

*Sooner or later, the assumptions people make about what a system should do will conflict. Here's how to recognize it before it's too late.*

Barry Boehm and Dan Port

# When Models Collide: Lessons from Software Systems Analysis

The first step in developing either an application or a system is to visualize it. And when you visualize a system, you can't help but use intellectual models to reason about what you're building and how it will behave. The model can be a pattern you follow or an analogy you use. Whatever the form, models are ubiquitous: Developers use them in building a small stand-alone package or a large custom system. Customers use them to visualize what they think they're getting from developers.

Models are very powerful. When you follow a model that makes sense, you get the feeling you're doing everything right. That's why expert programmers/systems analysts feel perfectly comfortable specifying an application that only other programmers or analysts can use.

Models are also deep-rooted. A model can be so natural that you forget you're using it—which is why models are seldom blamed when a project goes wrong. Instead, surface remedies are applied: Requirements are reestablished, managers fired, tools purchased, and standards imposed.

Part of the problem is a lack of awareness training. Few people have the background to recognize what's really going on underneath it all. Very often different stakeholders have unconsciously adopted different assumptions about what they need

and want. Sooner or later, these different models are bound to conflict.

Building, enhancing, and maintaining *any* IT system involves building four basic models: success, product, process, and property. Not one at a time, but concurrently and continuously. When these models collide, it creates confusion, mistrust, frustration, rework, and throwaways. It loses money, and it costs time. Model clashes can leave everyone involved, especially the developer, feeling as if they are slogging through a tar pit. And no surface remedy yet discovered can fix them.

## FOREWARNED IS FOREARMED

What if project managers could instead be trained to recognize model conflict early (before commitments are made) and continuously (as the project and the market evolve)? What if they could revisit their goals and recalibrate their development? What if they could learn to recognize the pattern of how things go wrong in time to react?

Just a few years ago, these goals seemed unrealistic. But systems analysis is rapidly coming of age. Many projects have successfully used a new approach called MBASE (Model-Based System Architecting and Software Engineering). The MBASE approach integrates the four common development models (success, product, process, and property) around the creation and use of a software architecture package. As analysts explicitly integrate models using MBASE, they can recognize and reconcile model clashes as a matter of course, instead of after the fact.

## Inside

**When Projects Succeed But Users Lose**

**Using Milestones to Anchor Your Project**

**Sources and Resources**

## When Projects Succeed But Users Lose

We all know the Golden Rule:

*Do unto others  
As you would have others do unto you*

**But even something as benign as this is dangerous when it becomes an unconscious systems analysis model. Because computer scientists and developers of large, custom systems will usually translate this rule into**

*Develop a system for others  
Assuming they all like to write software  
and know a good deal about computer science*

**The result? User interfaces with a powerful, flexible, but obscure and unforgiving command syntax. Programmers will love it. Mere users—the doctors, financial analysts, or real estate agents who just want to get something done—will consider the application a failure. The cause of tragedies like this is the underlying assumption of the Golden Rule:**

*Everyone is like me*

**As every user knows, no one is like them. No one has exactly the same needs, and no one thinks exactly the same way. Why would everyone need to use an application the same way?**

**The answer? Systems analysts should adopt the Platinum Rule:**

*Do unto others  
As they would be done unto*

MBASE has already demonstrated its practical value. Using an early version of MBASE, TRW brought a million-line command-and-control system in on time (most new IT projects are less than 80K lines of code). Key parts of MBASE are also incorporated into Rational's Unified Process. Rational is a leading provider of software engineering and management tools such as Rose98 and is the major contributor to the Unified Modeling Language (UML), a visual modeling language that is rapidly being adopted. Other organizations like Xerox and the US Federal Aviation Administration are integrating MBASE elements into their software and systems engineering processes.

Meanwhile, some very large and very public failures are motivating more IT organizations to look at something more rigorous than the seat-of-the-pants approach but more fluid than the old-fashioned specify-analyze-design-implement-test-and-deploy approach (aka the waterfall model).

In 1992, AMR Information Services Inc. (AMRIS), the subsidiary that manages American Airlines' SABRE airline reservation system, wrote off \$213 million after the

collapse of the Confirm project it undertook in partnership with Hilton Hotels, Marriott, and Budget Rental Cars. The suits and countersuits persisted for years.

In 1988, Bank of America and several other sponsors threw in the towel after nearly four years and \$80 million trying to build the Master Net trust accounting system. In 1984, BofA gave an uncontested sole-source contract to developer Premier Systems to develop Master Net. Several other banks signed up as sponsors and users. The project started in March 1984, with a budget of \$20 million and an initial delivery date of 31 December 1984. Attempts to use the system in 1986 failed; it was scrapped two years later.

These failures have something in common besides their scale: Both can be attributed in part to clashes among their development models.

### MODELS, PROJECT TYPES, AND FAILURE PATTERNS

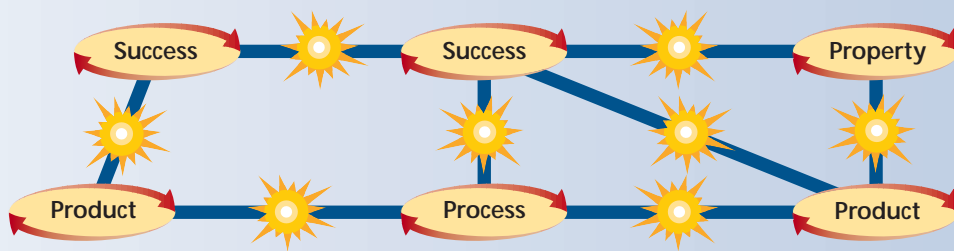
The four basic models (success, product, process, and property) have important interrelationships. Success models are by far the most important. What does it mean for the application to succeed? Does it need to make a certain return on investment? Do all the stakeholders have to "feel" it is a success?

Every set of stakeholders—perhaps every single stakeholder—will have their own idea. The IKIWISI (I'll know it when I see it) success model has become common, and is at the root of prototyping's current popularity. Other success models are imposed by law or business conventions.

The product models describe the system's context, requirements, and architecture. The process models describe how the system will be developed and evolved. The property models describe things like cost, performance, security, or dependability.

All these models represent moving targets and so may move in and out of conflict. When they do clash, however, they create certain kinds of failure patterns (Table 1).

We are also beginning to understand the failure profile for particular types of projects. The examples in Table 1 cover situations in which an entrepreneur hires a development team to build a system for a set of prospective users. Other model-clash patterns cover government acquisitions, such as the failure of the Denver International Airport's baggage-handling system or the London ambulance system. In other words, you can use project stereo-



**Table 1. Success model clashes most often encountered by IT pros.**

Type	Example	Cause	Solution
Product-process	Developers try to freeze requirements and processes, and Users lose interest in the system.	Developers want stability and control. Users want features and the flexibility to change them.	Stakeholders negotiate which requirements must be stable and which can be more flexible.
Product-property	Entrepreneurs and Developers tend to overpromise features and schedules.	Users want many features very quickly. Entrepreneurs and Developers say what they need to say to get their business.	Entrepreneurs and Users prioritize features. Developers organize project so that features can be shed to meet schedule.
Product-product	Developers will tend to use their preferred platform, even if it creates compatibility problems down the road for Users.	Developers want to maximize profit and minimize risk. Users want compatibility with existing systems.	Entrepreneurs consider compatibility in selecting Developers. At the first anchor point, system feasibility is confirmed.
Process-process	Entrepreneurs and Developers won't let Sponsors or Users know what process is being used.	Sponsors and Users might try to overcontrol the project. Developer and Entrepreneur culture may suppress warnings.	Stakeholders agree to a thorough review at the anchor points. Progress reports make visible key progress metrics.
Property-property	Detailed agreements about system cost and performance are made at contract signing that can't be met.	Everyone assumes that system behavior is predictable, when in fact it is not. Developers reuse scalable components to meet Entrepreneurs' cost-schedule targets.	Use anchor points to prototype and verify performance and scalability of system components and configurations.

types, like Entrepreneur or Government Acquisitions, to anticipate and avoid certain kinds of failure patterns.

Every combination of models could clash, but IT professionals are most likely to encounter five basic types.

### SUCCESS-SUCCESS CLASH

Obviously, not all success models are compatible. When they clash, the ripple effect can cause clashes in the other kinds of models. Identifying and addressing success model clashes before they turn into product, process, and property commitments can save you considerable time and money.

**Entrepreneur.** In the Entrepreneur stereotype, for example, the Entrepreneur often obtains funds from a set of Sponsor stakeholders and contracts with Developer stakeholders to provide services to User stakeholders. This situation implies four success models, some of which are in sharp conflict before the project begins.

*The Entrepreneur wants*

- A solid business case with assurances of a strong return on investment.
- A rapid schedule, to capture market share and begin

## Using Milestones to Anchor Your Project

Model clashes must be identified and resolved before they become embedded in business commitments. MBASE establishes three milestones, called anchor points, for the review of project artifacts. If the review verifies that there are no serious model clashes, the stakeholders can decide to proceed. Reviews become more detailed as development progresses.

The first two anchor points are the Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA). The focus of the LCO review is to ensure that at least one architecture choice is viable from a business perspective. The focus of the LCA review is to commit to a single detailed definition of the review artifacts. The project must have either eliminated all significant risks or put in place an acceptable risk-management plan.

At each of these anchor points the key stakeholders review six artifacts: *operational concept description, prototyping results, requirements description, architecture description, life-cycle plan, and feasibility rationale.*

The feasibility rationale covers the key pass/fail question: "If I build this product using the specified architecture and processes, will it support the operational concept, realize the prototyping results, satisfy the requirements, and finish within the budgets and schedules?"

The LCO milestone is the equivalent of getting engaged, and the LCA milestone is the equivalent of getting married. As in life, if you marry your architecture in haste, you and your stakeholders will repent at leisure. The third anchor point milestone, the Initial Operational Capability (IOC), constitutes an even larger commitment: It is the equivalent of having your first child.



- generating revenue quickly.
- Flexibility, to adapt to new market, competition, or technology developments.
- Controllability, to adapt to similar changes after deployment.
- Development visibility and control, to anticipate and adjust to development slippages and shortfalls.

### *The Sponsor wants*

- A solid business case.
- A competitive edge for the sponsor's product line.

- Development visibility and control.

### *The Developer wants*

- To reuse software, tools, and platforms.
- To build what can be reused in the future.
- Profits, generated via development, operations, and maintenance (depending on the agreement).
- Stable requirements.
- Not to be interfered with by micromanaging entrepreneurs.

### *The Users want*

- Affordability.
- The features they need to do what they want to do.
- Compatibility, controllability, and flexibility.
- Ease in converting legacy systems, converting data, and retraining employees.
- The "-ilities" (dependability, usability, performance, security, scalability, and others).

**Domino effect.** When one success model fails, it can take another one down with it. In the classic story of the New Jersey Department of Motor Vehicles project, the customers had two success models: low development cost and good throughput. Unfortunately, although the developers brought the project in on budget, throughput was so poor that at one point more than a million New Jersey automobiles were on the roads with unprocessed license renewals. So no one much cared that the development cost was low.

On another project, an IT shop relied on a requirements-first waterfall model to automate operations. In a hurry to develop detailed requirements, the analysts simply codified models of the existing operations. But this automating-the-existing-system approach, as so often happens, created so much unnecessary input and such obtuse outputs that the system was scrapped after about three years and \$20 million.

## PRODUCT-PRODUCT CLASH

Product-to-product clashes are by far the largest source of interoperability problems and a primary reason the current generation of information systems is so expensive to build and maintain.

In the Entrepreneur stereotype, for example, Users want compatibility with their existing products and controllable product evolution. Sponsors may want to dictate product choices. Developers want to reuse their product assets, or to develop the Entrepreneur's product for reuse to satisfy different customers. Many product model clashes lurk within these success models.

Looking at the Master Net failure, BofA had been a pure IBM shop since 1955. But Premier's assertive project leader, Steven Katz, had a strong relationship with Prime, so he built Master Net on Prime equipment. BofA lost com-

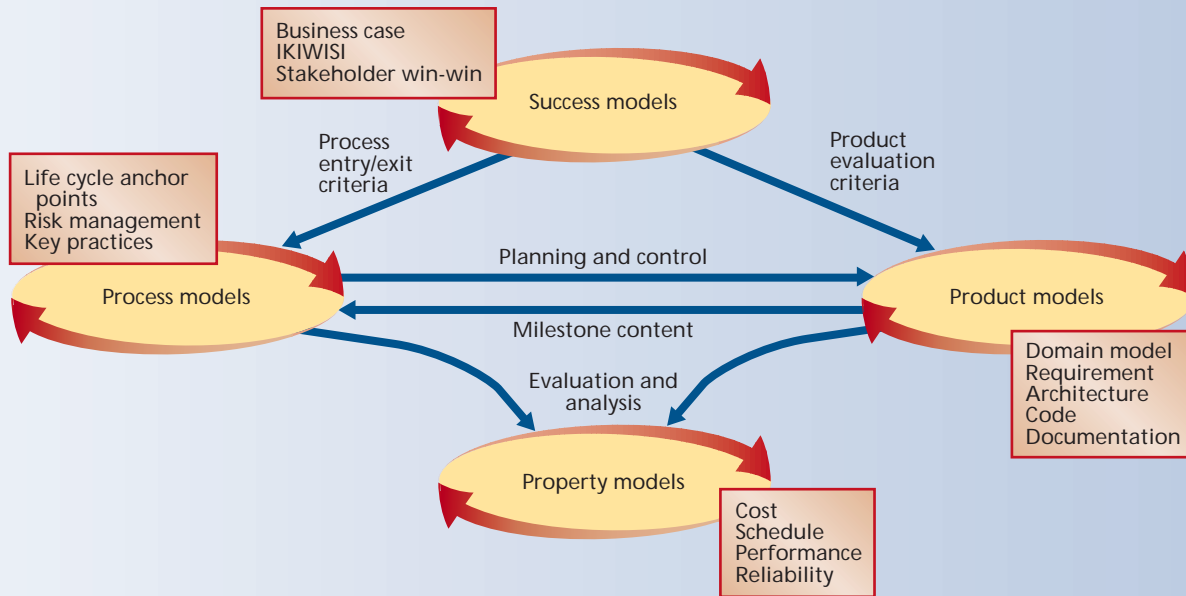
**Figure 1. MBASE: Preventing model collision before it's too late.**

Managers, analysts, and developers who use MBASE should get comfortable with doing more than one part of application definition at a time and revisiting each phase more than once.

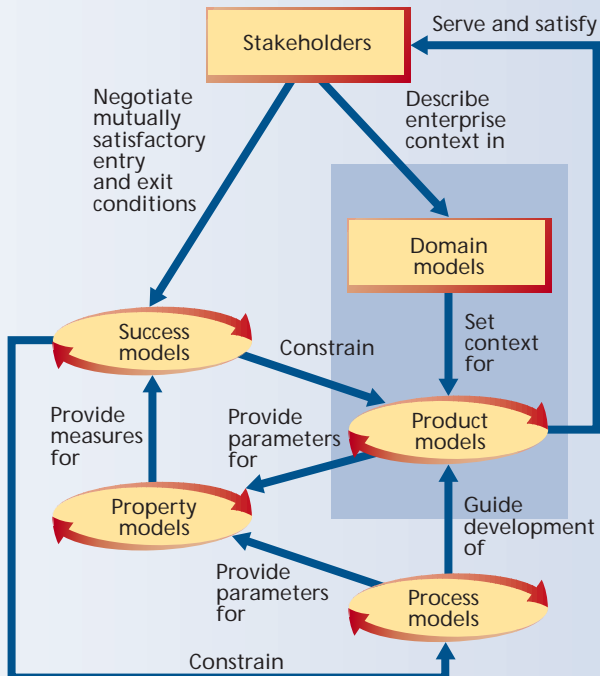
You essentially follow a spiral, looping through the process, checking in with the stakeholders in each phase, and gradually building a stronger and stronger bridge among all the parts.

To use MBASE, it helps to visualize application development as involving three principles: 1. Most projects involve integrating four system models, each one capturing specific interdependencies. 2. All four models have specific interdependencies. 3. The overall process is an iterative spiral that is anchored with management milestones. At each anchor point, key stakeholders decide whether and how to proceed with development.

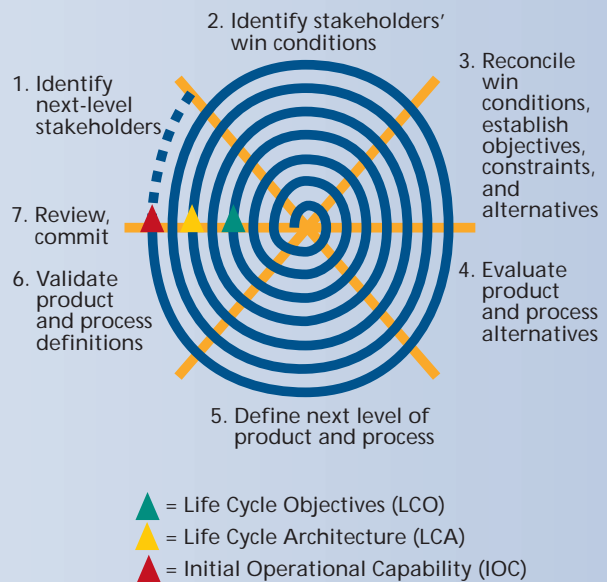
**1. Understand the relationships among models**



**2. Understand how the models interact**



**3. Use anchor points to stay on track**



## Sources and Resources

**Denver International Airport, London ambulance, Master Net, and Confirm project failures:** *Software Failure: Management Failure*, S. Flowers, John Wiley & Sons, 1996; and from *Software Runaways*, R. Glass, Prentice Hall, 1998.

**New Jersey Department of Motor Vehicles failure:** "New Jersey Motorists in Software Jam," C. Babcock, *Computerworld*, Sept. 30, 1985, pp. 1, 6.

**TRW success story:** *Software Project Management: A Unified Framework*, W. Royce, Addison Wesley Longman, 1998.

**How product-product clashes affect the integration of COTS into a legacy system:** "Architectural Mismatch: Why Reuse is So Hard," D. Garlan, R. Allen, and J. Ockerbloom, *IEEE Software*, Nov. 1995, pp. 17-26.

**A more complete accounting of model clash failures:** "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them," B. Boehm and D. Port, *ACM Software Engineering Notes*, Jan. 1999, pp. 36-48.

**How to find stakeholders' win conditions:** "Using the Win-Win Spiral Model: A Case Study," B. Boehm et al., *Computer*, July 1998, pp. 33-44.

**How to use the spiral process model:** "A Collaborative Spiral Process Model Based on Theory W," *Proc. ICSP3*, IEEE Press, 1994.

**How to use life-cycle anchor points:** "Anchoring the Software Process," *IEEE Software*, July 1996, pp. 73-82.

patibility with its other systems and met with serious Prime training, familiarization, and staff motivation problems.

Another kind of product-product clash can occur when you try to introduce commercial off-the-shelf software into an incompatible architecture. The results can be devastating. Researchers attempting to integrate just four incompatible COTS software components found that product model conflicts escalated a two-person, six-month project into a five-person, two-year project.

### PRODUCT-PROCESS CLASH

Conflict can also occur when a developer's model of how the application should be built clashes with what users expect. An obvious example is the clash between the IKI-WISI success model and a development process that assumes someone can actually *know* all the requirements before construction. This is a particular danger for IT environments, where systems affect operations in ways that users find hard to articulate, and analysts find hard to predict.

A serious product-process clash can also occur between

the use of off-the-shelf software and the legal process. For example, one contract we know of required a response time of less than one second. When the technical staff determined that no commercial database could deliver this, the lawyers took over. Months and thousands of dollars later, the requirement was finally abandoned.

Another particularly dangerous product-process clash occurs when an Entrepreneur issues a fixed-price or fixed-schedule contract and then proceeds to frequently change or "reinterpret" the requirements. On the other side of this clash is the Developer, who needs stable requirements for predictability and control.

For example, in the Confirm failure, the suit against Sponsors/Users Hilton, Marriott, and Budget by Entrepreneur/Developer AMRIS/Intrico claimed that they were unable to define their requirements by the 1989 deadline and continuously pushed for changes thereafter. The countersuits claimed that AMRIS and Intrico never understood the requirements and were inflexible when they were told what would make them satisfactory.

In the Master Net case, Katz dominated the system definition. He reportedly dismissed BofA's suggestions with such proclamations as, "Don't give us the solutions. Just tell us the problems." BofA reportedly undercut its own project manager, Clyde Claus, by reorganizing the system engineering group.

### PRODUCT-PROPERTY CLASH

System properties include cost, performance, and dependability. A product-property clash occurs when developers assume that tweaking the product in a certain way will generate a predictable result. Some early designers of satellite control systems assumed, for example, that they could always increase application speed by adding processors. Unfortunately, it didn't work. Data dependencies, control dependencies, and resource contention problems actually decreased throughput beyond a certain number of processors. It took a tremendous amount of rework to rearchitect and rebuild these systems.

Entrepreneurs and Users want many features and want them quickly. Developers in a competitive source selection may make overly optimistic delivery promises without a way to achieve them. In February 1991, AMRIS presented a revised plan under which only Hilton's requirements would be delivered on schedule. Marriott prepared to drop out, but AMRIS continued to bill Marriott.

If the User's property needs (dependability, performance, usability, and so on) are not explicitly emphasized, they can easily be sacrificed: A developer can elect to meet the Entrepreneur's ambitious cost and schedule objectives by reusing software that does not have the properties the User wants. AMRIS/Intrico reused several successful portions of SABRE—including IBM's MVS and Transaction Processing Facility operating systems, IBM's DB2 relational DBMS, and Texas Instrument's IEF CASE tool—

to help meet the ambitious schedule. Unfortunately, the bridge among these components was extremely slow and could not restart after a system crash.

Katz's successes with Prime were on relatively small banking systems. The Prime equipment and software did not scale up to Master Net's needs: A single 1-MIP Prime processor was originally specified; by 1987, three 8-MIP Prime processors were being used, but high paging rates caused poor response time. In the major cutover to Master Net, more than a dozen Prime disk drives failed, causing days-long system crashes. Often there is either too much or (as in this case) too little oversight of the development process.

### PROCESS-PROCESS CLASH

Excesses in either direction create serious model clashes here—from developers concealing shortfalls and overruns to entrepreneurs and sponsors too frequently “pulling up the developers by the roots to see how they’re growing.” At AMRIS/Intrico, the culture was to avoid giving bad news to superiors. Thus no high-level managers had a clear view of the problem. An AMRIS vice president reportedly buried a 1991 consultant report clarifying the problems and dismissed the consultant.

BofA's staff was further disadvantaged by a neglect of new-systems development in the 1970s. BofA also did not engage a strong technical systems engineering contractor to counterbalance Katz. This left BofA outgunned by Katz in any requirements or technology discussions.

### WHAT TO DO?

In virtually every case, you can eliminate or mitigate model clashes by concurrently evolving the requirements and other key choices (such as architecture and operational concepts) and through the judicious use of integrated models. That's the philosophy behind MBASE (Figure 1).

MBASE guides you in using inductive, constructive ways to reconcile stakeholders' success models and continuously explore alternatives. It helps you concurrently develop a system's operational concept, requirements, architecture, and life-cycle plans. And it gives you anchor point milestones to stay on track. This kind of approach is designed to keep the success, product, process, and property models consistent and well integrated.

Managers, analysts, and developers who use MBASE should get comfortable with doing more than one part of the process at a time and revisiting each phase to make adjustments as needed. You essentially follow a spiral, looping through the development process, checking in with the stakeholders in each phase, and gradually building a stronger and stronger bridge among all the parts. This contrasts to more traditional approaches that try to bridge models sequentially. Such bridges are often brittle and typically assume an unmanageable amount of preconception

and planning. Even the best design and planning is inadequate because no one can design for unanticipated yet critical requirements.

At critical management decision points, the MBASE anchor points offer an opportunity to adjust the models and rethink decisions. The MBASE milestones have been incorporated in Rational's Unified Management Process, which is used in conjunction with Rational's Requisite Pro and Rose98, and other development tools.

## Product-product model clashes are a primary cause of the high development and maintenance costs associated with today's information systems.

---

MBASE is constantly being refined and expanded as part of a fundamental research initiative at the University of Southern California's Center for Software Engineering and is free to the public. It is tested and improved every year by the rapid architecting of 15 to 20 digital library products, of which six to eight of the best go on to be developed and deployed.

### COMMON-SENSE SCOPING

As you loop through the development process, you will need to scope each cycle. One of your best tools here is common sense. Remember that success models are key. Begin by identifying the key stakeholders. These generally include 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). These stakeholders' critical interests should determine the priorities, desired levels, and acceptable levels of your system's success criteria. These, in turn, will determine which parts of an application's domain and its environment are relevant.

For example, if the top-priority success model is “Demonstrate a competitive, agent-based, electronic commerce system on the floor of Comdex in nine months,” don't expect to have provably correct code or a fully documented system. Your product model should have a hierarchy of features so that you can eliminate lower priorities to make the demo deadline. The process model will be “Design to schedule,” and the property model will focus on portability with just enough reliability to get through the demonstration.

An IKIWISI success model would dictate a prototyping process and leave most of the product and property mod-

els undefined at first. If you want to build a product line, you would start by defining a fairly rigorous domain architecture. You would then model the process and properties to make a business case for the breadth of the line and how it should be rolled out.

A key operational-concept objective is to determine the system boundary; everything inside is the system you propose to develop, and everything outside is the environment. For the Comdex demo, you would set the boundary largely by the schedule. Even if you need credit-card verification at some later point, this feature is currently outside the boundary of the proposed system—the demo—and so is tagged as an evolutionary requirement.

A loop or spiral cycle is considered complete according to what is defined in the life-cycle plan. For the Comdex demo, a loop would constitute the core minimum set of features; other features could be added in subsequent loops until Comdex begins. The boundary is rubber, and it is stretched or compressed as appropriate until time runs out.

The domain scope for the demo system would be very much determined by the COTS products you can integrate. Determining the appropriate combination of COTS products and extensions could take several iterative cycles of experimental prototyping, as well as cost-schedule modeling to determine how much capability you could realistically develop in nine months.

To make sure the success models can be achieved, you need some kind of validation. For the demo system, it makes sense to use a mix of expert judgment and a cost-schedule estimation model to ensure that the product, process, and success models are consistent.

**W**e have found, after analyzing many projects, that the model-clash concept helps considerably in understanding the sources of stickiness in the software project tar pit. The MBASE approach helps in both diagnosing and avoiding model clashes that would otherwise have seriously compromised the project. And we have found in using the MBASE approach on more than 50 projects that it is effective for the rapid architecting and development of leading-edge IT applications. We continue to find additional model clashes that point to future MBASE refinements. You can access three years' worth of projects at <http://sunset.usc.edu/classes/classes.html>. ■

**Barry Boehm** is director of USC's Center for Software Engineering. Contact him at [boehm@sunset.usc.edu](mailto:boehm@sunset.usc.edu).

**Dan Port** is an assistant professor at USC's Center for Software Engineering. Contact him at [dport@sunset.usc.edu](mailto:dport@sunset.usc.edu).

## Read What the Pros Read



Our members receive *Computer*, a monthly magazine for all computing professionals. And they subscribe to 11 specialty magazines.

*IEEE Internet Computing* offers a unique perspective on Internet technology.

Subscribe @

<http://computer.org/subscribe/index.htm>



Did you know?

Members can get an online subscription to 17 Computer Society titles for just \$99.

IEEE Computer Society  
Publisher of

