Introduction to MBASE
(Model-Based (System) Architecting and Software Engineering)

DAVID KLAPPHOLZ

Department of Computer Science
Stevens Institute of Technology
USA
d.klappholz@worldnet.att.net

DANIEL PORT

Information Technology Management
University of Hawaii, Manoa
USA
dport@hawaii.edu

Abstract
Even when sophisticated software development process is used, e.g., the Stakeholder Win Win Risk-Driven Spiral Process, software development projects often fail. A prime reason for this failure is unresolved, often undetected, differences among stakeholders’ sets of assumptions—the “models”—of various aspects of the project. MBASE is an approach to identifying “model clashes” so that their risks can be dealt with. Although it was originally developed as an extension to plan-driven development processes like the Stakeholder Win Win Risk-Driven Spiral Process, it can be adapted for use with any type of development process from Agile to Plan-driven. We discuss the notion of a “model,” the various types of “model clashes” and their consequences, and the MBASE approach to dealing with them.

1. Introduction

1.1. Software Engineering Models

1.2. Model Clashes

2. Dealing With Model Clashes

2.1. Retrospective Approaches
1. Introduction

MBASE [14,15] is a framework that can be used with any type of software development process, from Agile to Plan-driven [16], to deal with “model clashes,” a type of problem that has sabotaged many development projects and that had not, to the authors’ knowledge, been dealt with systematically before MBASE’s introduction in 1998 [14].

In Section 1.1 we discuss the notion of models in software development, and, in Section 1.2 model clashes, and their potentially severe consequences. In Section 2, we discuss retrospective methods for dealing with model clashes, show that their use is infeasible, except in the case of very small projects, and introduce the notion of dealing with model clashes during model construction—the foundation of MBASE. Section 3 defines and elaborates the MBASE approach and in Section 4 we discuss MBASE and experiences in its application.

1.1 Software Engineering Models

Stakeholders in a software development project typically include customers, end users, developers, and maintainers. Additional stakeholders may include marketers, venture capitalists, proprietors of collaboratively-developed interoperating products, and others, specific ones depending upon the project’s specific circumstances. In the case of safety-critical systems such as automobiles and medical systems, the general public are also stakeholders.

When an individual becomes a stakeholder in a new or ongoing software development project, s/he usually approaches it with some assumptions about how it will/should proceed and what its outcome will/should be. Even a customer who has
INTRODUCTION TO MBASE

never before participated in a software development project comes to it with assump-
– at the very least, assumptions about the organizational objective for engaging in
the project, and the benefits to be realized;
– possibly, assumptions about the cost of developing the software;
– possibly, some idea—perhaps incorrect—about the way customers and devel-
opers interact during a development project;
– perhaps, assumptions—often erroneous—concerning what developers know
about the way organizations involved in the customer’s line of business oper-
ate.

The naive customer is also likely to be lacking assumptions that might avoid later
disappointment, e.g., assumptions about the need for maintenance and its cost, or, to
put it differently, the naive customer is likely to tacitly assume that the only cost will
be that of developing the software product.

Developers coming to a new project are likely to have assumptions about: which
software development process will be used, say Waterfall, because their computer
science education emphasized it; their demands on customer and end-user time dur-
ing the course of the project; the amount of cost and schedule slippage that the cus-
tomer should be willing to tolerate; etc.

Different stakeholders may have assumptions on related issues; some of their as-
sumptions on a given issue may be consistent with one another, and some may not.
Different types of stakeholder are also likely to have assumptions on different sets
of issues. In what follows we will term a set of assumptions on a specific issue a
“model.”

While stakeholders often come to a development project with a some general,
most of the detailed models involved in a project are developed as the project pro-
gresses. The developer’s models include representations that s/he has created of vari-
ous aspects of the ultimate product and of the process whereby it will be developed.
These models typically include such artifacts as UML diagrams, Entity-Relationship
Diagrams, etc. They also include such artifacts as Gantt charts, Pert charts, and
cost/effort/schedule estimates. The ultimate representation/model is the software it-
self.

An important property of those models is that they are likely to change, some only
slightly, but some drastically, as a project progresses, for example, as prototypes
cause end users and/or customers to realize that developers aren’t envisioning details
of operation in the “right” way, as business cases clarify the relative importance of
different features, etc.

In a project that uses a Plan-driven development process, developers document
models, not just their own, but, also those of other stakeholders, in writing. In large
development organizations or in organizations subject to industry or government standards, the precise titles and the nature of the content of each of the mandated documents is strictly specified, either by company policy or by the relevant industry or government standards. In an agile project, models are, naturally, developed, and are discussed by relevant stakeholders, but far less emphasis is placed on writing them down, or on preserving those that are; rather, a document is produced and preserved only when its continuing value to the project or users can be demonstrated.

Regardless of the process used in a development project and regardless of whether that process requires that models be discussed only verbally or written down to either loose or tight specifications, it is clear that the developers must create models in the following product- and process-related areas:

– Operational Concept: product’s objectives and relation to organization objectives; product’s scope; people and other software with which the product will interact; operational scenarios; perhaps, the direction in which the objectives are likely to evolve over time.
– Requirements: product’s detailed functions, perhaps prioritized; details of product’s interfaces; necessary quality attribute (reliability, availability, maintainability, etc.) levels.
– Architecture/Design: logical and physical elements/components that will constitute the product, and their relationships.
– Development Plan: the development process to be used; schedules and milestones; individual developers’ responsibilities.
– Business Case: project rationale, including costs, value-added and return on investment estimates; process rationale, requirements satisfaction; project risk assessment.

It is easy to see that a project runs most smoothly when the following is true of all stakeholders’ models:

– Each model is internally consistent. That is, in no model should there be a set of assumptions that contradict one another, nor should there be a set of assumptions, whose logical consequences contradict one another;
– An individual stakeholder’s models—on different issues—are consistent with one another;
– Different stakeholders’ models on the same aspect of the product/project are consistent with one another.

Plan-driven software development processes \[16,27\] typically require inter-document (traceability) references that can be viewed as a small first step toward verifying certain aspects of consistency; some \[30,28\] require that a document, with
1.1.1 Model Types

Recall that earlier in Section 1.1 we have defined a “model” to be a set of assumptions on a specific issue held by a stakeholder of a software development process. The developers of MBASE have found it useful to classify models used in software development projects into four categories: success models, product models, process models, and property models. The following are definitions of the four model types together with examples of possible members of each:

- A success model consists of assumptions that must be satisfied in order for the stakeholder to consider the product and project to be a success. Examples of success model assumptions, each annotated by the type of stakeholder who might have it in his/her success model, are:
  - the product shall pay for itself within eighteen months of installation (customer);
  - the product, or, at least, a significant set of end-to-end core capabilities, shall be running in time for the product to be shown at the next Comdex show (marketing director);
  - I won’t know what the system’s precise behavior should be until I see a prototype that looks right to me (end user);
  - I will get a 5% share of profits from all sales of the product over $100,000 (developer);
  - the DBMS used in the product will be either Oracle or Sybase (maintainer).

- A product model is a set of assumptions about the details of the product, e.g.: the product’s operational concept; the product’s scope; the product’s environment, i.e., people and other hardware/software with which the product will/should interface; the product’s requirements, the product’s architecture/design; etc. Examples of product model assumptions, each annotated by the type of stakeholder who might have it in one of his/her product models, are:
  - The product will be a web site that enables customers from around the world to buy books from Smith’s Fine & Rare Book Company (customer and developer);
  - Records of books for sale and sold will be maintained by a COTS database management system, i.e., the DBMS design and implementation is not part of the present project (developer and customer);
  - Because of security concerns, payment for books ordered will be done via an online credit card company rather than directly by the product (customer);
– The product’s behavior will be as described in the set of use case diagrams in
the design document (developer);
– The DBMS used in the product will be either Oracle or Sybase (maintainer).

A process model is a set of assumptions about the way the product will be
developed and put into operation, i.e., about the way the project will be run/managed. Examples of process model assumptions, each annotated by the type of stakeholder
who might have it in his/her process model, are:

– The product will be developed using Extreme Programming [17] (developer);
– The product will be developed using a Risk-Driven Spiral Process [7] (developer);
– A demonstration must be ready in nine months, so the product will be developed
using a Schedule As Independent Variable (SAIV [9]) process (developer and marketing manager);
– The requirements won’t be fixed until I see a prototype that’s easy to use (end-user);
– Anticipated changes in government regulations make it necessary that require-
ments remain somewhat flexible until next January (customer);
– The manual system will run in parallel with the new product for three months
before a full-cutover decision is made (operations manager).

A property model is a set of assumptions about such aspects of the product
or process as cost, schedule, performance, and “ilities” such as reliability, portability, extensibility, maintainability, security, etc., i.e., properties whose extents can
be placed on either absolute or relative scales. Examples of property model assumptions, each annotated by the type of stakeholder who might have it in his/her property
model, are:

– The product shall be developed, in three months (customer);
– The product shall be developed at a cost not to exceed $500,000 (developer);
– On average, the product shall not fail more than once every two years, and shall
take no longer than four hours to bring back up (maintainer);
– The product shall be designed and implemented in such way as to be able to
be re-worked to handle any change in internet sales tax in no longer than a
week (customer);
– The product shall be able to support at least 100 concurrent users (customer).

When a project starts, most stakeholder assumptions are members of the stake-
holders’ success models, and, in fact, some stakeholders may initially have no mod-
els other than success models. As a project proceeds, most stakeholders mentally create and populate models of the other three types—product, project, and property. Some of a stakeholder’s original success model assumptions may gain (additional) membership in one or more of that stakeholder’s other models. An assumption can, thus, be a member of multiple different models. For example, as is suggested above, a nine-month delivery constraint or a $2,000,000 budget constraint might start as assumptions in a customer’s success model, and may later be incorporated into developers’ property models. Elaborations on success model assumptions and/or on ways of achieving them also often become parts of other-than-success models.

In a software development project, some stakeholders’ initial models are revealed during early project meetings. Particular assumptions or entire models are often revealed to their holder, who didn’t previously realize that s/he had assumptions on various topics, at the same time as they are revealed to other stakeholders. Often, inconsistencies, or “clashes,” between or among stakeholder models, are uncovered and are discussed at project meetings; some are resolved and some may be temporarily tabled, to be revisited later. As a project proceeds, dropped assumptions, changed assumptions, and new assumptions are often recognized and discussed, and new inconsistencies are often revealed, and either resolved or tabled.

Product-product model clashes, being the most obvious, have been recognized and discussed [19] in the past; with few exceptions, however, most other types of clash (product-process, product-property, process-property, process-process, etc.) received little or no attention before the introduction of MBASE [14].

1.1.2 Common Models

The following table illustrates common clusters of generic success model assumptions of the four major types of stakeholder:

- **Users:**
  - The product shall include a sizeable number of features.
  - I want to be free to require additional features as I think of them over the course of the project.
  - Requirements shall remain changeable for as long as possible into the project.
  - The product shall be compatible with as many as possible, preferably all, of the products that I currently use.
  - I want to be involved in the decision as to whether the new software product is needed, in who should be engaged to do the development, and, should there be go/no-go decisions along the way, in those decisions as well.
– The development contract shall be flexible so that changes to aspects of both
the project and the product that affect me can be made as late as I want.
– The product shall be ready at a very early date.

• Maintainers:
– Transition of the product into production use shall require a minimum amount
of my, and my staff’s, time.
– Maintenance of the product shall require a minimum amount of my, and my
staff’s, time—by being developed using languages and COTS products with
which I am familiar.
– The product shall be compatible with most or all of the products that I already
maintain.
– I want to be involved in the decision as to whether a new software product is
needed, in who should be engaged to do the development, and, should there be
go/no-go decisions along the way, in those decisions as well.

• Acquirers (Customers):
– The product shall produce a significant return on investment and/or other sig-
nificant benefits related to my organization’s business objectives (and my own
personal objectives).
– The product shall be built at a low cost and shall be ready at an early date.
– The product shall comply with all potentially relevant government standards.
– The product’s GUI shall not be perceived to be offensive to people of either
gender or to members of any ethnic group.
– I shall get credit within my organization for initiating the project and shall have
ultimate control over its course.
– The development contract shall tie down the developers to the greatest extent
possible (and shall tie me down to the least extent possible).

• Developers:
– The development contract should tie me down to the least extent possible.
– The budget and schedule should have sufficient slack for me to be able to finish
the job regardless of what problems I run into; furthermore both budget and
schedule should always remain negotiable as the project proceeds.
– The requirements should be fixed as early as possible, and should change as
little as possible.
The development team should be free to choose any development process we like, and should be free to program the language(s) that we prefer.

The development team will be formed by common agreement of potential members—and its members will decide on later additions, should any become necessary.

The development team should be free to choose which COTS products will be incorporated into the product.

Less obvious models used by software developers, but ones that can have critical effects on software development projects, are the sets of assumptions—and their consequences—underlying:

- each software development process, e.g., RUP, XP, SCRUM, other agile, RAD, JAD, Risk-Driven Spiral, etc.;
- each complexity/cost/schedule estimation method, e.g., function points, feature points, COCOMOII [8], etc.;
- software analysis, design and implementation methods such as OO;
- analysis and design notations such as the various types of UML diagram; etc.;
- choices of equipment, infrastructure software, programming languages, etc.

As an example, the following constitute a simple version of the key assumptions underlying the Waterfall model:

1. Participants can determine all requirements in advance of implementation.
2. Requirements have no unresolved, high-risk implications such as
   - risks associated with commercial-off-the-shelf software choices;
   - consequences of cost, schedule, performance, security, user interface, and organizational issues.
3. Participants understand the right architecture for implementing the requirements.
4. Stated requirements match the expectations of all the system’s key stakeholders.
5. The requirements’ nature will change little during development and evolution.
6. Deadlines allow enough calendar time to proceed sequentially through design, implementation, testing, and deployment.

Knowing what model assumptions are being made is often vital a vital element in managing or avoiding potentially risky situations. For example, assumption (1) stipulates that participants can determine requirements in advance of implementation. Project staff that believes this or that collect progress payments by delivering
A complete requirements specification under a Waterfall-Process contract may formalize user-interface formats and behaviors as ironclad requirements specifications, and then build the system to those specifications. This can easily result in a delivered system with a GUI that fails the user’s IKIWISI success model test. Inevitably, the client either rejects the system or demands a major rework effort that is both expensive and time-consuming. Assumptions (2), (3), and (4) frequently conflict with property-model assumptions if a software acquisition hastily locks the project into unrealizable property-level contract requirements. We now take a closer look at how model assumptions may conflict and their potential consequences.

1.2 Model Clashes

A model clash is an inconsistency between/among different models of a single stakeholder or between/among same or different models of different stakeholders. If one model belonging to a single stakeholder contains inconsistent assumptions, then, formally speaking, that model is “unsound,” i.e., it can’t properly be called “a model.” In the present discussion we will ignore this fine point and will use the term “model clash” to include the situation in which a single “model” is unsound as well as the case in which two different models are inconsistent with one another.

The simplest type of model-clash creating inconsistency is between an assumption and its negation, e.g., between one developer’s process model assumption of “a waterfall process will be used to develop the product” and another developer’s process model assumption of “a waterfall process will not be used to develop the product.” Some model clashes result from assumptions whose contradiction is slightly less immediate, but which any literate person would catch, e.g., between a customer’s property model assumption that “the product will be delivered in six months” and a developer’s property model assumption that “the product will be delivered in twelve months.”

Many model clashes, though, exist between assumptions that only an experienced software developer, or software development customer, would understand to have contradictory consequences, e.g.,

- between a customer’s success model assumption that “requirements will have to remain somewhat flexible until well into the product’s development” and a developer’s process model assumption that “a Waterfall Process will be used to develop the product,”

1 In modern software development projects, the customer and users often don’t know precisely what precise functionality and precise interface they want/need until they see it in a prototype that is developed iteratively with their participation; hence the term IKIWISI: I’ll Know It When I See It.
INTRODUCTION TO MBASE

• between a developer’s process model assumption that “Extreme Programming will be used to develop the project” and a customer’s success model assumption that “we are so understaffed, that we will meet with developers during the project’s first week, and will not meet with developers again until the product is ready to be installed.”

• between:
  – a development manager’s process model assumption of “the product has to be ready by Comdex or we’ll lose too much market share to stay afloat, so we’ll work on all the product’s features and I’ll move additional developers from some other project into this one in the last two weeks if it doesn’t look like we’ll make it,”
  – and the customer’s success model assumption of “better a reduced set of end-to-end core capabilities by Comdex than a complete product delivered the day after Comdex.”

How do model clashes occur? There are many situations that lead to model clashes. Many of them are completely “innocent” in that stakeholders involved are following accepted (or perhaps even mandated) practice. Some common model clash situations that may lead to serious project risk are when:

• An inexperienced stakeholder doesn’t realize that two of his/her assumptions are inconsistent, e.g., the process model assumption that a Waterfall Process will be used and the success model or product model assumption that product will be largely or completely constructed from COTS products.

• One or more stakeholders change their assumptions on a particular issue over time so that previously consistent assumptions become inconsistent; e.g., stakeholders agree that requirements should be fixed, and, subsequently, one realizes that there are critical requirements that must be changed.

• An apparently reasonable assumption turns out to be false, e.g., a developer’s property model contains an assumption that a particular COTS product will perform at a given level, and testing proves him/her wrong.

• A stakeholder accidentally forgets to articulate a critical assumption; e.g., the customer forgets to indicate that the product’s financial success depends on its availability on multiple platforms, and the developer assumes that the product need run on only one—the most popular—platform.

• A stakeholder doesn’t bother to articulate a critical assumption because “everyone knows that . . .”; e.g., a customer neglects to describe a project- or product-critical aspect of the way his/her business operates, and the developers, having no previous experience in the customer’s field, aren’t among the “everyone” who knows that the customer’s business operates that way.
1.2.1 Model Clash Patterns

Figure 1 shows common model clashes that occur between product, process, and property models of the common clusters of generic success model assumptions for the four major types of stakeholders as described previously in Section 1.1.2.

Although it may not be entirely obvious, clashes can occur between models of every pair of the four types, as is exemplified in Table I.

1.2.2 Consequences of Model Clashes

Consequences of model clashes can be:
- insignificant in the case of clashes whose negative effects on the project or product are easily reversed. An example would be two (different stakeholders’) product models that include different ways of approaching the graphical design of web pages;
- time- and resource-consuming in the case of a clash resulting from:
  - a developer’s (success model) desire to be free to start a more interesting project in a month, and, as a consequence, a product model incorporating a quick-to-implement but inflexible architecture;
  - a customer’s (success model) need for the product to be extensible;
- catastrophic in the case of:
  - an inexperienced development manager’s process model in which the development team will proceed apace on all the product’s features, with the idea of doubling the team’s size if progress hasn’t been sufficient a month before the scheduled delivery date;
  - a marketing manager’s success model requiring that the product be shown at Comdex two days after the scheduled delivery date;

The MasterNet project, described in [20] is one of many important software development projects that turned into catastrophes because of undetected model clashes and/or detected but unresolved model clashes, and/or detected but badly resolved model clashes. The MasterNet project was to develop a system that would update and automate the online generation of monthly statements for Bank of America’s trust accounts. The project began in the early 1980s with a $22 million budget and a two-year completion schedule.

Project contractor/developer Premier Systems tried to build MasterNet by scaling up a small existing trust system. Their misguided effort took five years, cost $80 million, and delivered a system that Bank of America rejected. Worse yet, the project tarnished Bank of America’s reputation, causing some of its major customers to lose confidence in the bank. The total number of institutional accounts dropped from 800
INTRODUCTION TO MBASE

Fig. 1. Common model clashes (MasterNet).
### Table I

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

\textsuperscript{a} Structure clashes between a project’s input and output structures [21].

\textsuperscript{b} Traceability clashes among a product’s requirements, design and code [28]; and architectural style.

\textsuperscript{c} Clashes in integrating commercial off-the-shelf (COTS) or other reusable software components [19].

To 700, while managed assets shrank from $38 billion to $34 billion. The MasterNet project suffered from the following model clashes:

- **Product-Property.** The MasterNet users’ (product model) desire of many features resulted in 3.5 million lines of code, which conflicted with Bank of America’s, i.e., the customers’ (property model of) limited budget and schedule. This conflict led to a large overrun in budget and schedule.

- **Property-Product.** The customer’s budget and schedule (property) models assumed stable requirements, but the users’ and developer’s (product) model of frequent feature changes also contributed to the large overrun.

- **Product-Product.** The developer’s (product) model of users’ needs provided features, such as full customer access, that matched poorly with real users’ (product model) needs such as accurate and timely reports.
INTRODUCTION TO MBASE 217

– **Product-Property.** The developer’s (product) model of server to be used—one manufactured by Prime, Inc.—offered inadequate performance and reliability to meet users’ desired level of service (property) models. Even after several upgrades, the Prime system suffered repeated performance overloads and crashes.

– **Product-Success.** The developer’s (product) model of server to be used also conflicted with the users’ and maintainers’ applications-compatibility (success) model, as Bank of America had relied exclusively on IBM hardware and software up to that point.

– **Process-Property.** The maintainers’ tightly scheduled transition (process) model conflicted with the testing delays brought on by the customer’s limited development budget and schedule (property) model.

The clashes shown with solid lines in Fig. 1 are among those that occurred in the MasterNet project.

2. Dealing With Model Clashes

The first step in dealing with model clashes is identifying them. In the present section, we discuss two retrospective approaches to model-clash identification, i.e., two approaches to identifying model clashes after models have been constructed. In Section 3 we discuss MBASE’s prospective approach to dealing with model clashes, i.e., an approach in which clashes are identified and resolved as models are constructed.

2.1 Retrospective Approaches

The most obvious way to identify model clashes is to write down all assumptions in all stakeholders’ models, and to perform pair-wise consistency analysis. If each individual stakeholder’s model is already known to be sound, a bit of organizational help can be had by first sorting assumptions by stakeholder and organizing them as in the “spider web” diagram of the sort shown in Fig. 1. This seat-of-the-pants approach is extremely time consuming for a project of more than minimal size, and relies entirely on the developer’s experience and talent.

A second retrospective method based on [1], one that provides the developer with substantive aid in finding clashes, starts with:

1. a list/table of areas for which models must exist in any software development project;

2. for each model area, a list/table of critical issues that a model must address;
(3) for each major software development method/process, a list/table of that method’s/process’ assumptions on each major issue on which it makes assumptions (see below and [1] for examples);

(4) An “inverted” version of the lists/tables in item (3) that can be used to determine if a method/process chosen by the development team is consistent with stakeholder assumptions regarding the project at hand.

Examples of instances of the various lists are, respectively:

(1) The model areas shown in the middle column of the “Model Assumptions Identification Taxonomy” of Fig. 2. (The model areas are: Requirements; Architecture & Design; Coding–Testing–Integration–Transition; Maintenance; Environment; Developer(s); and Customer(s).)

(2) For each model area, the critical issues, shown in the right-most column of Fig. 2, that must be addressed by the model(s) in that area. For example, a requirements model for a project must address the issues of:

– when requirements are likely to be complete, i.e., whether they are complete at the project’s start or must evolve over time;
– the stability of requirements, i.e., the likelihood that requirements already agreed upon will have to be changed during the course of the project;
– whether it is a foregone conclusion that requirements can be met, or whether meeting the requirements is a challenge (risk issue);
– how clearly requirements must be specified, i.e., how much leeway developers have in deciding their fine points;
– the detail with which requirements are/must be documented.

(3) The waterfall process model assumptions and the COTS development model assumptions (see [1] for details on this).

(4) The model clash tables of [1], a small excerpt from which is shown in Table II.

3. MBASE

MBASE is founded upon pro-active, rather than retrospective, model “integration.” In a retrospective approach, models are first created and then checked for model clashes. In MBASE’s prospective approach, model clashes are identified and dealt with during the construction of models. But attempting to integrate every model, during its construction, with every other, partially-developed or completed, model would be implausibly time consuming if there were no further aid/direction.
An approach to this is to boil-down to the essentials what is required in order to enact effective model-clash identification and avoidance through model integration. Through years of refinement and empirical study a set of essentials has been established that if incorporated into a development process, will substantially reduce the risk of model-clashes [1]. These essentials define MBASE and any approach that in-
Table II

Example of Model-Clash Identification

<table>
<thead>
<tr>
<th>Model</th>
<th>Assumption</th>
<th>Completeness before design</th>
<th>Clash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>Developer defines and documents a complete set of requirements for each system capability before starting the system detail design</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>COTS</td>
<td>System requirements and architectures are often driven by the capabilities and structures of the final set of COTS components selected for integration</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

3.1 MBASE Model Integration Framework

The first essential comes from extensive empirical studies of model usage in large numbers of projects and analysis of their model-clash patterns. These studies reveal that the highest-risk model clashes are those that violate one or more of the relationships, shown in Fig. 3, among the four model types: success, product, process, and property.

![Fig. 3. Element integration.](image-url)
The five relationships shown in Fig. 3, with examples, are as follows:

**Success \rightarrow Product:** A product must satisfy the success criteria agreed upon among critical stakeholders. Another way of looking at this is that the agreed upon success models imply evaluation criteria for the product. Looked at this way, product models must represent a product that satisfies the evaluation criteria established by the success models. For example:

- An organization is developing a new hardware/software system; the organization has a strategic partnership with a specific hardware vendor; one element of the organization’s success model will, therefore, be the continued use of the vendor’s products. Use of the vendor’s products thus becomes an evaluation criterion for the product; it is violated by a developer’s product model in which the new system incorporates COTS products that do not run on the vendor’s platforms.

- A university service department is having an outside team develop an enhancement of its web site; the budget is reasonably low. One desired, but not absolutely required, feature is the ability to receive payment for services by credit card, as opposed to just by mail. The development team designs and implements a system that uses Paypal. Because of the low budget, the designers decide to use mySQL as the database engine. Because of the development team’s technology preferences, and the members’ backgrounds, they decide to use PHP as the scripting language. The developers do an excellent job of eliciting the client’s functional requirements, but fail to elicit the client’s success model elements to the effect that:
  - the head of the department is not willing to take on personal financial liability for departmental activities;
  - the web site must be hosted by the university’s IT department.

  The university doesn’t have a Paypal account, so the department head would have to open one and assume personal liability. The university’s IT department does not support mySQL or PHP. As a result, the product fails to meet critical success criteria and, though implemented and tested extremely well, cannot be made operational.

- A research department of a high-tech company is having a team develop a system for posting of papers and research results for use by its members. A product model includes universal posting rights as well as universal access. The customer’s success model naturally includes avoiding legal liability. The product model violates the implied evaluation criterion of allowing only legal postings by not vetting posts and, thereby, allowing users to post copyrighted material without appropriate compensation to copyright holders.
A software company is developing a product, on speculation, with the intention of demonstrating it at Comdex in nine months. The project staff consists of two developers and there is no budget for hiring additional staff. The product models include full maintenance and user documentation, an impossibility given the time to Comdex.

**Success → Process**: A software development process is defined by a set of activities and conditions, or criteria, for beginning and finishing—entering and exiting—each activity. (In this context, “process” refers not just to the overall process to be used to execute the project—XP, waterfall, spiral, etc.—but, rather, includes the result of applying the high-level process’s approach to the details of a particular software development project. It thus includes individual activities, dependencies among them, etc.) Success models must provide or, at least, be consistent with, entry and exit criteria for process.—Each milestone review in the process needs to use the agreed-on success models as pass/fail criteria. For example:

- A client’s IKIWISI success model is consistent with an overall process consisting of the construction of a prototype followed by evolutionary development.
- A project with a success model of extending a successful family of software products to meet developing market trends is consistent with a waterfall development process. In fact, the success model likely dictates a waterfall process as the least costly so long as the market trends are not changing too fast.
- A marketing manager’s success model of presenting a product at the next Comdex is inconsistent with a developer’s process model of performing beta testing before product release or of proving correctness before release.
- The stakeholders of a project that is developing a system on which lives may depend has the, extremely obvious, success model of no loss of life after the system goes operational. This success model is consistent with—probably dictates—that a testing method such as “clean room” be used or that formal techniques be used to prove correctness (exit criteria for testing).

**Process ↔ Product**: Process models must provide plans and control for product model development (and product models must supply milestone content for process models). For example:

- In general, the schedule (process model) of any project must include tasks/activities covering all the architectural (product model), class diagram (product model), etc. elements; it must also address the question of which elements will be developed by which developer(s), and when; it must also conform to product element dependencies.
– In general, the process model of any development project must include (control) details for the validation and verification of developed product (model) components, for the review of product (model) components, for the pace (control) and adjustment of pace of development of product (model) components.

– A very short schedule requires a product architecture (model) architected for ease of shedding lower-priority features.

Property → Process & Property → Product: Cost and schedule (property) models are used to evaluate and analyze the consistency of product, process, and success (not shown in diagram) models. In general:

– In any software development project, a cost-schedule (property model) estimate would determine whether implementation of the proposed (product model) design/architecture following the proposed (process model) schedule would result in software with the required (product model) capabilities and levels of service in the proposed (success model) time and within the proposed (success model) budget.

### 3.2 MBASE Process Integration Framework

The Model Integration Framework shows a static picture of inter-model consistencies that must be achieved for project success. Moreover, as indicated earlier, these are the very consistency patterns that are most frequently violated in actual projects, and, in fact, whose violations are among those most frequently responsible for project failure. Because it shows only a static picture, the Model Integration Framework doesn’t completely describe the MBASE notion of integrating models, to eliminate risky clashes, during the construction of models. This latter, dynamic, function is served by the Process Integration framework shown in Fig. 4.

This dynamic integration framework dictates how particular model-integration activities must relate to each other. In contrast to model-element integration which deals chiefly with the relationships between values contained within models as discussed in Section 3.1, process integration indicates what model development and stakeholder activities must be present between the models focus on high-risk model identification and avoidance. As to be expected, these integration activities are highly dependent on the model-element integrations, but are not equivalent to them. A concrete example of how the two integration frameworks collaborate is illustrated in detail within the extend example of Section 3.5.
3.3 MBASE Anchor Points

The idea of standardized milestones, or “anchor points,” that divide software development projects into phases is at least as old as the Waterfall Process, whose anchor points are [?,4]: completion of system requirements, completion of software requirements, completion of preliminary design, completion of detailed design, completion of implementation (coding), completion of unit testing, completion of software acceptance testing, and completion of system acceptance testing.

Where Waterfall is not used, developers still need well-defined anchor points to plan, organize, monitor, and control a project. A critical reason for using anchor points is to allow critical stakeholders to periodically—at important phase change points—determine if a project is proceeding satisfactorily and whether it pays to continue its execution. In [4], Boehm suggests three anchor points for use with any software development process, spiral or other. The three anchor points are called Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC). In very simple terms, a project reaches LCO when developers have produced a first cut at: objectives to be achieved for the sponsoring organization; a design/architecture; a project plan; and a demonstration of the feasibility of achieving the objectives through implementation of the design/architecture using the project plan.
TABLE III
LCO, LCA, IOC SUCCESS CRITERIA

<table>
<thead>
<tr>
<th>LCO</th>
<th>LCA</th>
<th>IOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>For at least one architecture, a system built to that architecture will:</td>
<td>For a specific detailed architecture, a system built to that architecture will:</td>
<td>An implemented architecture, an operational system that has:</td>
</tr>
<tr>
<td>- Support the core Operational Concept.</td>
<td>- Support the elaborated Operational Concept.</td>
<td>- Realized the Operational Concept.</td>
</tr>
<tr>
<td>- Satisfy the core Requirements.</td>
<td>- Satisfy the elaborated Requirements.</td>
<td>- Implemented the initial Operational Requirements.</td>
</tr>
<tr>
<td>- Be faithful to the Prototype(s).</td>
<td>- Be faithful to the Prototype(s).</td>
<td>- Prepared a system operation and support plan.</td>
</tr>
<tr>
<td>- Be buildable within the budgets and schedules in the plan.</td>
<td>- Be buildable within the budgets and schedules in the plan.</td>
<td>- Prepared the initial site(s) in which the system will be deployed for transition.</td>
</tr>
<tr>
<td>- Show a viable business case.</td>
<td>- Have all major risks resolved or covered by a risk management plan.</td>
<td>- Prepared the users, operators, and maintainers to assume their operational roles.</td>
</tr>
<tr>
<td>- Have its key stakeholders committed to support the Elaboration Phase (to LCA).</td>
<td>- Have its key stakeholders committed to support the full life cycle.</td>
<td></td>
</tr>
</tbody>
</table>

In similarly simple terms, a project reaches LCA when developers have produced a detailed/mature version: of objectives to be achieved for the sponsoring organization; of a design/architecture; of a project plan; of a demonstration of the feasibility of achieving the objectives through implementation of the design/architecture using the project plan.

A slightly more detailed description of the definitions of LCO, LCA, and IOC may be found in Table III.

The precise definition of each of the three milestones is in terms of a specific set of models, the degree of development that each must have attained, and the types of consistency that must be demonstrated between/among them in order for a project to attain that milestone. These naturally vary depending upon the specific development process being used.

3.4 MBASE Risk Management and Stakeholder (Win-Win) Involvement

For the first few decades of software development, the Waterfall process—requirements definition, followed by analysis and design, followed by implemen-
tation, testing, and deployment in that order—was the most commonly used development process. Developers working in later phases of a Waterfall project regularly recognized that errors had been made in earlier phases, and performed the rework necessary to correct them, though doing so violated the strict definition of Waterfall Process. Unfortunately, neither commercial nor governmental contracting practices caught up with these realities quickly enough and, as a result, many Waterfall projects ran into devastating technical and legal problems.

The worst problem with Waterfall Process is that early fixing of requirements doesn’t allow the developer the flexibility to deal reasonably with a project’s high-risk aspects if there are any. For example, it is often difficult to know in advance how expensive it might be for a proposed new system to achieve a specified response time—or even if current technology can support the response time. The Waterfall Process, however, dictates that some response time must be fixed in the requirements phase. A more prudent approach, but one inconsistent with strict Waterfall Process would be to:

– recognize that demanding the proposed response time is risky;
– buy information about the risk by doing a partial design and implementation of the response-time-related parts of the proposed system and measuring response times attainable using one or more available technologies;
– finalize an attainable response time requirement based upon tests performed—or canceling the system’s development because the response time initially proposed is critical to the system’s success, but is not attainable, at an acceptable cost, using current technology.

These days an equally important reason for doing risk-driven software development is that much software development is COTS-based [10]. Among the risks that arise from the incorporation of COTS products into a new software product is that:

– a COTS product may not have precisely the functionality promised in vendor literature;
– a COTS product may not live up to quality attribute levels promised in vendor literature;
– a new release of a COTS product may not support the same interfaces as previous versions; i.e., it may not integrates with other components of the new product as did earlier versions;

Waterfall Process works perfectly well in many types of highly-precedented project, for example: in the re-implementation, for a new platform, of a software product that has been successful on an older platform; in an implementation of the next member of a successful family of software products, where the new product’s extensions of the old one have been gleaned from years of experience with the old product.
– a new release of a COTS product may not have precisely the same functionality
and/or may not support the same quality attribute levels as did earlier versions;
– a COTS vendor may go bankrupt and no longer support periodically required
increases in quality attribute levels.

It was for reasons like these that Boehm proposed [4] the Risk-Driven Spiral
process for software development. While MBASE doesn’t, by any means, require
that the basic development process be the Spiral Process, every process can be
viewed as having phases of some sort; the notion of being risk-driven in MBASE
simply means:

– Establishing, for each phase:
  – product and project objectives;
  – constraints affecting both the product and the project;
  – alternative courses of action that can be taken to meet the objectives and
    satisfy the constraints;
– Evaluating the relative merits of the various alternatives, and their associated
  risks.
– Deciding upon the specific course of action (from among the possible alterna-
  tives) to be followed during the next phase.
– Validating the chosen course of action.
– Reviewing the validated choice and commencing the next phase.

MBASE’s borrowing is, thus, not the notion of spirals/iterations, but, rather, the
notion of:
– recognizing those aspects of a software development project that present high
  risks;
– buying information needed to resolve, mitigate or live with the risks, by what-
  ever means is reasonable, e.g., various types of prototyping;
– committing to a course of action based upon the results of the above.

A recurring theme is the need to reconcile the key stakeholders’ success models.
Thus, a stakeholder win-win negotiation process becomes a key step in each spiral
cycle of the MBASE approach, as shown in Fig. 5.

In the COMDEX application discussed previously (Section 3.1), for example, the
initial spiral cycle would focus on evaluating COTS products and scoping the overall
system to be buildable in nine months. In a subsequent spiral cycle, the next-level
stakeholders would include representative users of the e-commerce system, and the
reconciliation of their win conditions would include prototyping of the user interface
to eliminate the risk of showing up at COMDEX with an unfriendly user interface.

The MBASE tool support includes a groupware system called Easy WinWin, which enables distributed stakeholders to enter their win conditions and to negotiate mutually satisfactory (win-win) agreements with other stakeholders [12].

The WinWin spiral model in Fig. 5 provides another view of how risk considerations are used to reconcile stakeholder success conditions in terms of product, process, and property models. A complementary view was shown in Fig. 4 (see Section 3.2), which also identifies the win-win spiral model’s role in guiding the early feedback cycles involved in defining and reconciling the system’s domain, product, process, and property models.

3.5 MBASE Integration Example

The elements described above all work in concert to help identify and avoid potentially high-risk model clashes. The particular integration elements and activities are driven by the MBASE spiral (which incorporates the risk-driven stakeholder win-win approach) to deliver a comprehensive integrated models. To illustrate the synergistic application of MBASE elements, we consider the following example taken from an actual project. A USC Librarian presented the following problem statement for a multi-media information system:
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.

We now describe how the model-integration steps within MBASE elements might be performed within the Medieval Manuscript (MM) project throughout one full MBASE spiral cycle. The initial goal here is to develop models that will satisfy the LCO success criteria (i.e., identify at least one feasible architecture). That is, pass the first anchor point milestone (as described in Section 3.3).

Spiral step 1: Identify initial stakeholders. The primary stakeholders for the MM project are the librarian Mrs. W. who is an authority on the archival of medieval manuscripts and will serve as domain expert in addition to customer, and Mr. Z. a graduate student in Software Engineering who is leading the development team. Hence the initial stakeholder set for this step is Mrs. W. and Mr. Z. The Process Integration framework dictates that these stakeholder must work together to “describe the enterprise context in” for domain of the MM project. In this case, the enterprise context is “Multimedia Library Services” and the domain is “Multimedia Information Services.” A Domain model was developed by making use of a domain description from previous Multimedia Archive Library project. This step is illustrated graphically for this instance in Fig. 6.

Spiral step 2: Identify stakeholder win conditions. This step dictates that Mrs. W. and Mr. Z. converge on an initial set of mutually satisfactory win-conditions for the MM project. Aside from the functionally elements (e.g., main application functions,
Humanized I/O, etc.), Mrs. W. wanted to ensure development and operational costs did not outweigh the value MM could provide in a short amount of time (high return on her investment). As this project originated within a 2-semester software engineering course project, the project must be completed within 12 months. The win-conditions establish the initial Success and the Basic and Main capabilities were negotiated to provide the candidate Product models (whose context was obtained from the Domain model created previously). (See Fig. 7.)

Spiral step 3: Reconcile win-conditions; establish objective, constraints, and alternatives. This is a major step in the model-integration process. Here the process integration framework dictates that Mrs. W. and Mr. Z. need to agree on what constitutes a mutually satisfactory starting and ending point for the project. In this case, “success” has been defined that the MM product will have a high return on investment and be completed within 12 months. It was agreed that a high return on investment was likely to be achieved if enough “valuable” features were implemented. Here “enough” meant that there was a high-value for relatively little effort (i.e., “high-payoff”). The features were ordered according to development dependencies and value to the Librarians (in terms of time saved, lower staffing requirements, customer satisfaction, etc.). The “exit” point for the development now was defined as “build enough valuable features in order to obtain at least 80% of the overall expected value of the MM system.” This is what is meant by the “high payoff” range. The model-integration framework dictates that this must be used to evaluate the Basic and Main features described in the Product models. In addition, Mr. W. had to choose a Process model that would likely develop the product into the high-payoff

Fig. 7. MM project spiral step 2.
range and satisfy the 12-month schedule constraint. A Design to Schedule model was chosen was thought to be compatible with this. (See Fig. 8.)

**Spiral step 4:** Evaluate product and process alternatives. In this step, the integration and process frameworks dictate that the Product models be evaluated with respect to their possible size and used to estimate the overall effort. A Function Point (FP) estimate was made for the Basic capabilities and used as the size input for a COCOMO II effort estimate. Again, as dictated by the MBASE integration framework, this effort estimate must reconcile with the 12-month development constraint within a design to schedule to deliver the high-payoff features. If it does not reconcile, then the development parameters must be adjusted (more staff, extra tools, buy COTS, etc.) or a different Process model must be considered. As a last resort, the high-payoff feature set can be re-negotiated. In the case of MM, some re-negotiation of the Basic Capabilities was needed. Specifically, it was agreed that a COTS package would be used to enable the user to navigate a manuscript image (zoom, move to a location, etc.) within a web browser. This increased the project cost and modified the specifics of some of the Basic Capability requirements. It will have to be verified that these changes do not reduce the value outside the high-payoff range. (See Fig. 9.)

**Spiral step 5:** Define next Product and Process elements and activities. At this point the Product models are refined into basic designs. The integration framework dictates that these designs must be used as input ("milestone content") to the Design to Schedule process. This must result in an explicit plan within the Design to Schedule process to implement the Basic Capabilities (however, the Design to Schedule process can and likely will include other plans). The high-risk potential due to the use of an unknown COTS package indicates that the development plan should include some prototyping use of the COTS package under consideration and draft web-page implementations. (See Fig. 10.)

**Spiral step 6:** Validate product and process definitions. As per the process integration framework, Mr. Z. now verifies that the new effort estimate satisfies the 12-month development constraint and can be used as an explicit measure (preliminary) of likely success of the project. A problem occurred in that the COCOMO II effort estimate cannot adequately account for use of the COTS package now under consideration. This Property model now clashes with the new Product model (COTS). With this model clash identified, Mr. Z. chose to undertake some initial prototyping with the COTS package to determine what the COTS effort might be. After this, Mr. Z. was able to make a confident COTS integration effort estimate and add it to the COCOMO II effort predicted for implementing the Basic Capabilities outside what the COTS package handles. (See Fig. 11.)

**Spiral step 7:** Review commitments. Now with slightly re-negotiated Basic Capabilities, verified that they still are compatible with the Success, Process, and Property
FIG. 10. MM project spiral step 5.
INTRODUCTION TO MBASE 235

Fig. 11. MM project spiral step 6.
models. That is, buy the COTS package to implement the image navigation capabilities and custom build the remainder of the Basic Capabilities. (See Fig. 12.)

Our tour through one MBASE cycle has illustrated how the integration frameworks work together to drive a project towards the LCO anchor point while either avoiding model clashes as a matter of course (though judicious model choice or explicit sharing of information) or identifying potentially risky model clashes by explicitly verifying model element compatibility. While we did not explicit describe what particular activities evolved the project towards LCO, these are somewhat self evident given the definition of the integration frameworks. For example, developing the Success models and Product models contribute to the Operational Concept elements of the LCO.

3.6 MBASE Invariants

MBASE is intentionally very broad in order to encompass a broad spectrum of projects both large and small. Within the MBASE superset, there are five elements, the model integration guidelines that are universal for all MBASE projects. These elements are called invariants. Additionally, there are elements of MBASE, the process and method guidelines, which are categorized as variants, and can be adjusted according to particular project parameters (team size, application scope, etc.). Through use of the invariants and particular choices of these variants MBASE becomes a development process add-on framework/adjunct that can be used with any software development process, from agile to Plan-driven, for identifying and dealing with model clashes. For example, MBASE model integration, MBASE process integration, and the three other attributes that define MBASE can be attached/adjointed to XP, or SCRUM, or Waterfall, or Risk-Driven Spiral to form MBASE-XP, MBASE-SCRUM, MBASE-Waterfall, or MBASE-Risk-Driven Spiral.

There are five “MBASE invariants” based on the essentials described in Sections 3.1–3.4 that define MBASE. If X is a software development model, then X-BASE is the extended version of X in which:

(i) the model integration framework (discussed in Section 3.1) and
(ii) the process integration framework (discussed in Section 3.2) have been added to X;
(iii) the developers use the Life Cycle Objectives (LCO, Life Cycle Architecture (LCA), and Initial Operational Capability (IOC) milestones/anchor points (defined in Section 3.3) as stakeholder commitment points for proceeding from stage to stage of the project. (Note that [30] LCO is the milestone at the end of the Inception phase of a project, LCA is the milestone at the end of the Elaboration Phase, and IOC is the milestone at the end of the Construction phase.)
Fig. 12. MM project spiral step 7.
(iv) a stakeholder win-win relationship (defined in Section 3.4) is defined and sustained, in a fashion compatible with X, for the purpose of dealing with model clashes, throughout the project’s life-cycle. This means that periodically:

- success-critical stakeholders are identified;
- each stakeholder participates in the integration of models relevant to him/her;
- models are modified, as required;
- all critical stakeholders come to an agreement on the latest, updated models to be used to further pursue the project.

(v) model integration activities are risk-driven as are contents of all artifacts produced during model integration (discussed further in Section 3.4). That is, such details as:

- the number of times that stakeholders meet between anchor points;
- the specific issues/models dealt with in developing project artifacts, other than LCO-, LCA-, and IOC-mandated issues/models (see below);
- the detail with which each issue/model is addressed;
- the total amount of time spent on model integration.

be decided through an evaluation of their risk-reduction value.

The invariants impose a significant amount of structure on a process used to integrate models, yet they leave room to accommodate a number of important project-related choices. In fact, the invariants imply that certain choices must be “tailor” made per project. These are called MBASE “variants,” i.e., choices of project details that are not dictated. Some of these include:

- The development process used to execute the project—XP, Scrum, RAD, waterfall, spiral, RUP, incremental, etc.
- The methods used to construct, or notations used to represent, specific models.
- The amount of development effort between anchor points.
- The staffing levels of the various, MBASE and non-MBASE project activities.

Table IV summarizes the MBASE invariants and variants which will be further elaborated in the subsequent sections.

### 3.7 Plan-Driven MBASE

As indicated previously, MBASE is intended for use with any software development process. Up to this point it has been used mainly with more “Plan-driven” or...
INTRODUCTION TO MBASE

TABLE IV
MBASE INVARIANTS AND VARIANTS

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Defining and sustaining a stakeholder win-win relationship through the system’s lifecycle.</td>
<td>(1) Use of particular success, process, product, or property models.</td>
</tr>
<tr>
<td>(2) Using the MBASE Model Integration Framework.</td>
<td>(2) Choice of process or product representation.</td>
</tr>
<tr>
<td>(3) Using the MBASE Process Integration Framework.</td>
<td>(3) Degree of detail of process, product, property, or success modeling.</td>
</tr>
<tr>
<td>(4) Using the LCO and LCA Anchor Point milestones.</td>
<td>(4) Number of spiral cycles or builds between anchor points.</td>
</tr>
<tr>
<td>(5) Ensuring that the content of MBASE artifacts and activities is risk-driven.</td>
<td>(5) Mapping of activities onto Inception–Elaboration–Construction–Transition phases.</td>
</tr>
<tr>
<td></td>
<td>(6) Mapping of staff levels onto activities.</td>
</tr>
</tbody>
</table>

"heavier weight" development processes—variants of Stakeholder Win-Win Spiral Process. One such variant of MBASE has been used by over 1000 developers, on about 200 projects, in Barry Boehm’s USC CS577, a two-semester graduate-level Introduction to Software Engineering. Other variants have been used by XEROX need to fill some detail on industry and/or government uses such as with the U.S. Army. More agile variants of MBASE are currently under development.

Few of the over-1000 MBASE developers were, however, sufficiently skilled and sufficiently experienced to be able to use MBASE given no more than its definition in Sections 3.1–3.5 above. The intended use of MBASE is, thus, via incorporation into software development guidelines, different ones for different types of project. That is, each set of MBASE guidelines specifies particular models to be constructed as well as methods for constructing them in such way that the five MBASE invariants are followed in their construction.

For example, in CS577 students execute real software development projects for real clients, mostly USC faculty and staff, but occasionally for outsiders—non-profit organizations and, less frequently, small businesses. Development teams typically consist of five people, and projects run for two semesters. As a result, the development process used is usually Schedule As Independent Variable [9], a process in which the inception and elaboration phases are run as Stakeholder Win-Win, Risk-Driven Spiral projects, and the implementation phase is run either as a Waterfall project or as one type or another of Incremental project—depending upon the project’s circumstances, and decided upon, with guidance, by the teams themselves.
The purpose of CS577 is, however, two-fold. One purpose is to guide students through an actual project in which five people work, part time, over a period of approximately 24 weeks, with product delivery and transition required by the end of the twenty-fourth week. A second purpose is to educate students in developing larger scale software products. The CS577 MBASE guidelines [18] are therefore, designed to elicit somewhat more documentation than is actually required for the execution of the project. Industry/government MBASE guidelines have been developed for specific types of larger projects. We are currently developing a set of minimal MBASE—referred to as minBASE—guidelines that can be tailored upward for use with any development process from agile to “Plan-Driven.”

In the CS577 version of the MBASE guidelines, the models to be created are organized under artifacts named Operational Concept Definition (OCD), System and Software Requirements Definition (SSRD), System and Software Architecture Definition (SSAD), Life Cycle Plan (LCP), and Feasibility Rationale Description (FRD). As an example, Table V below shows the components of the LCO (Life Cycle Objectives) version of the OCD. Appendix A shows the components of all LCO, LCA (Life Cycle Architecture) and IOC (Initial Operational Capability) versions of CS577 MBASE artifacts.

<table>
<thead>
<tr>
<th>Table V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LCO Milestone Elements</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operational Concept Definition (OCD)</th>
<th>Top level system objectives and scope</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>– system boundary (i.e., what are the developers responsible for developing and what in the system’s environment may they assume is either being developed by someone else or already exists);</td>
</tr>
<tr>
<td></td>
<td>– environment parameters and assumptions (i.e., key parameters and assumptions about the nature of the system’s users, data volume and consistency, workload levels, interoperating systems, etc., both at initial operating levels and in terms of likely evolution);</td>
</tr>
<tr>
<td></td>
<td>– evolution parameters.</td>
</tr>
<tr>
<td></td>
<td>Operational Concept</td>
</tr>
<tr>
<td></td>
<td>– operations and maintenance scenarios and parameters. (These many involve prototypes, screen layouts, UML diagrams, and other relevant representations. They should include scenarios for off-nominal situations if important and for system maintenance, including which organizations are responsible for funding and performing the various functions.)</td>
</tr>
<tr>
<td></td>
<td>– organizational (stakeholders’) life-cycle responsibilities.</td>
</tr>
</tbody>
</table>
The guidelines specify traceability and consistency requirements, as well as required FRD (Feasibility Rationale Description) arguments which, together, guarantee, to the extent possible, the consistency of all project models.

Table VI shows some of these consistency traces.

Given the relatively Plan-Driven nature of the MBASE guidelines variant described above, it might be unclear what a more agile version of MBASE might look like. The first point to note in this regard is that the following proviso is included in every section of more Plan-Driven MBASE guidelines:

If it’s risky to document something, because doing so expends stakeholder time and other resources, but doesn’t likely contribute significantly to the success of the project, then it should not be documented; if its risky not to document something, because the risk exposure arising from not documenting it exceeds the cost, in time and other resources, of documenting it, then it should be documented.

Agile methods take this one step further [17] by starting from a baseline of producing an artifact to be archived only when stakeholders agree that the cost of doing so is outweighed by the long-term value of the artifact, or, in other words, if the risk exposure of not producing the document is greater than the cost of producing it. In terms of other agile-process details, MBASE, by virtue of its variants (see Section 3.6 above) holds no brief for or against such practices as pair programming.

### Table VI: Coverage/Traceability of MBASE Product Models*

<table>
<thead>
<tr>
<th>Domain Description</th>
<th>System Analysis</th>
<th>System Design</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Background</td>
<td>Statement of Purpose</td>
<td>System Definitions</td>
<td>Release Description</td>
</tr>
<tr>
<td>Organization Goals</td>
<td>Project Goals</td>
<td>Project Requirements</td>
<td>Req. Satisfication</td>
</tr>
<tr>
<td>System Capabilities</td>
<td>Levels of Service Goals</td>
<td>LOS Requirements</td>
<td>LOS Tests</td>
</tr>
<tr>
<td>Organization Activities</td>
<td>System Capabilities</td>
<td>Capability Requirements</td>
<td>Capability Tests</td>
</tr>
<tr>
<td>Organization Entities</td>
<td>Behavior Model</td>
<td>Operation Model</td>
<td>Methods/Functions</td>
</tr>
<tr>
<td>Interaction Model</td>
<td>Component Model</td>
<td>Object Model</td>
<td>Data Structures</td>
</tr>
<tr>
<td></td>
<td>Enterprise model</td>
<td>Class Model</td>
<td>*Does not include all MBASE models</td>
</tr>
</tbody>
</table>

*Operational Concept Description (OCD)  Construction,Transition,Support (CTS)  System and Software Requirements Definition (SSRD)  System and Software Architecture Description (SSAD)
XP-style low-detail-upfront architecture, user stories vs. UML use cases, short, e.g., weekly, incremental cycles, or, for that matter, most other aspects of XP, SCRUM, and other agile processes.

A project is perfectly in consonance with the definition of MBASE if, in the most extreme case, stakeholders can, in face-to-face conversations, however short or long in duration:

– agree on a set of models to construct;
– sketch out those models without preserving any written documentation of them;
– refine the models, using informal versions of the MBASE model and process integration guidelines;
– agree on the consistency of all models, except, possibly, those whose clashes present low risk to the project;
– at suitable points in time, agree that LCO, LCA, and IOC milestone requirements have been met.

It is with this in mind that we are developing a minimal, upwardly-tailorable, set of MBASE guidelines.

As important as is model-clash elimination to the success of software development projects, not all projects are candidates for the use of MBASE—at least not full-scale MBASE. For one thing, MBASE comes with a certain amount of overhead that is justified only when it is outweighed by a sufficiently large reduction in risk exposure. As an example, consider as project whose goal is to port an successful product to a new platform or one whose goal is to implement relatively small incremental extensions to a successful product family. In both these cases, the small reduction in risk exposure provided by the use of MBASE is not likely to be justified by the added overhead.

Finally, some of the work dictated by MBASE to be done during one project phase, say elaboration, is for use in a later phase, say implementation. For example, the LCA (end-of-elaboration phase) LCP (Life Cycle Plan) in the CS577 MBASE guidelines must contain a detailed schedule for the implementation phase. A project whose goal is the production of a prototype to possibly be implemented at a later date, is not, therefore not a candidate to be an MBASE project.

4. MBASE Usage Experience

For the past seven years, USC has used and refined MBASE extensively within its two-semester graduate software-engineering course. The students work on a Web-based electronic services project for a real USC client (frequently a digital library
application for the university information services division) from initial system definition through transition, utilizing a specialized form of MBASE. This specialization includes particular tools and models such as Easy WinWin, Rational Rose, MSProject, and elements of the Rational Unified Process. More than 200 real-client projects have used MBASE, and over 90 percent have delivered highly satisfactory products on very short fixed schedules. The annual lessons learned have been organized into an extensive set of usage guidelines and an Electronic Process Guide [18], all accessible at http://sunset.usc.edu/research/MBASE. In the spring of 1999, MBASE was used in both the undergraduate and graduate software engineering courses at Columbia University. Although these are single semester courses, MBASE was successfully adapted to help student teams complete a full project life cycle for real clients.

Within industry, Xerox has adopted many elements of MBASE to form its time-to-market process, including the use of the LCO and LCA anchor points as synchronization points for the hardware and software portions of their printer product definitions.

As mentioned previously, Rational has adopted the LCO, LCA, and IOC anchor points within their Rational Unified Process while MBASE adopted Rational’s Inception–Elaboration–Construction–Transition phase definitions.

C-Bridge has mapped their define, design, develop, deploy rapid development methodology for e-commerce systems to the MBASE spiral model.

The Internet startup company Media Connex adopted MBASE and used Easy WinWin to establish win-win relationships among their key stakeholders. Each of these companies converged on different balances of discipline and flexibility to satisfy their stakeholders’ success models.

Additionally, there are numerous companies and organizations directly making use of MBASE elements within their project development efforts. For example, the U.S. Army Tank and Automotive Command has used Easy WinWin and other MBASE elements to reconcile its software technology organizations’ process and product strategies.

ACKNOWLEDGEMENTS

The majority of the material presented here was developed by, or in heavy collaboration with Barry Boehm, Director of the University of Southern California’s Center for Software Engineering. We thank him for his many contributions and editing of this presentation. Several of the extensions and refinements of MBASE presented here were developed by Mohammad Al-Said as part of his doctoral dissertation. We would like to acknowledge the support of the Defense Advanced Research Projects Agency and the National Science Foundation in establishing and refining MBASE,
the DoD Software Intensive Systems Directorate in supporting its application to DoD programs and organizations, and the affiliates of the USC Center for Software Engineering for their contributions to MBASE.

Appendix A: LCO, LCA, and IOC Artifacts

Tables A.1–A.5 show detail of the models, and their degrees of completion that define attainment of the LCO milestone.

Table A.6 shows detail of the models, and their degrees of completion that define attainment of the LCA milestone.

**Table A.1**

<table>
<thead>
<tr>
<th>LCO Milestone Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operational Concept Definition (OCD)</strong></td>
</tr>
<tr>
<td>- Top level system objectives and scope</td>
</tr>
<tr>
<td>- system boundary (i.e., what are the developers responsible for developing and what in the system’s environment may they assume is either being developed by someone else or already exists);</td>
</tr>
<tr>
<td>- environment parameters and assumptions (i.e., key parameters and assumptions about the nature of the system’s users, data volume and consistency, workload levels, interoperating systems, etc., both at initial operating levels and in terms of likely evolution);</td>
</tr>
<tr>
<td>- evolution parameters.</td>
</tr>
<tr>
<td>- Organizational Concept</td>
</tr>
<tr>
<td>- operations and maintenance scenarios and parameters. (These many involve prototypes, screen layouts, UML diagrams, and other relevant representations. They should include scenarios for off-nominal situations if important and for system maintenance, including which organizations are responsible for funding and performing the various functions);</td>
</tr>
<tr>
<td>- organizational (stakeholders’) life-cycle responsibilities.</td>
</tr>
</tbody>
</table>

**Table A.2**

<table>
<thead>
<tr>
<th>LCO Milestone Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System &amp; Software Requirements Definition (SSRD)</strong></td>
</tr>
<tr>
<td>- Top level functions, interfaces, and quality attribute levels, including</td>
</tr>
<tr>
<td>- priorities;</td>
</tr>
<tr>
<td>- growth vectors.</td>
</tr>
</tbody>
</table>
INTRODUCTION TO MBASE

TABLE A.3

LCO Milestone Elements

<table>
<thead>
<tr>
<th>System &amp; Software Architecture Definition (SSAD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Top level definition of at least one feasible architecture, including:</td>
</tr>
<tr>
<td>– logical and physical elements and their relationships;</td>
</tr>
<tr>
<td>– choices of COTS products and other reusable software components.</td>
</tr>
<tr>
<td>Note that:</td>
</tr>
<tr>
<td>– At this (LCO) stage, it is acceptable to still be considering multiple feasible architectures, e.g., two feasible alternative COTS choices—with different architectural implications—for one or more critical components.</td>
</tr>
<tr>
<td>– If no feasible architecture can be developed, then the project should be canceled or its requirements, scope, and objectives should be changed.</td>
</tr>
<tr>
<td>• Identification of infeasible architecture options (in order to prevent wasted time resulting from later re-consideration of alternatives already ruled out).</td>
</tr>
</tbody>
</table>

TABLE A.4

LCO Milestone Elements

<table>
<thead>
<tr>
<th>Life Cycle Plan (LCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Identification of life-cycle stakeholders, e.g., customers, users, developers, maintainers, inter-operators, general public, etc.</td>
</tr>
<tr>
<td>• Identification of life-cycle process model.</td>
</tr>
<tr>
<td>• Top level WWWWWHH (Why, What, When, Who, Where, How, and How Much) by stage. The WWWWWHH principle organizes the plan into:</td>
</tr>
<tr>
<td>– Objectives: Why is the system being developed?</td>
</tr>
<tr>
<td>– Milestones and Schedules: What will be done by When?</td>
</tr>
<tr>
<td>– Responsibilities: Who is responsible for a function? Where are they organizationally located?</td>
</tr>
<tr>
<td>– Approach: How will the job be done, technically and managerially?</td>
</tr>
<tr>
<td>– Resources: How much of each resource is necessary?</td>
</tr>
</tbody>
</table>

The key elements of the IOC anchor point are:

– Software preparation, including both operational and support software with appropriate commentary and documentation; data preparation or conversion; the necessary licenses and rights for COTS and reused software, and appropriate operational readiness testing.
TABLE A.5

<table>
<thead>
<tr>
<th>LCO Milestone Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility Rationale Definition (FRD)</td>
</tr>
<tr>
<td>• Assurance of consistency among above-listed milestone elements</td>
</tr>
<tr>
<td>– via analysis, measurement, prototyping, simulation, etc. (with the purpose of demonstrating that adherence to the LCP in the implementation of the design in the SSAD will result in a product that satisfies the requirements in the SSRD);</td>
</tr>
<tr>
<td>– via business case analysis for requirements and feasible architectures (with the purpose of showing that implementing the requirements in the SSRD will accomplish the objectives in the OCD and will create enough value to justify the investment).</td>
</tr>
</tbody>
</table>

TABLE A.6

<table>
<thead>
<tr>
<th>LCA Milestone Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational Concept Definition</td>
</tr>
<tr>
<td>• Elaboration of system objectives and scope.</td>
</tr>
<tr>
<td>• Elaboration of operational concept.</td>
</tr>
<tr>
<td>System &amp; Software Requirements Definition</td>
</tr>
<tr>
<td>• Elaboration of functions, interfaces, and quality attribute levels including identifications of TBD's (items To Be Determined).</td>
</tr>
<tr>
<td>• Prioritization of Concerns.</td>
</tr>
<tr>
<td>System &amp; Software Architecture Definition</td>
</tr>
<tr>
<td>• Choice of architecture and elaboration of:</td>
</tr>
<tr>
<td>– logical and physical components, connectors, configurations, and constraints;</td>
</tr>
<tr>
<td>– COTS and other software reuse choices;</td>
</tr>
<tr>
<td>– Domain architecture and architectural style choices;</td>
</tr>
<tr>
<td>• Architecture evolution parameters.</td>
</tr>
<tr>
<td>Life Cycle Plan</td>
</tr>
<tr>
<td>• Elaboration of WWWWWHH for Initial Operational Capability (IOC), including partial identification And elaboration of key TBD’s.</td>
</tr>
<tr>
<td>Feasibility Rationale Definition</td>
</tr>
<tr>
<td>• Assurance of consistency among all (OCD, SSRD SSAD, and LCP) LCA milestone elements.</td>
</tr>
<tr>
<td>• Resolution or coverage, by risk management plan, of all major risks.</td>
</tr>
</tbody>
</table>

- Site preparation, including facilities, equipment, supplies, and COTS vendor support arrangements.
- User, operator and maintainer preparation, including selection, team building, training and other qualification for familiarization usage, operations, and maintenance.
REFERENCES


248

D. KLAPPHOLZ AND D. PORT


