Conceptual Modeling Challenges for Model-Based Architecting and Software Engineering (MBASE)

BARRY BOEHM AND DAN PORT

USC Center for Software Engineering

(boehm, dport@sunset.usc.edu)

1. Summary

The difference between failure and success in developing a software-intensive system can often be traced to the presence or absence of clashes among the models used to define the system's product, process, property, and success characteristics. (Here, we use a simplified version of one of Webster's definitions of "model" a description or analogy used to help visualize something. We include analysis as a form of visualization).

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 models. Section 3 introduces the Model-Based Architecting and Software Engineering (MBASE) approach for endowing a software project with a mutually supportive base of models. Section 4 presents examples of applying the MBASE approach to a family of digital library projects.

Section 5 summarizes the main conceptual modeling challenges involved in the MBASE approach, including integration of multiple product views and integration of various classes of product, process, property, and success models. Section 6 summarizes current conclusions and future prospects.

2. Model Clashes and the Need to Avoid Them

The concept of model clashes among software product models has been addressed in several forms. 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 reuseable software components [Garlan et.al., 1995].

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.

Less well appreciated is the importance of two other classes of models--software-system property models and success models--and the need to apply all four of these classes of models in consistent and integrated ways.

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.

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.

Unexamined success models can also cause serious clashes. A good example is the Golden Rule, "Do unto others as you would have others do unto you." An expert programmer will frequently interpret this to mean, "I would like people to build me systems with terse, maybe obscure, but powerful commands; and direct access to system files and operating system calls; so that's how I should build systems for others." This can cause many serious problems when the "others" are doctors, airplane pilots, or fire dispatchers, who would be better served by a user-task oriented interface resulting from a stakeholder win-win success model.

Table 1 provides a model-clash matrix showing that serious model clashes can come from any combination of product, process, or success models.

Table 1. Examples of Model Clashes

Table 1

3. MBASE Overview

Figure 1 summarizes the overall 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 various success models, whose priorities and consistency should be considered first. Thus, if the overriding top-priority success model is to "Demonstrate a competitive agent-based data mining system on the floor of COMDEX in 9 months," this constrains the ambition level of other success models (provably correct code, fully documented as a maintainer win condition). It 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. 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.

Anchor Point Milestones

In each case, 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. It has been found advisable to do this especially at two particular "anchor point" life cycle process milestones summarized in Table 2 [Boehm, 1996].

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.

The second milestone is the Life Cycle Architecture (LCA) milestone, at which management verifies the basis for a sound commitment to product development (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 LCO and LCA have also become key milestones in Rational's Objectory process [Rational, 1997].

Figure 1

Figure 1. MBASE Integration Framework

4. Example MBASE Application

4.1 Digital Library Multimedia Archive Projects

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 artifacts shown in Table 2. 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.

Table 2. Contents of LCO and LCA Milestones

Table 2

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, as indicated in Table 3. 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 7 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 (see Figure 2). Further details are provided in [Boehm et al, 1997].

Table 3. Example Library Multimedia Problem Statements

Table 3

Figure 2

Figure 2. Multimedia Archive Project Guidelines

4.2 MBASE Model Integration for LCO Stage

The integration of these models for the LCO stage is shown in Figure 3. The end point at the bottom of Figure 3 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 3

Figure 3. MBASE Model Integration: LCO Stage

The beginning point at the top of Figure 3 is the multimedia archive extension domain model furnished to the students, illustrated in Figure 4. The parts of the domain model shown in Figure 4 are 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 3, 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).

Figure 4

Figure 4. Multimedia Archive Extension Domain Model

The sequence of activities between the beginning point and the LCO end point were determined by the WinWin Spiral Model. As illustrated in Figure 5, 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].

Figure 5

Figure 5. The WinWin Spiral Model

4.3 Project Results

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 have 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 103 pages as compared to 160 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. Most of the artifacts from the 1996-97 and 1997-98 project can be accessed via the USC-CSE web site at http://sunset.usc.edu/classes.

5. Conceptual Modeling Challenges

5.1 MBASE Conceptual Framework

Figure 6 provides an overall conceptual 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 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 (cf. Figure 4).

The upper loop in Figure 6 shows how the stakeholders use their success models, and associated property models (e.g. cost models and revenue or effeciveness models for the business case analysis) to determine the appropriate operating region for the project. As shown in Figure 7, the operating region consists of operating points, each representative of a choice of system product specification and life-cycle process (including such choices as staffing, methods, and tools). The space of conceptual multimedia archive operating points is shown as an example domain.

Figure 6

Figure 6. MBASE Conceptual Framework

Success Models and Metrics

Figure 7 also shows how the stakeholder-determined success models impose metrics on the space of (product, process) operating points, via the use of property models, which enable reasoning about the various success attributes (Figure 6). For example, if a 5 month initial operational capability (IOC) is a critical success condition, a cost-schedule estimation model such as COCOMO, Checkpoint, or PRICE S will determine the estimated schedule for the project as a function of its domain (precedentedness, required reliability, application complexity), product (size, performance, platform characteristics), and process (staffing, methods, and tools) characteristics.

The results of these property-model estimates are shown in conceptual form in figure 7. The “5 month” line divides the space of operating points into systems buildable in more or less than 5 months. Similarly, if other key success conditions include 1-second response-time performance and a set of core product features (e.g. for a digital image archive) these also determine metrics which divide the space of operating points into satisfactory and unsatisfactory regions.

The intersection of the satisfactory regions for each success condition determines the satisfactory operating region for the project. Often, the initial

satisfactory operating region will be empty, and the stakeholders will iterate around the upper loop of Figure 6, adjusting and renegotiating their success models and ambition levels for the domain, product, and process until a feasible operating region is generated. In the context of the stakeholder win-win success model, the satisfactory regions are determined by stakeholder win conditions, and the win-win region is similarly the intersection of these regions [Lee, 1996].

Figure 7

Figure 7. Success Models Impose Metrics on Space of (Domain, Product, Process) Operating Points

MBASE Product Model Integration

The upper loop in Figure 6 primarily involves top-level conceptual models of the product and process, although these are selectively elaborated in specifics to address critical negotiation issues or to resolve high-risk uncertainty areas. The lower loop in Figure 6 primarily involves elaboration of the conceptual product models through a sequence of intermediate product models until these are reified to the point that a compiler or applications generator can transform them into an executable computer program for a particular platform configuration. This elaboration is guided by the chosen process models, although product choices (COTS components, re-engineered legacy code, hardware choices) may impose constraints on the process models.

Further, these constraints (or new opportunities) may cause a reconsideration of the top-level success models and criteria, and a backtracking process through the upper loop of Figure 6. Finally, the outer loop in Figure 6 involves evaluating how well the reified, executing product actually serves and satisfies the stakeholders, often leading to further traversals of the upper and lower loops as the system evolves through its life cycle.

Intermediate Product Models

The overall MBASE approach draws on, has contributed to, and shares many aspects with Rational's Objectory process [Rational, 1997] and Unified Software

Management approach [Royce, 1998]. The particular MBASE object-oriented approach to defining intermediate product models follows the Integrated Systems Development Methodology (ISDM) [Port, 1998].

In the ISDM approach, the conceptual product models in Figure 6 are focused on people-understandable, more general operations-concept and domain entities, attributes, and relationships. The reified product models are automatically translatable into specific-technology, computer-executable computer programs. The transition from general-people oriented representations to specific-technology oriented representations involves sets of specific-people oriented representations (generally associated with requirements or object-oriented analysis) and general-technology oriented representations (generally associated with object-oriented design); see Figure 8. 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 8 the quadrants can be considered to relatively address the questions of why/where; what; how; and "do."

Figure 8

Figure 8. ISDM Product Model Relationships

The people-technology and general-specific distinctions in Figure 8 are not entirely orthogonal, however 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 functional decomposition. 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. As

shown in Figure 8, the Domain/Environment Models are validated by the stakeholders. This layer is referred to as the "Domain Description."

Specific-People: Concurrently, a determination must be made for people to understand "what" software product is to be developed in order to realize the system concept. 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 in this layer.

General-Technology: The issues in this layer resolve the "how" the system concept can be represented as a software system. As such, it must describe the overall design of the software system and how it represents the system concept. This layer is called the "System Design;" the technical feasibility of actually implementing the designs it contains can only be validated by the implementers, thus they are the primary audience.

Specific-Technology: For this layer, the implementers reify the System Design into those specialized technology oriented model elements. These software can then be targeted to and validated by the software system's platform technology (such as hardware configuration, operating system, compiler, etc.). This layer is where the actual software representation is accomplished, or the "do" part of development, and hence is given the name "System Implementation."

Traversing Layers Via Functional Decomposition

The model views (i.e. creating related collections of model elements by looking at the system from a particular point of view) contained within each development layer are the result of applying a functional decomposition process to the various model layers as depicted in Figure 8. The functional decomposition

process consists of six basic tasks that serve to capture, analyze, and assemble the system concepts into reified model elements (and ultimately software elements). As an example, the process of conceptualizing within System Analysis, that is, describing the top-level specific-people kinds of abstractions, produces the System Requirements view. Moving down a bit into the Design layer, the view that represents identification of general-technology "form" model elements that stem from the System Requirements (which in were identified previously as conceptualizing in Design) is well represented by traditional Object relationship models.

Figure 9 summarizes this functional decomposition process. It is presented in terms of six numbered tasks, indicating the general progression of the process. But the tasks will necessarily have a good deal of risk-driven concurrency, particularly when large-content, high-risk exposure decisions must be made, such as the choice of COTS components. The degree of elaboration of the intermediate representations will be risk-driven as well.

Figure 9

Figure 9. Functional Decomposition Process

In task 1, a focal point (which serves as a simple an entry point) is chosen which sets the center of context around which this model will be oriented. For example, within the Analysis we seek a specific-people kind focal point which will be called the "Statement of Purpose" for the software system. In task 2, the external influences that may affect the qualities of the abstractions are listed. This often takes the form of goals, as with Product Goals, which come about when describing issues that influence the system with respect to the Analysis but are not strictly part of the system, such as time/cost/quality goals. Note that it is set apart from the process flow indicated in Figure 9. This is to indicate that influences may independently have effects throughout the model. Consider for example, that the Product Goal, "System should be able to handle up to 10,000 new users per year" will very likely effect several of the systems components and behaviors. In task 3, the top-level concepts, such as System Requirements, are generated and used as a reference model to maintain the consistency and soundness of the model elements to follow. This constitutes the top-level "gathering" layer of the process.

We now move to the decomposition (or analysis) layer of the process. In task 4 we identify directly from the concepts derived in task 3, the abstractions that have "form" (i.e. can undergo state transitions), such as System Entities, Components and Objects (when “identification” is applied to Domain Description, System Analysis, and System Design respectively). Concurrently, in task 5, the concepts are decomposed and elaborated on, thereby becoming more specific through refinement. This is often limited to a point appropriate for the particular Product Model being created, for example within System Analysis, a System Requirement may be refined to the point where System Operations are specified, but not so far as to specify the literal functions that carry out the operations. The models created through identification and refinement of the concepts are known as *intermediate product representation* models, as they are never used directly to represent the whole product and are generally not concrete enough to accommodate any such representation. In task 6, the “form” abstractions, along with the concept refinements are merged and reified through engineering operations such as classification, generalization, specialization, factoring, and so forth. This often results in the common technique of classification that is used to formally represent a collection of abstractions related by "type-of" and "kind-of" relationships. Associations were already handled within task 4. Specifically, engineering within the System Analysis results in an Enterprise Model, which describe the general groupings of components within the system domain as sets of component and behavior classification hierarchies.

Integrating Product, Process, Property, and Success Models

In some situations, one can do a great deal to integrate a project’s product, process, property, and success models. The best example is domain-driven model integration, which is feasible for families of projects in relatively mature or bounded domains. The multimedia archive domain discussed in Section 4 is a good example. There, the domain model determined baseline models for the stakeholders’ roles and portions of their success models; and for the application’s concept of operation, system boundary, and environment model. The domain

model also determined baseline values and models for properties such as performance, and determined the project's baseline process model within the LCO-LCA-IOC framework.

Similar determinations can be made from domain models in other mature, bounded domains, such as moderate-size financial planning, inventory control, or data acquisition and analysis applications; or from domain models constructed for software product lines, e.g., the XYB banking domain in [Jacobson et al., 1997].

In other situations, compatible models can be determined from previous compatibility analyses. One example is the process model decision table shown in Figure 10 [Boehm, 1989; pp. 436-37]. It shows the results of a risk analysis of various combinations of domain characteristics (size of application, requirements understanding, available COTS components, architecture understanding) and success model characteristics (required robustness; cost or schedule constraint) to determine which process model is the best match to the characteristics. Thus, a pure waterfall model is not a good choice if the requirements or architecture are not well understood. But if the system needs to be very robust to be successful, a process involving initial spiral cycles to determine appropriate requirements and architecture, followed by a waterfall model with strong requirements verification is a good choice.

Figure 10

Figure 10. Software Process Model Decision Table

Some other examples of model compatibility analyses are the Quality Attribute Requirements Conflict Consultant (QARCC) [Boehm-In, 1996]; the Software Architecture Analysis Method (SAAM) [Kazman et al., 1994]; and Expert COCOMO [Madachy, 1995], which identifies risky combinations of project choices entered as COCOMO cost driver ratings (e.g., proposing to develop an ultra-reliable product with an inexperienced team on a highly compressed schedule).

Beyond this, one can perform some degree of model integration by using model-clash tables such as Table 1 as checklists. We are attempting to elaborate Table 1 into a considerably more comprehensive checklist via analyses of our annual series of digital library projects and other sources of experience.

Ultimately, our objective is to develop a knowledge-based toolset to provide advice on avoiding model clashes. Viewed in this way, the MBASE conceptual modeling challenges are a special case of the general knowledge representation challenge for expert systems.

6. Conclusions

Software projects need to make many tactical decisions, and a much smaller number of strategic decisions. It is easy to get so enmeshed in the daily tactical decisions that one makes incompatible strategic decisions involving model clashes. The MBASE approach provides a framework and a number of techniques for avoiding such model clashes.

The MBASE approach is still only partially elaborated. Two of its key challenges are the choice of and the integration of the various conceptual models needed to represent and reason about a project's success, product, process, and property characteristics. Representations such as Figures 1 and 3 begin to capture model integration characteristics, but clearly fall short of representing the full integration of the relevant models. We welcome collaboration with the conceptual modeling community in determining the most effective choices of such models.

References

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

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

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

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

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

[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.

[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.

[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, 1998]. D. Port, Integrated Systems Development Methodology, Telos Press, 1998 (to appear).

[Rational, 1997]. Rational Objectory Process, Version 4.1, Rational Software Corp., Santa Clara, CA, 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, Unified Software Management, Addison Wesley, 1998 (to appear).

Table 1

	Product Model	Process Model	Property Model	Success Model
Product Model	• Structure clash • Traceability clash • Architecture style clash	• COTS-driven product vs. Waterfall (requirements-driven) process	• Interdependent multiprocessor product vs. linear performance scalability model	• 4GL-based product vs. low development cost and performance scalability
Process Model		• Multi-increment development process vs. single-increment support tools	• Evolutionary development process vs. Rayleigh-curve cost model	• Waterfall process model vs. "I'll know it when I see it" (IKIWISI) prototyping success model
Property Model			• Minimize cost and schedule vs. maximize quality ("Quality is free")	• Fixed-price contract vs. easy-to-change, volatile requirements
Success Model				• Golden Rule vs. stakeholder win-win

Table 2

Milestone Element	Life Cycle Objectives (LCO)	Life Cycle Architecture (LCA)
Definition of Operational Concept	• Top-level system objectives and scope - System boundary - Environment parameters and assumptions - Evolution parameters • Operational concept • Operations and maintenance scenarios and parameters	• Elaboration of system objectives and scope by increment • Elaboration of operational concept by increment
System Prototype(s)	• Organizational life-cycle responsibilities (stakeholders)	•
Definition of System Requirements	• Top-level functions, interfaces, quality attribute levels, including: - Growth vectors - Priorities • Stakeholders' concurrence on essentials	• Elaboration of functions, interfaces, quality attributes by increment - Identification of TBDs (to-be-determined items) • Stakeholders' concurrence on their priority concerns
Definition of System and Software Architecture	• Top-level definition of at least one feasible architecture - Physical and logical elements and relationships - Choices of COTS and reusable software elements • Identification of infeasible architecture options	• Choice of architecture and elaboration by increment - Physical and logical components, connectors, configurations, constraints - COTS, reuse choices - Domain-architecture and architectural style choices • Architecture evolution parameters
Definition of Life-Cycle Plan	• Identification of life-cycle stakeholders - Users, customers, developers, maintainers, interpreters, general public, others • Identification of life-cycle process model - Top-level stages, increments • Top-level WWWWWHH* by stage	• Elaboration of WWWWWHH* for Initial Operational Capability (IOC) - Partial elaboration, identification of key TBDs for later increments
Feasibility Rationale	• Assurance of consistency among elements above - Via analysis, measurement, prototyping, simulation, etc. - Business case analysis for requirements, feasible architectures	• 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

Table 3

Problem Set #2: Photographic Materials in Archives Jean Crampon, Hancock Library of Biology and Oceanography There is a substantial collection of photographs, slides, and films in some of the Library's archival collections. As an example of the type of materials available, I would like to suggest using the archival collections of the Hancock Library of Biology and Oceanography to see if better access could be designed. Material from this collection is used by both scholars on campus and worldwide. Most of the Hancock materials are still under copyright, but the copyright is owned by USC in most cases.
Problem Set #8: Medieval Manuscripts Ruth Wallach, Reference Center, Doheny Memorial Library I am interested in the problem of scanning medieval manuscripts in such a way that a researcher would be able to both read the content, but also study the scribe's hand, special markings, etc. A related issue is that of transmitting such images over the network.
Problem Set #9: Formatting Information Caroline Sisneros, Crocker Business Library Increasingly the government is using the WWW as a tool for dissemination of information. Two much-used sites are the Edgar Database of Corporate Information (http://www.sec.gov/edgarhp.htm) and the Bureau of the Census (http://www.census.gov). Part of the problem is that some of the information (particularly that at the EDGAR site) in only available as ASCII files. For information that is textual in nature, while the files can be cleaned up, formatting of statistical tables is often lost in downloading, e-mailing, or transferring to statistical programs. And while this information is useful for the typical library researcher, who usually have a very distinct information need, the investment in what it would take to put this information is a usable format is often too much trouble.
Problem Set #13: Moving Image Archive Sandra Joy Lee, Moving Image Archive, School of Cinema/TV The USC Moving Image Archive houses USC student film and video productions dating from the1930s to current productions in the School of Cinema-Television. Moving image materials in multiple formats, specialized viewing equipment, limited storage space, and complex access needs create challenges that may be solved with new computer technologies. Fifteen movie clips (.mov format), each approximately 45 minutes in length, over 100 digital film stills (.gif format), and textual descriptions of the films will be made available to students wishing to explore this project.

Figure 1

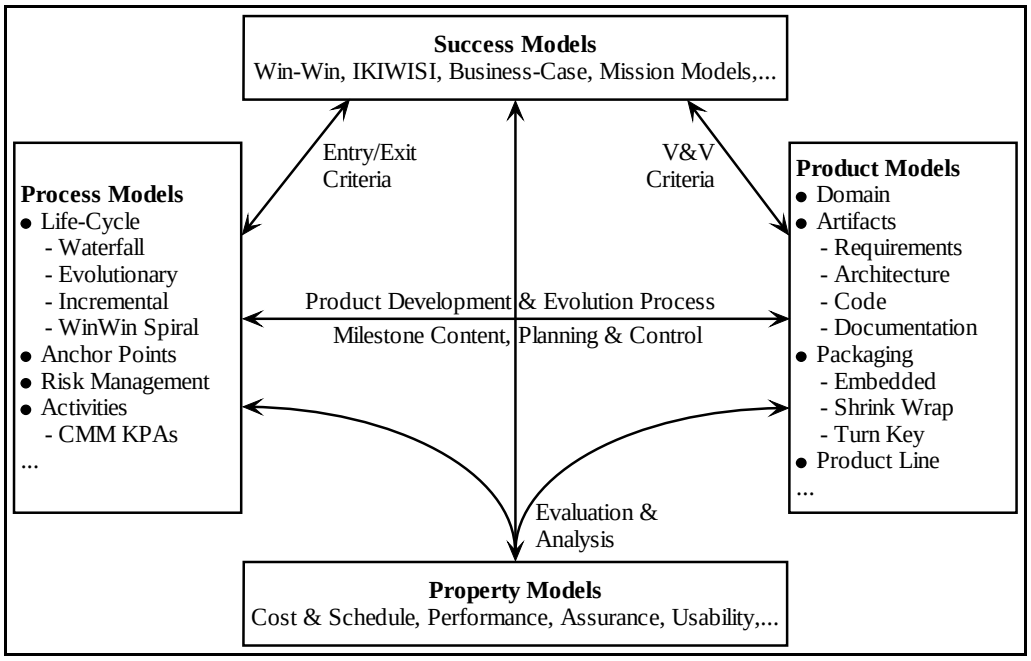


Figure 3 (note out of sequence)

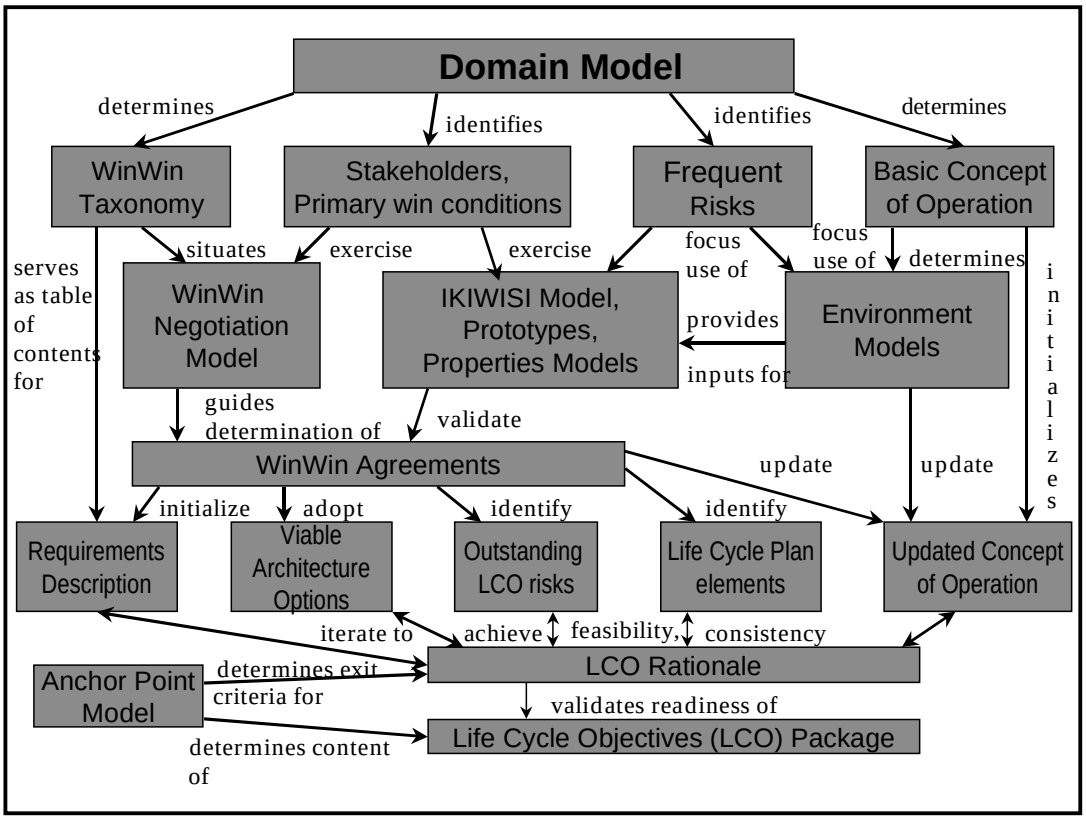


Figure 2 (note out of sequence)

Project Objectives

Create the artifacts necessary to establish a successful life cycle architecture and plan for adding a multimedia access capability to the USC Library Information System. These artifacts are:

1. An Operational Concept Definition
2. A System Requirements Definition
3. A System and Software Architecture Definition
4. A Prototype of Key System Features
5. A Life Cycle Plan
6. A Feasibility Rationale, assuring the consistency and feasibility of items 1-5

Team Structure

Each of the six team members will be responsible for developing the LCO and LCA versions of one of the six project artifacts. In addition, the team member responsible for the Feasibility Rationale will serve as Project Manager with the following primary responsibilities:

1. Ensuring consistency among the team members' artifacts (and documenting this in the Rationale).
2. Leading the team's development of plans for achieving the project results, and ensuring that project performance tracks the plans.

Project Approach

Each team will develop the project artifacts concurrently, using the WinWin Spiral approach defined in the paper "Anchoring the Software Process." There will be two critical project milestones: the Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA) milestones summarized in Table 1.

The LCA package should be sufficiently complete to support development of an Initial Operational Capability (IOC) version of the planned multimedia access capability by a CS577b student team during the Spring 1997 semester. The Life Cycle Plan should establish the appropriate size and structure of such a team.

WinWin User Negotiations

Each team will work with a representative of a community of potential users of the multimedia capability (art, cinema, engineering, business, etc.) to determine that community's most significant multimedia access needs, and to reconcile these needs with a feasible implementation architecture and plan. The teams will accomplish this reconciliation by using the USC WinWin groupware support system for requirements negotiation. This system provides facilities for stakeholders to express their Win Conditions for the system; to define Issues dealing with conflicts among Win Conditions; to support Options for resolving the Issues; and to consummate Agreements to adopt mutually satisfactory (win-win) Options.

There will be three stakeholder roles:

- Developer: The Architecture and Prototype team members will represent developer concerns, such as use of familiar packages, stability of requirements, availability of support tools, and technically challenging approaches.
- Customer: The Plan and Rationale team members will represent customer concerns, such as the need to develop an IOC in one semester, limited budgets for support tools, and low-risk technical approaches.
- User: The Operational Concept and Requirements team members will work with their designated user-community representative to represent user concerns, such as particular multimedia access features, fast response time, friendly user interface, high reliability, and flexibility of requirements.

Major Milestones

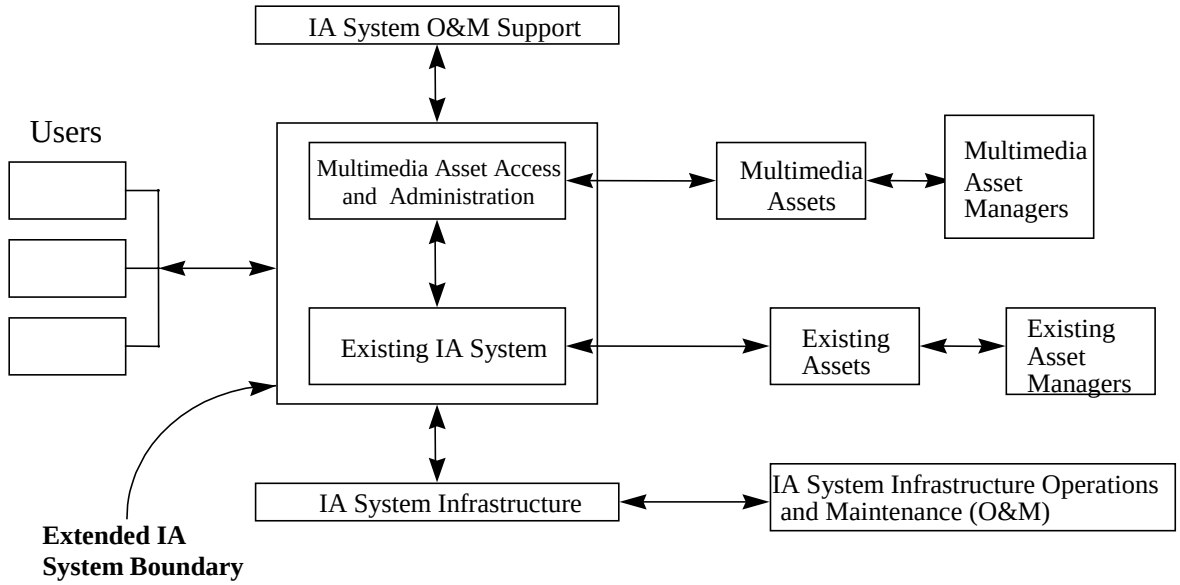
September 16	---	All teams formed
October 14	---	WinWin Negotiation Results
October 21,23	---	LCO Reviews
October 28	---	LCO Package Due
November 4	---	Feedback on LCO Package
December 6	---	LCA Package Due, Individual Critique Due

Individual Project Critique

The project critique is to be done by each individual student. It should be about 3-5 pages, and should answer the question, "If we were to do the project over again, how would we do it better - and how does that relate to the software engineering principles in the course?"

Figure 4

1. **System Block Diagram:**
This diagram shows the usual block diagram for extensions providing access to and administration of multimedia information archive assets from an existing text-based information archive (IA) System:



The system boundary focuses on the automated applications portion of the operation, and excludes such entities as users, operators, maintainers, assets, and infrastructure (campus networks, etc.) as part of the system environment. The diagram abstracts out such capabilities as asset catalogues and direct user access to O&M support and asset managers.

2. **Some Stakeholder Roles and Responsibilities**

2.1 Asset Managers. Furnish and update asset content and catalogue descriptors. Ensure access to assets. Provide accessibility status information. Ensure asset-base recoverability. Support problem analysis, explanation, training, instrumentation, operations analysis.

2.2 Operators. Maintain high level of system performance and availability. Accommodate asset and services growth and change. Protect stakeholder privacy and intellectual property rights. Support problem analysis, explanation, training, instrumentation, operations analysis.

2.3 Users. Obtain training. Access system. Query and browse assets. Import and operate on assets. Establish, populate, update, and access asset-related user files. Comply with system policies. Provide feedback on usage.

2.4 Application Software Maintainer. Perform corrective, adaptive and perfective (tuning, restructuring) maintenance on software. Analyze and support prioritization of proposed changes. Plan, design, develop, and verify selected changes. Support problem analysis, explanation, training, instrumentation, operations analysis.

2.5 Infrastructure service providers (e.g. network, database, or facilities management services).
Similar roles and responsibilities to Asset Managers under 2.1

Figure 5

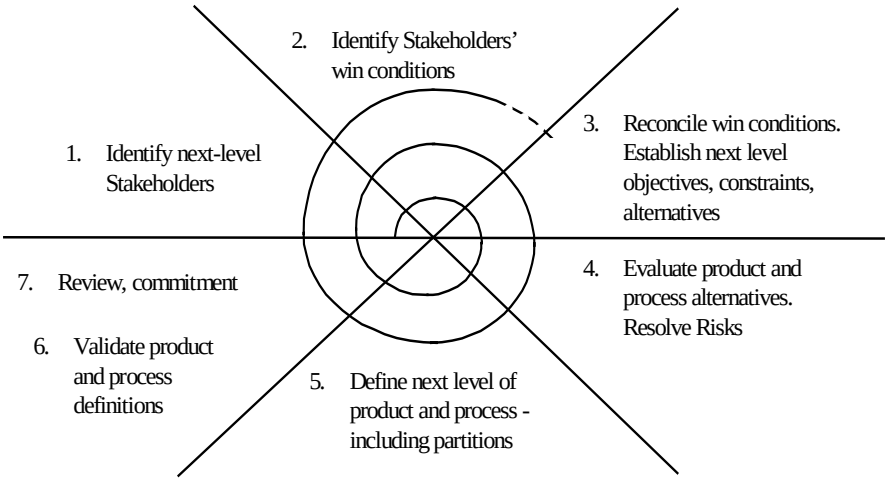

2. Identify Stakeholders' win conditions
3. Reconcile win conditions. Establish next level objectives, constraints, alternatives
1. Identify next-level Stakeholders
4. Evaluate product and process alternatives. Resolve Risks
7. Review, commitment
6. Validate product and process definitions
5. Define next level of product and process - including partitions

Figure 6

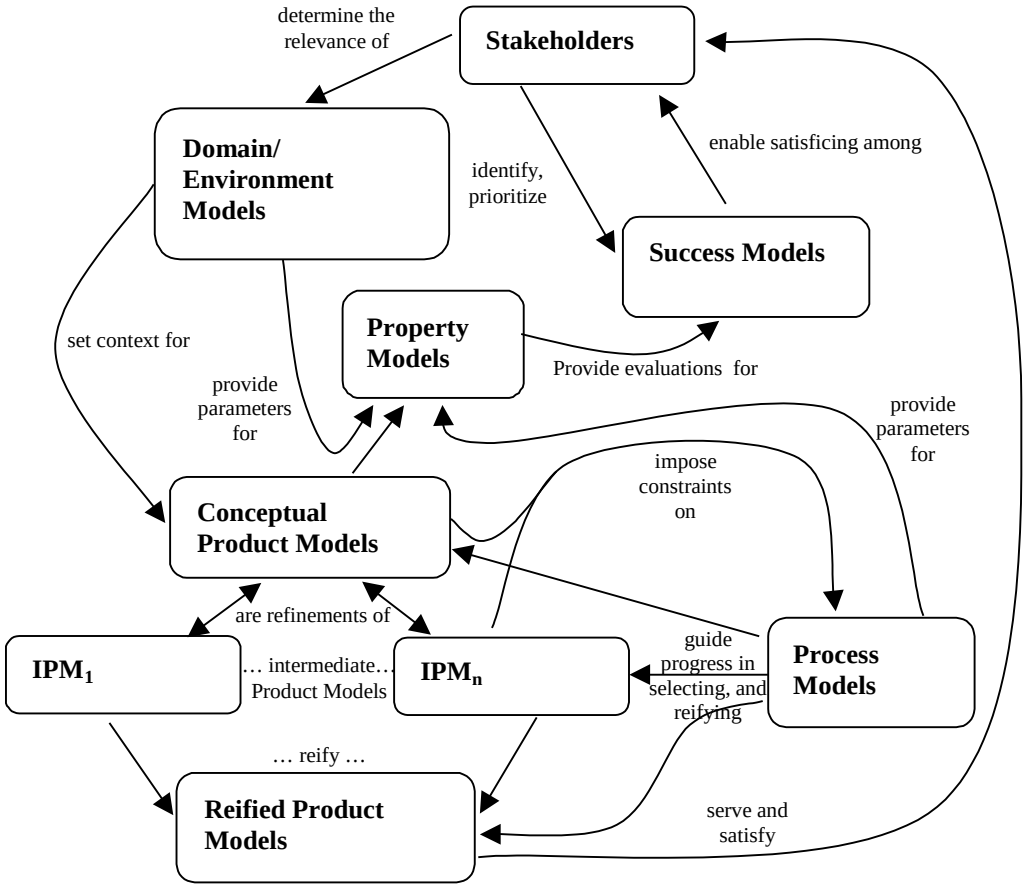

determine the relevance of
Stakeholders
Domain/ Environment Models
identify, prioritize
enable satisficing among
Success Models
Property Models
set context for
provide parameters for
Provide evaluations for
provide parameters for
impose constraints on
Conceptual Product Models
are refinements of
IPM1
… intermediate… Product Models
IPMn
guide progress in selecting, and reifying
Process Models
… reify …
Reified Product Models
serve and satisfy

Figure 7

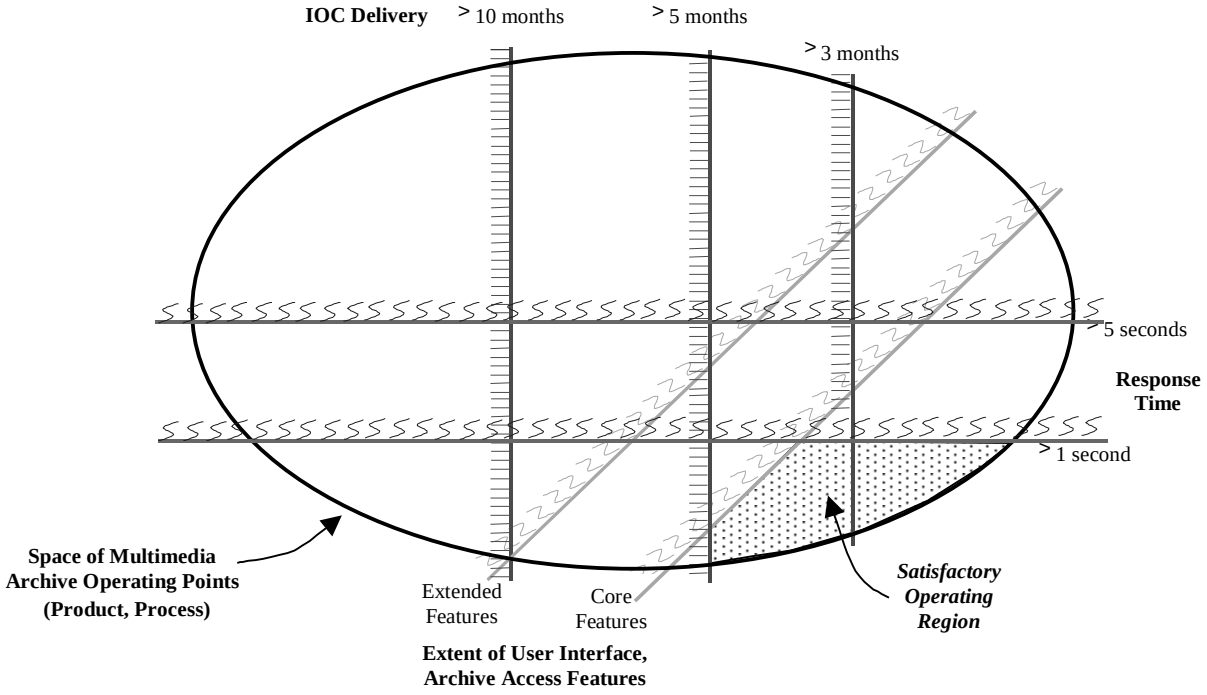


Figure 8

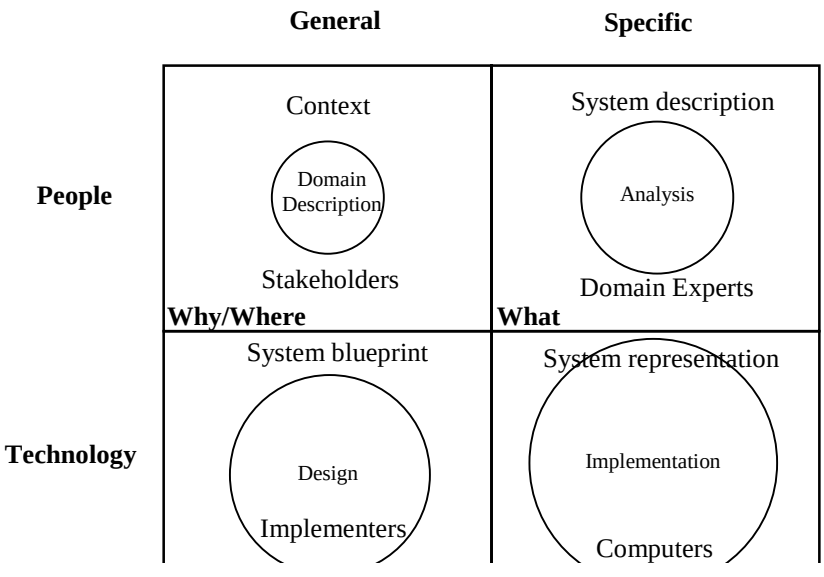


Figure 9

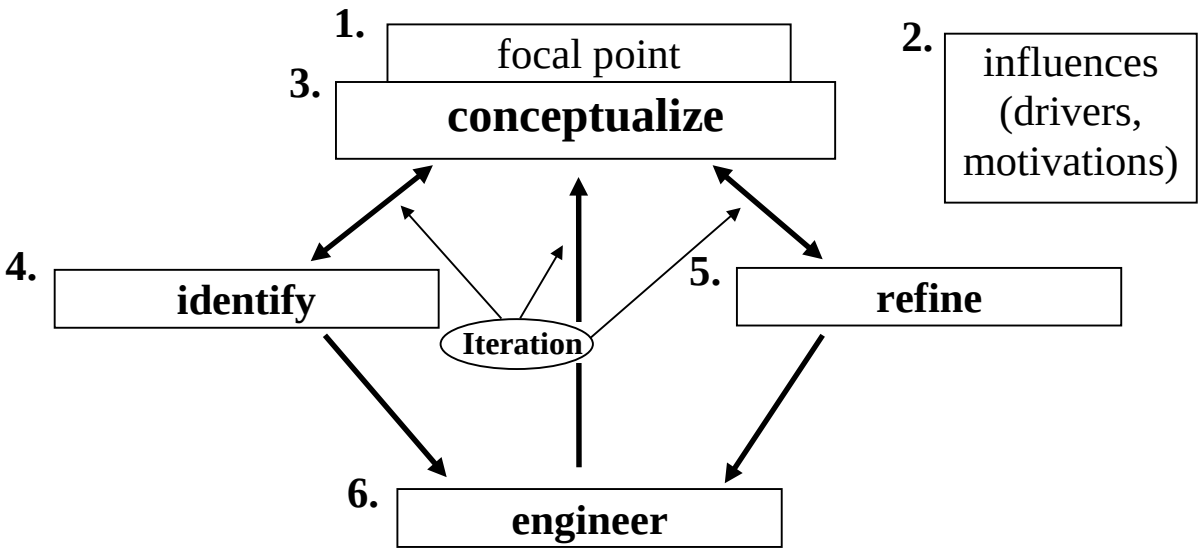

1.
focal point
2.
influences (drivers, motivations)
3.
conceptualize
4.
identify
5.
refine
Iteration
6.
engineer

Figure 10

Objectives, Constraints			Alternatives		Model	Example
Growth Envelope	**Understanding of Rqts.**	**Robustness**	**Available Technology**	**Architecture Understanding**		
Limited			COTS		Buy COTS	Simple Inventory Control
Limited			4GL		Transform or Evolutionary Development	Small Business-DP Application
Limited	Low	Low		Low	Evol. Prototype	Advanced Pattern Recognition
Limited to Large	High	High		High	Waterfall	Rebuild of Old System
	Low	High			Risk Reduction Followed by Waterfall	Complex Situation Assessment
		High		Low		High-performance Avionics
Limited to Medium	Low	Low-Medium		High	Evolutionary Development	New Decision Support System
Limited to Large			Large Reusable Components	Medium to High	Capabilities-to-Requirements	Electronic Publishing
Very Large					Risk Reduction & Waterfall	Air Traffic Control
Medium to Large	Low	Medium	Partial COTS	Low to Medium	Spiral	Software Support Environment

Conditions for Additional Complementary Process Model Options

-*Design-to-cost or schedule*: Fixed Budget or Schedule Available

-*Incremental Development*: Early Capability Needed

(only one condition is sufficient)

Limited Staff or Budget Available

Downstream Requirements Poorly Understood

High-Risk System Nucleus

Large to Very Large Application

Required Phasing With System Increments