led to several improvements in the MBASE model provided to the student teams for Fall 1997, in which 16 teams developed LCO and LCA packages for 15 more general digital library applications. For example, in 1996, the WinWin negotiations were done before the LCO milestone, while the prototypes were done after the LCO milestone. This led to considerable breakage in the features and user interface characteristics described in the LCO documents, once the clients exercised the prototypes. As a result, one of the top three items in the course critiques was to schedule the prototypes earlier. This was actually a model clash between a specification-oriented stakeholder win-win success model and the prototype-oriented IKIWISI success model. The 1997 MBASE approach removed this model clash by scheduling the initial prototypes to be done concurrently with the WinWin negotiations.

Another example was to remove several redundancies and overlaps from the document guidelines: as a result, the 1997 LCO packages averaged 110 pages as compared to 160 in 1996. The 1997 LCA packages averaged 154 pages as compared to 230 in 1996. A final example was to strongly couple the roles, responsibilities, and procedures material in the Operational Concept Description with the product transition planning, preparation, and execution activities performed during development. Further information on the 1997-98 projects is provided in [Boehm et al., 1998].

Continuing USC Library satisfaction with the 1997-98 project results, and increased student interest in the course (some local employers will only hire people who have taken the course), have led to our having 20 student teams defining 1998-99 digital library products. Our main improvement was to provide considerably more detailed guidelines for defining the LCA and LCO package elements. We have just completed the LCO reviews, with a strong reduction in several types of model clash. But we continue to find additional model clashes pointing toward further needed MBASE refinements. 1996-97, 1997-98, and 1998-99 projects can be accessed via the USC-CSE web site at <http://sunset.usc.edu/classes/classes.html>.

MBASE Application to Large Projects

There is also increasing evidence that the MBASE approach scales up well to large projects. The successful million-line CCPDS-R project described in [Royce, 1998] used predecessors of the MBASE model elements. Numerous organizations are using portions of MBASE on large projects as part of Rational's Objectory or Unified Management Process [Rational, 1997; Royce, 1998]. Other organizations such as Xerox and FAA are in the process of integrating MBASE elements into their software and systems engineering processes.

6. Conclusions

We have found via a number of project analyses that the model-clash concept helps considerably in understanding the sources of stickiness in the software project tar pit. We have found in a number of reviews of large and small projects that the MBASE approach is helpful for both diagnosing and avoiding model clashes which would otherwise have seriously compromised the project. And we have found in using the MBASE approach on over 30 projects that it is effective for the rapid architecting and development of leading-edge information technology applications.

MBASE is still a work in progress. Much more needs to be done in characterizing model clashes and their effects. The current MBASE guidelines need testing in a number of domains, and refinement based on the testing and evaluation results. We welcome further efforts toward achieving these ends.

7. References

[Babcock, 1985]. C. Babcock, "New Jersey Motorists in Software Jam," ComputerWorld, September 30, 1985, pp. 1, 6.

[Balzer, 1996]. R. Balzer. Current state and future perspectives of software process technology. Keynote Speech, Software Process (SP 96), Brighton, 2-6 December 1996.

[Boehm, 1989]. B. Boehm, Software Risk Management, IEEE-CS Press, 1989.

[Boehm-Bose, 1994]. B. Boehm and P. Bose, "A Collaborative Spiral Process Model Based on Theory W," Proceedings, ICSP3, IEEE, 1994.

[Boehm, 1996]. B. Boehm, "Anchoring the Software Process," IEEE Software, July 1996, pp. 73-82.

[Boehm-In, 1996]. B. Boehm and H. In, "Identifying Quality-Requirement Conflicts," IEEE Software, March 1996, pp. 25-35.

[Boehm, et al., 1997]. B. Boehm, A. Egyed, J. Kwan, and R. Madachy, "Developing Multimedia Applications with the WinWin Spiral Model," Proceedings, ESEC/ FSE 97, Springer Verlag, 1997.

[Boehm, 1998]. B. Boehm, "A Spiral Model of Software Development and Enhancement," Computer, May 1988, pp. 61-72.

[Boehm et al., 1998]. B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, "Using the Win Win Spiral Model: A Case Study," IEEE Software, July 1998, pp. 33-44.

[Boehm, et al., 1999]. B. Boehm, D. Port, A. Egyed, and M. Abi-Antoun, "The MBASE Life Cycle Architecture Milestone Package: No Architecture Is An Island," to appear in Proceedings of the 1st Working International Conference on Software Architecture, 1999.

[Booch, 1991], G. Booch, Object Oriented Design With Applications, Benjamin/Cummings, 1991

[Brooks, 1975]. F. Brooks, The Mythical Man-Month, Addison-Wesley, 1975 (2nd ed., 1995).

[Fisher et al., 1998]. J. Fisher et al., "Model-Based Systems Engineering: A New Paradigm," INCOSE INSIGHT, October 1998, pp. 3-16.

[Fisher-Ury,1981]. R. Fisher and W. Ury, Getting To Yes, Houghton-Mifflin, 1981.

[Gargaro-Peterson, 1996]. A. Gargaro and A.S. Peterson, "Transitioning a Model-Based Software Engineering Architectural Style to Ada 95," SEI Technical Report CMU/SEI-96-TR-017, 1996. See also <http://sei/cmu.edu/mbse/is.html>

[Garlan et al., 1995]. D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch: Why Reuse is So Hard," IEEE Software, November 1995, pp. 17-26.

[Honeywell, 1998]. Honeywell Technology Center, "Model-Based Software Development," Course Announcement, Minneapolis, MN, 1998.

[Jackson, 1975]. M. A. Jackson, Principles of Program Design, Academic Press, 1975.

[Jacobson et al., 1997]. I. Jacobson, M. Griss, and P. Jonsson, Software Reuse, Addison Wesley, 1997.

[Kazman et al; 1994]. R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architectures," Proceedings, ICSE 16, ACM/IEEE, 1994, pp. 81-90.

[Kruchten, 1999]. P. Kruchten, The Rational Unified Process, Addison Wesley, 1999.

[Lee, 1996]. M.J. Lee, Formal Modeling of the WinWin Requirements Negotiation System, Ph.D. Thesis, USC Computer Sciences Dept., 1996.

[Madachy, 1995]. R. Madachy, "Knowledge-Based Risk Assessment Using Cost Factors", Automated Software Engineering, 2, 1995.

[Marenzano, 1995]. J. Marenzano, "System Architecture Validation Review Findings," in D. Garlan, ed., ICSE17 Architecture Workshop Proceedings, CMU, Pittsburgh, PA 1995.

[Port, 1999]. D. Port, Integrated Systems Development Methodology, Telos Press, 1999 (to appear).

[Rational, 1997]. Rational Objectory Process, Version 4.1, Rational Software Corp., Santa Clara, CA, 1997.

[Rechtin, 1991]. E. Rechtin, Systems Architecting, Prentice Hall, 1991.

[Rechtin-Maier, 1997] E. Rechtin and M. Maier, The Art of System Architecting, CRC Press, 1997.

[Rosove, 1967]. P.E. Rosove, Developing Computer-Based Information Systems, John Wiley and Sons, Inc., 1967.

[Ross-Schoman, 1977]. D.T. Ross and K.E. Schoman, "Structured Analysis for Requirements Definition," IEEE Trans. SW Engr., January 1977, pp. 41-48.

[Royce, 1998]. W.E. Royce, Software Project Management: A Unified Framework, Addison Wesley, 1998.

[Shaw-Garlan, 1996]. M. Shaw and D. Garlan, Software Architecture, Prentice Hall, 1996.

[Tracz, 1996]. W. Tracz, "Domain Analysis, Domain Modeling, and Domain-Specific Software Architectures: Lessons Learned", IEEE Fourth International Conference on Software Reuse, Orlando, FL, April 1998, p.232-233

[Weide et. al., 1996]. B.W. Weide, S.H. Edwards, W.D. Heym, T.J. Long, and W.F. Ogden, "Characterizing Observeability and Controlability of Software Components", IEEE Fourth International Conference on Software Reuse, Orlando, FL, April 1998, p.62-71