

# A Framework for Systematic Evaluation of Software Technologies

Alan W. Brown & Kurt C. Wallnau

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213  
(awb, kcw@sei.cmu.edu)

## Abstract

*Many organizations struggle to make informed decisions when investing in new software technologies. This paper examines the problems of evaluating the likely impact of a new software technology within an organization, and describes an important component of such an evaluation based on understanding the “deltas” provided by the new technology. The main contribution of this paper is a framework for organizing software technology evaluations that highlights technology deltas. That is, the framework centers on the premise that new technologies must be positioned within the context of their contemporaries, and analyzed from the point of view of what they contribute in relation to those existing technologies.*

*The technology delta framework is described, and is illustrated through its application to the Object Management Group’s Object Management Architecture (OMA). Strengths and weaknesses of the approach are then analyzed, and required future work in this area discussed.*

## 1. Introduction

All organizations developing or using software-intensive systems must continually make decisions regarding the selection, application, and introduction of new software technologies. Some technology decisions are made explicitly following a detailed examination of the alternatives (e.g., deciding on a standard word processor within the organization), while others are made implicitly with little detailed study of the potential impact of the decision (e.g., deciding to ignore a new product line from an operating system vendor). In all of these cases the organization is attempting to understand and balance a number of competing concerns regarding the new technology. These concerns include:

- the initial cost of acquiring the new technology;

- the long term effect on quality, time to market, and overall cost of the organization’s products and services when using the new technology;
- the impact of introducing the new technology into the organization in terms of training, and other necessary support services;
- the relationship of this new technology in the organization’s overall future technology plans;
- the attitude and actions of direct competitor organizations with respect to this new technology.

Based on these and other factors the organization develops an assessment of the likely return on investment (ROI) of using this new technology. While in many domains ROI has a well-defined meaning and is calculated using established techniques and formulas, this is not the case in the software technology domain. In fact, while highly desirable, attempts at developing a general, repeatable approach to the calculation of software technology ROI have been unsuccessful. There are many reasons for this, foremost among them being the difficulty in establishing cause and effect when assessing the impact of new software technologies within an organization.

Unable to generate a concrete prediction of ROI for a new technology, most organizations must instead obtain an informed view of the technology based on applying a collection of techniques. While these techniques are mostly informal in nature (e.g., attending trade shows, pilot application, case studies) they nevertheless lead to an “informed intuition” about the technology that can be used to make a decision.

Performing technology evaluations is an important component of the role of Software Engineering Institute (SEI). The SEI provides advice to its customers on current software engineering best practices, helps to transition promising technologies, and raises awareness of future technology trends. One particular effort underway at the SEI, called **STEIM** (Software Technology, Evaluation, Integration and Measurement), focuses on developing

evaluation techniques and measures which are particularly applicable to technologies that support integration of systems comprised of off-the-shelf components. Our experiences with examining a range of system integration technologies highlighted the need for greater rigor in the way in which our evaluation data was collected and analyzed, and for a more systematic approach to planning, executing, and interpreting evaluation results. To address these problems we have developed a conceptual framework that can be used to categorize the different software technology evaluations that are possible, and suggest a method for carrying out a systematic evaluation based on populating the framework with the results of specific experiments that reveal information about a software technology.

Briefly stated, the premise of our evaluation framework is that one key piece of evidence needed by an organization is qualitative and quantitative reports on technology “deltas”, i.e., descriptions of the impact of the new features introduced by a technology as differentiated from features found in existing technologies. Hence, in this paper we describe a framework that highlights techniques for identifying and describing these technology deltas, and for defining focused application-oriented experiments to estimate the costs and benefits of these deltas in particular usage scenarios.

The remainder of this paper is organized as follows. Section 2 reviews existing techniques that are used to evaluate and compare different software technologies. Section 3 describes the technology delta framework and its use in performing software technology evaluations. Section 4 provides a detailed case study of the use of the technology delta framework. Section 5 summarizes the main points of the paper and briefly describes future work we plan to carry out in this area.

## 2. Current Approaches to Technology Evaluation

Most organizations recognize the importance of technology refreshment to improve the quality of their products and services, to be competitive with other organizations providing similar products and services, and to remain attractive to investors in the organization and to a technology-oriented workforce. To stay in business an organization must invest in appropriate new technologies.

Hence, careful decision making on new technologies is essential to an organization. Given the release of an update to an existing technology, or the availability of a new competing technology, an organization must initiate an evaluation process that provides timely, balanced information on which an informed decision can be made. As a result, a number of approaches and techniques for

technology evaluation are in use, and a number have been described in the software engineering literature. In this section we review the approaches commonly in use in practice, and those described in the literature.

### 2.1. Technology Evaluation in Practice

Today technology evaluations are typically carried out in an ad hoc way, heavily reliant on the skills and intuition of the staff carrying out the evaluation. By examining current industrial practice, and through surveying existing experience reports we can identify a number of approaches that are being employed by an organization. These approaches include:

- obtaining objective data on the technology by documenting case studies at other organizations;
- gathering subjective opinions and experiences with the new technology by attending trade shows, and by conducting interviews or by sending out questionnaires to vendors and users of the technology;
- conducting focused experiments to mitigate high-risk aspects of a new technology;
- demonstrating the feasibility of a new technology by executing a pilot project;
- comparing a new technology to existing practices by conducting a shadow project and examining the results of both approaches;
- phased exposure to a new technology by initiating demonstrator projects within a small part of the organization.

Sometimes an organization focuses on one of these approaches, while at other times some combination of approaches is employed.

Regardless of the approaches used we find that what is missing is a well-developed conceptual framework for technology evaluation that allows the results of the evaluation to be considered in terms of what this new technology adds to the existing technology base. Rather, a typical organization carries out one or more of the approaches above, gathers any resultant data, and forms an intuition concerning the value of that technology. It is our assertion that much of the informality in interpreting the results of any evaluation is due to the absence of:

1. well-defined goals before starting on the evaluation;
2. controlled, rigorous techniques for data gathering when carrying out the evaluation;
3. a conceptual framework for analyzing the data that is produced in the context of existing technologies.

While similar observations have been made [1], most attention has been concentrated on the second of these issues, exploring the need for rigorous data gathering and

the use of quantitative software evaluation techniques (e.g., [2][3][4]). In this paper we address the remaining two items by developing a conceptual framework that allows evaluation goals to be defined and the resultant data to be analyzed in context.

## 2.2. Technology Evaluation in the Literature

A number of interesting papers on technology evaluation have appeared in the software engineering literature over the past few years. In analyzing these papers it is useful to distinguish between two classes of evaluation which we can refer to as product-oriented and process-oriented:

- *Product-oriented*: selecting among a set of products that provide similar functionality (e.g., a new operating system, design tool, or workstation);
- *Process-oriented*: assessing the impact of a new technology on existing practices to understand how it will improve performance or increase quality (e.g., a new design methodology, programming language, or software configuration management technique).

Many organizations have relatively mature evaluation techniques for the product-oriented decisions, and this is reflected in the literature. For example, a number of papers describe general product evaluation criteria (e.g., [5]), while others describe specialized techniques which take into account the particular needs of specific application domains (e.g., CASE tools [6] or testing tools [7]).

Process-oriented evaluation approaches address a more difficult decision faced by an organization which is trying to assess the potential impact of a new technology approach as a whole, and to estimate the impact of that technology approach on the organization's practices, products, and services. Documented approaches in the literature tend to focus on process improvement first (often based around the SEI's Capability Maturity Model (CMM) [8] or ISO 9000 standards [9]), with technology support a secondary consideration.

There are two interesting exceptions to this approach. First, a paper by Boloix and Robillard describes a system evaluation framework that provides high-level information to managers about the characteristics of a software system [10]. The strength of the approach is that it provides a broad snapshot of a system by considering a number of different perspectives (end-user, developer, and operators). However, it aims at providing a rapid high-level review (for example, a complete review of a system takes about an hour and a half). Little detailed insight into the strengths and weaknesses of a technology in comparison with its peers are either sought or revealed.

Second, in work by Bruckhaus a technology impact model is defined and applied to a large CASE tool development [11][12]. The method provides quantitative data on the impact of various CASE tool alternatives based on assessing the number of steps that are required when carrying out a particular scenario. While the approach is valuable, it concentrates on impact purely as a measure of increasing or decreasing the number of process steps, ignoring the intrinsic value of the technology itself, and avoiding any attempt to compare peer technologies for their relative value-added.

## 2.3. Summary

The problem of evaluating software technology has been addressed with a wide range of methods and techniques that address different aspects of the problem. Our examination of this work has found a number of limitations with respect to its lack of a clear framework for defining goals and analyzing results, and its focus on single product instances. Our work has been aimed at overcoming these limitations to provide a conceptual framework that facilitates:

- setting of technology evaluation goals based on understanding the value-added of a new technology approach;
- use of a range of evaluation techniques within an overall framework for synthesizing the disparate results obtained;
- individual product evaluations that concentrate on their distinguishing characteristics in relation to their technology precursors and product peers.

## 3. The Feature Delta Framework: Principles and Techniques

In order to determine the value-added of a technology, it is necessary to identify the features which differentiate a candidate technology from other technologies, and to evaluate these differential features (the *feature delta*) in a specific and well-defined application context. But how does one identify and then assess feature deltas in a disciplined way? Figure 1 provides a high-level depiction of the technology delta framework, an approach we have developed to answer this question.

### 3.1. Technology Delta Principles

The framework embodies a number of important principles. First, of course, is the notion that the potential impact of a technology is best understood in terms of its feature delta. That is, while all of the features of a technology are relevant to understand the technology and how to apply it, to understand the value-added of a new technology it is essential to focus attention on its distinctive

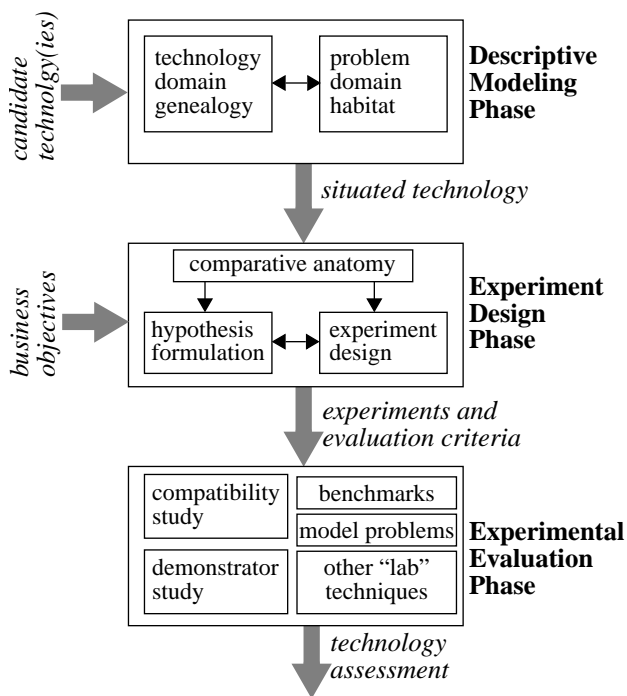


Figure 1: Technology Delta Evaluation Framework

features, i.e., to discover its value-added *with respect to some other technologies*.

Second, it is not sufficient to simply describe the distinctive features of a technology; it is also essential to evaluate these features in a well-defined and sharply-focused usage context. The technology delta framework exhibits a strong bias towards the use of rigorous experimental techniques for evaluating feature deltas. Specifically, hypotheses are formed about how a feature delta supports a defined usage context, and experimental techniques are then used to confirm or refute these hypotheses.

Third, the framework reflects and embraces the inherent complexity, ambiguity and dynamism of the technology marketplace. Technologies are not static; nor do they exist in isolation. A technology's feature set is a function of the way it is intended to be used, and a function of what competing technologies provide, and both change over time. The use of descriptive modeling to analyze and document the inter-dependencies between technologies, and between technologies and their usage contexts, is crucial to a disciplined technology evaluation.

Last, the framework reflects a limited but well-defined objective. There are many factors to consider when evaluating a technology, for example installation costs, market forecasting, organizational resistance, and other non-technical considerations. We view the technology delta framework as addressing a necessary but not

sufficient aspect of technology evaluation. Our desire is to robustly address the technical aspects of determining value-added, and leave questions pertaining to, e.g., technology transition and return on investment, to disciplines better equipped to deal with such issues.

In the following sections we describe the techniques we have used in each of the evaluation phases depicted in Figure 1.

### 3.2. Descriptive Modeling Phase

The descriptive modeling phase addresses feature discovery and impact prediction through the development of *technology genealogies* and *problem habitats*, respectively. The output of this phase is a *situated technology*: models which describe how a technology is related to other technologies, and the usage contexts in which it can be evaluated. Situating a technology in a technology marketplace and in a problem domain provides a basis for identifying feature deltas.

The technology genealogy reflects the fact that new technologies are most often minor improvements of existing technologies (in fact, this assertion represents a capsule summary of the history of technology). As a consequence, if we want to understand a technology in terms of its features, we need to understand the historical and technological antecedents which led to the evolution of the technology. For example, to understand the unique contribution of object-oriented programming technology it is essential to understand its heritage in programming language theory (data abstraction, polymorphism), design theory (information hiding), and simulation and human-computer interaction (modeling real-world entities in software).

However, the features of a technology alone are insufficient to understand value-added; for this, it is necessary to understand how these features will be used and what benefits will accrue from their use—this is the role of the problem habitat. For example, the world-wide web is essentially an integration of several pre-existing technologies (internet, graphical user interface, scripting language, protocols, and hypertext), and can be described largely in terms of the features of these constituents. The potential impact of the web, however, is enormous despite its modest lineage. To predict this impact, the web must be understood in terms of its potential use in electronic publishing, entertainment, and direct sales to name a few.

Our choice of terminology for denoting these models is significant. There is an interesting analogy between the task of describing technology features and usage contexts and the task of describing biological species and habitats. In both cases we define species and technologies through externally visible characteristics: morphology for species

and features for technology. Further, the characteristics we use for descriptive purposes, while not arbitrary, are selected because they are easy to use and are sufficient to distinguish species and assign individuals to species. Finally, for Darwinians, there is a strong correlation between these characteristics and habitat, both of which change over time.

The important point to note is that descriptive models are just that—they are descriptions of assumptions made by the technology evaluator concerning the kinds of features that are of interest, and their relationship to usage contexts that will later form the basis of experimental evaluation. There are no formal tests for the completeness or consistency of these models, nor should we expect there to be. Instead, we should view the descriptive models as a foundation for a rigorous approach to describing technologies, for achieving community consensus on the key features needed to distinguish technologies, and for documenting the evaluation process itself.

Descriptive modeling has analogues in the field of domain analysis [13]. Concepts have been borrowed from feature-oriented domain analysis methods, in particular ODM [14]. From ODM, notions of descriptive modeling, domain genealogy and comparative feature analysis are borrowed whole cloth. Unlike domain analysis, however, the descriptive models generated for technology deltas are not end-products, but are instead used as guides for structuring evaluation experiments and for interpreting the results of these experiments. Thus, we are less concerned with formalizing the notation or the specific form of the work products produced for descriptive modeling than are practitioners of domain analysis.

Genealogies and habitats can be modeled as semantic networks. While we have not yet defined a rigorous ontology for constructing such networks—and believe that it is premature to do so—several different node and link types have proven useful, and these are summarized<sup>1</sup> in Table 1. Nodes in these models represent different ways of thinking about technology, while the links help establish relationships between these views of technology, and form a basis for making assertions about feature deltas. To this end it is also useful to annotate the links with feature lists. For example, consider a hypothetical genealogy of middleware technologies. In such a genealogy it would be plausible to describe SoftBench [15] and FIELD [16] as *peers*; this link could be annotated with the features that distinguish SoftBench from FIELD.

1. Technology analysts should freely extend this ontology, but should be guided by the principle that each element of the ontology should serve to relate technologies, and ultimately yield feature deltas.

**Table 1: Primitives for Genealogy and Habit Models**

Ontology	Form	Interpretation
Technology	Node	A class of functionality, analogous in meaning to “species.”
Specification	Node	A description of a technology for its producers/consumers.
Product	Node	An implementation of a technology or specification.
Peer	Link	Nodes (of any type) that have similar features.
Competitor	Link	Products or specifications that compete in marketplace.
Problem Context	Link	Class of problems addressed by a node (of any type).
Is-A	Link	Product or specification is an instance of a technology.
Part-Of	Link	Bundled and/or separable products or specifications.

Examples of genealogy and habitat models are provided in Section 4. We note here, however, that these models are co-dependent (as is illustrated in Figure 1). That is, it is often necessary to develop these models iteratively, as the problem-domain habitat suggests technologies to consider, while the technology-domain genealogy conditions our ideas about which problems can be addressed.

### 3.3. Experiment Design Phase

The experiment design phase is essentially a planning activity. The output of this phase is a set of hypotheses about the value-added of a technology that can be substantiated or refuted through experimentally-acquired evidence, and a set of defined experiments that can generate this evidence and that are of sufficient breadth and depth to support sound conclusions regarding value-added. As illustrated in Figure 1, three activities are involved in this planning: *comparative anatomy*, *hypothesis formulation* and *experiment design*.

Comparative anatomy involves a more detailed investigation of features deltas. We have found that as hypotheses are formed and experiments are designed questions arise that require a more detailed examination of technology features; conversely, these examinations often suggest hypotheses and experimental approaches. It might appear that comparative anatomy should be undertaken in the descriptive modeling phase as just another kind of feature study. However, we placed comparative anatomy in

the experimental design phase because we have empirical rather than purely descriptive techniques for conducting comparative anatomy:

- reference model benchmarking for qualitative feature descriptions using an *a priori* feature vocabulary; and,
- feature benchmarking for quantitative feature descriptions.

There is no precise definition for what constitutes a reference model; in our usage we consider a reference model to be an annotated feature list. In some domains reference models have already been developed [17], often exhibiting considerably more internal structure than mere feature lists. A reference model provides a ready to hand vocabulary of features and, sometimes, their inter-relationships. By mapping peer technologies to these reference models using profiling techniques [18], feature descriptions can be normalized to a common vocabulary, and surprising questions are sometimes surfaced. For example, two competing technologies may be found to provide complementary rather than wholly-overlapping services; this, in turn, may suggest compatibility experiments that address their combined use in a particular problem setting.

Feature benchmarks are quantitative measures of features in a context-neutral setting: they quantify features in terms that make sense for the feature and the technology, but in a way that is independent of any problem domain. To illustrate, for middleware products we might measure message throughput under various load conditions. Such benchmarks may represent weak hypotheses, i.e., the kinds of load factors that will influence performance, but these hypotheses need not be explicit nor tied to the problem domain habitat. As with reference model benchmarking, feature benchmarking may reveal properties of a technology that suggest hypotheses about its use in a particular problem setting.

Examples of reference model benchmarking and feature benchmarking are provided in Section 4. At the risk of over-extending the biological metaphor, reference modeling and feature benchmarking can be thought of as *in vitro* analysis—“in an artificial environment, outside the living organism.”<sup>2</sup> In contrast, the experimentally-based evaluation phase can be thought of as involving *in vivo* benchmarking—“within a living organism” (i.e., in actual problem settings).

There is not much to say about hypothesis formulation and experimental design other than to note that a hypothesis needs to be carefully crafted to ensure that:

- it is refutable from experimental evidence;

- suitable experimental techniques exist to address it; and,
- the set of hypotheses are sufficient to form a basis for evaluating value-added.

The first item above requires nothing more than discipline and precision on the part of the evaluator. The third item is inherently difficult to validate, and perhaps the best that can be done to ensure completeness is to establish traceability links from hypotheses to problem domain habitat (which incidentally suggests a fairly detailed model of the habitat). The second item requires a familiarity with various evaluation techniques (referred to as “lab techniques” in Figure 1). This topic is addressed in the following section.

### 3.4. Experimental Evaluation Phase

The evaluation phase is where experiments are conducted, experimental evidence is gathered and analyzed, and hypotheses are confirmed or refuted. We have begun to catalogue different experimental techniques that may be useful given certain kinds of hypotheses, requirements for precision, and budget considerations. These are described, below.

Before describing these techniques, we note that there are collateral benefits to employing a hands-on, problem-domain-situated evaluation of a technology beyond the rigorous accumulation of data. Foremost among these benefits is that the evaluation produces as a side-effect a competence in the use of the technology. Moreover, we have found that hypotheses often focus on critical usage issues, e.g., design and implementation issues, and that experiments yield not just confirmation or refutation of hypotheses, but also yield insights into the optimal use of a technology to address these underlying critical issues.

#### *Model Problems*

Model problems are narrowly-defined problems that the technology can address, and that can be extracted from, and to some extent understood in isolation of, broader considerations of an application domain. Examples of model problems might include determining schedulability in real time and manufacturing domains; and integrating applications with independent control loops in the component integration domain. The virtues of using model problems is that they provide a narrow evaluation context, and that they allow a direct comparison of alternative technologies in a way that might be too expensive to do in a broader setting (e.g., through demonstrators).

#### *Compatibility Studies*

Compatibility experiments are the complement of model problems: rather than determining how a single technology behaves in a narrow problem context, compatibility experiments are intended to determine

---

2. From the American Heritage Dictionary.

whether technologies interfere with each other or, optimally, whether they can be effectively used together. Compatibility experiments are particularly useful if a technology might be inserted into an established technology baseline where interactions between the new and established technologies are suspected. Compatibility experiments can also be useful if competing technologies provide disjoint features in addition to overlapping features: it might be useful to determine whether the disjoint features can be used without having overlapping features interfere.

#### *Demonstrator Studies*

Although narrowly-focused experiments may reveal many interesting characteristics of a technology, there is no substitute for trial applications of a technology in a real-life, scaled-up application setting. Although full-scale demonstrator applications can be expensive, a properly designed demonstrator can achieve some of the scale factors needed to stress-test a technology under investigation, while excluding other factors. For example, system documentation could be dispensed with; reliability and performance might be down-played (assuming these were not crucial to any hypothesis). Nevertheless, demonstrators may require a substantial commitment of resources by an organization, and this phase of an evaluation will probably be deferred until it is determined that a technology has a reasonable likelihood of success.

An important point to note is that we distinguish a demonstrator study from what is commonly referred to as a *pilot project*. In our view, pilot projects are intended to be initial full-deployments of a technology. In contrast, demonstrators, as noted above, may elide some otherwise essential engineering efforts. A further distinction between demonstrators and pilots is that demonstrators (at least in the technology delta parlance) remain focused on the distinguishing features of a technology. In other words, a demonstrator may deliberately exaggerate the use of some features of a technology in a problem setting in order to expose its strengths and deficiencies. In contrast, pilot efforts will be driven instead by optimal engineering trade-offs.

#### *Synthetic Benchmarks*

Synthetic benchmarks may only be useful to a narrow range of problem domains. Synthetic benchmarks are primarily useful in examining technologies that have runtime aspects, and where the problem domain can be simulated. For example, in evaluating a middleware technology, test messages can be injected into a command and control system to simulate system operation in normal and crisis modes. Synthetic benchmarks may be difficult to conduct without a pre-existing application that permits such instrumentation. Also, considerable effort may be

required to acquire valid synthetic loads—again, this could require instrumentation of existing applications. Synthetic benchmarks are different from feature benchmarks precisely in the degree to which such problem-domain specific factors are included in the experiment.

### **3.5. Summary of Key Points**

The key idea behind the technology delta framework is that the evaluation of a technology depends upon two factors:

1. understanding how the evaluated technology differs from other technologies; and
2. understanding how these differences address the needs of specific usage contexts for the technology.

Our emphasis is on developing rigorous techniques to address both. These techniques include a range of informal descriptive modeling techniques for documenting assertions about the nature of a technology and its usage context, and more empirical techniques for conducting experiments.

The main virtue of the technology delta approach is that it provides a framework for analyzing and describing inherently subjective aspects of technology evaluation, and for linking subjective assertions to objective data. A secondary virtue is that the experimental methodology also yields a competence in the technology and its use in sharply-focused application settings.

## **4. Application of the Technology Delta Framework to OMA and CORBA**

To illustrate the technology delta framework we describe its application to one technology that has direct bearing on software systems integration<sup>3</sup>, and that has also been the subject of considerable media attention and speculation: the Object Management Group's Object Management Architecture (OMA) [19]. Within recent months implementations of the OMA and its more widely-known component, the Common Object Request Broker Architecture (CORBA) [20], have begun to appear in the marketplace. However, there are many competing claims in the literature and among experts in object-oriented technology, middleware technology, and operating systems, about what is unique about the OMA, and how effective it will be in various problem domains. It was, in part, to answer such questions that we began to systematically investigate the OMA technology delta.

In the following discussion we describe how the technology delta framework has guided our evaluation of

---

3. Our project is centered on software component integration as a topic of investigation.

the OMA and CORBA. We highlight aspects of the application of the framework that resulted in a new or clearer understanding of the OMA. The description concentrates on how the framework has been used and omits many of the results of the evaluations themselves. These details are provided elsewhere [21][22][23].

#### 4.1. OMA/CORBA Descriptive Models<sup>4</sup>

The key elements of the OMA are depicted in Figure 2.

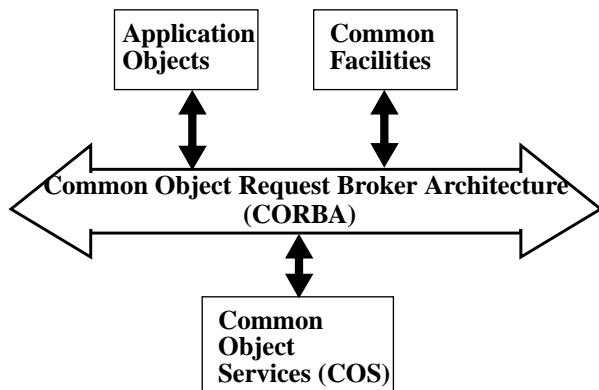


Figure 2: The Object Management Architecture

CORBA is a communication infrastructure that enables clients to locate and make requests of (possibly remote) objects. There are different classes of objects within the OMA. Common Object Services (COS) are objects that provide widely applicable services, e.g., transactions, event management, and persistence. Common Facilities are objects that provide useful but less widely-used services, e.g., electronic mail. Finally, application objects are application-specific, and are not (at this time) a subject for standardization within the Object Management Group. Note that in the following discussion, the OMA acronym will be used to refer to all of the elements depicted in Figure 2, while the CORBA acronym always refers only to the communications infrastructure of the OMA.

##### 4.1.1 OMA/CORBA Genealogy

Figure 3 depicts the OMA genealogy. The most significant point made by the genealogy is that CORBA and the OMA are, to some extent, separable technologies. To those familiar with these technologies, this point is neither subtle nor profound; yet, in our experience, even enlightened technologists confuse the details of the OMA

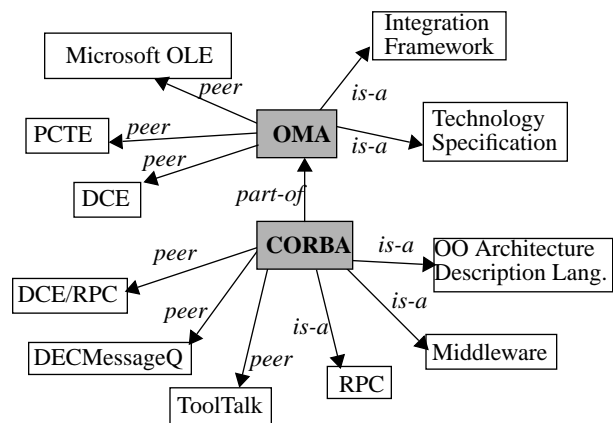


Figure 3: OMA/CORBA Genealogy

and CORBA as well as the role each can play in a system integration effort.

A number of more subtle points are made by the genealogy. The concepts relating to OMA via “is-a” relationships are not necessarily technologies, but rather are evocative of properties that may be significant in evaluating the OMA. To illustrate, the “*OMA is-a technology specification*” relationship implies, minimally, that implementations of the OMA need to conform to an authoritative specification<sup>5</sup>. The “*PCTE is-a technology specification*”<sup>6</sup> also implies conformance rules. However, while both OMA and PCTE [24] specifications define notions of conformance, both do so in different ways. Both the common features of technology specifications and their variant interpretations by peer technologies (e.g., PCTE) are revealing about the OMA.

Another point worth noting concerns peer relationships, and what these relationships imply, and do not imply. First, an assertion that Microsoft’s OLE [25] is a peer technology to OMA implies that there are some features in common between both technologies.<sup>7</sup> The relationship does not imply, however, that there is a substitutability relationship between OLE and OMA—that is an entirely different assertion. This serves as a warning to analysts: the open-ended nature of genealogy and habitat models does not imply a lack of precision in the models; the relationships should be documented and used in a consistent fashion. This is essential if the models are to serve a role in providing a rational foundation for experimentation.

4. In the following discussion we provide a very brief overview of the OMA and CORBA. Readers wishing further details on either should consult [19][20].

5. Other properties, such as the manner in which specifications are defined and evolve, may also be significant and are relevant to this illustration.

6. Some relationships are not shown in the interest of clarity.

7. More accurately, there are common features between both technology specifications. As with the PCTE illustration, understanding how OMA specifications differ from Microsoft’s specifications can be revealing.

### 4.1.2 OMA/CORBA Habitat

Figure 4 illustrates a portion of the OMA habitat. In general a habitat will be more complex than a genealogy, and care must be taken to balance thoroughness against complexity. With this in mind the concepts that are missing

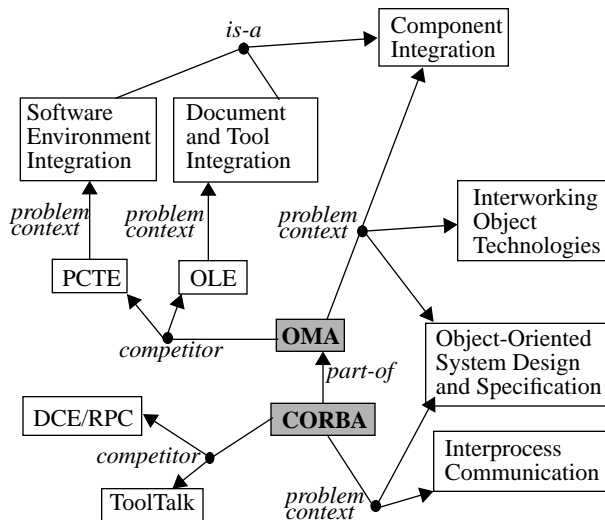


Figure 4: Elements of the OMA/CORBA Habitat

from the habitat are just as significant as the concepts that appear: the habitat is an assertion by the analyst about the features of a technology that are of interest. Thus, although it can be argued that the OMA addresses a problem context concerned with developing object-oriented patterns and frameworks [27], this problem context was not of interest for our investigations. Instead, our interests were in the use of the OMA to support component integration, interworking object technologies, and object-oriented systems development.

The OMA habitat also illustrates that it is often useful to extend the model beyond the immediate technologies of interest in order to obtain a deeper understanding of the technology being evaluated. For example, it is often thought that the OMA and Microsoft's OLE are competing technologies; in fact, there is some truth to this. However, upon a deeper investigation of both (driven by the need to substantiate this assertion in the habitat), it emerged that the OMA addresses component integration in general, while OLE addresses a more limited form of component integration centered on document management. The OMA habitat asserts an "is-a" link between the OLE and OMA problem contexts to capture this subset relationship. The consequence of this observation on our evaluation was to defer experiments relating to the OMA/OLE feature delta until such time as a sufficient range of OMA common

facilities for document management have been defined and are implemented.

On the other hand, if document management architectures were our concern rather than component integration, we might find it useful to conduct a comparative feature study of the OLE component object model and the CORBA object model. This might lead to a better understanding of the strengths and weaknesses that the OMA might experience if it were to form the basis of a document management architecture. Although we did not undertake this more detailed comparison of OLE with the OMA, we did use the genealogy and habitat models to develop more detailed feature comparisons between the OMA and PCTE, ToolTalk [26] and remote procedure call (RPC). Some of these studies are highlighted below.

### 4.2. OMA/CORBA Experiment Design

We conducted an extensive comparative anatomy of the OMA and several technologies identified in the OMA genealogy and habitat. A sampling of our efforts, and their influence on the evaluation, are described below. For fluency of exposition, discussion of concrete hypotheses and experiment designs is deferred to Section 4.3, where actual experiments are described.

#### Reference Models

We developed a number of feature-level reference models that described the OMA in terms of related technologies. Although each technology that appears as either a peer or a competitor in the genealogy and habitat should also appear in a reference model, for practical considerations our investigations focused primarily on feature comparisons of the OMA with PCTE, ToolTalk, and Sun/RPC<sup>8</sup>.

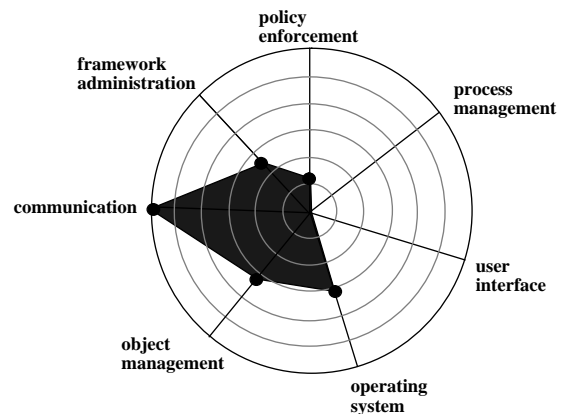


Figure 5: CORBA Mapping to NIST Software Environment Framework Reference Model

8. We did not have an implementation of DCE/RPC available to us.

We constructed one form of reference model by mapping CORBA to a reference model of software environment integration framework technologies [17]. We employed a two-step mapping process of first mapping features found in the reference model to CORBA, and then mapping CORBA features to the reference model. The first step identifies the features CORBA shares with software development environment integration frameworks, while the second step highlights features of CORBA not found in environment framework technology. A summary of the reference model-to-CORBA mapping is illustrated in Figure 5 (with the outer rings corresponding to “greater correlation”). Full details of this mapping are provided in [28].

This kind of rough survey can be very useful in focusing analysis efforts to particular classes of features; the mapping illustrated in Figure 5 led us towards more detailed comparisons of CORBA with communications mechanisms (ToolTalk and RPC, for instance) and object management mechanisms (PCTE, for instance). It is also useful for conducting comparative descriptions of technologies. Consider the feature mapping illustrated in Figure 6, which superimposes a mapping of PCTE onto the CORBA mapping. From this mapping an analyst can postulate that a system might make use of PCTE services for policy enforcement (e.g., security policy) while using CORBA services for inter-object communication. Conversely, the analyst might postulate that there is sufficient overlap between object management services that a hybrid PCTE/CORBA system would need to defer all object management functions to either PCTE or CORBA in order to avoid feature collision.

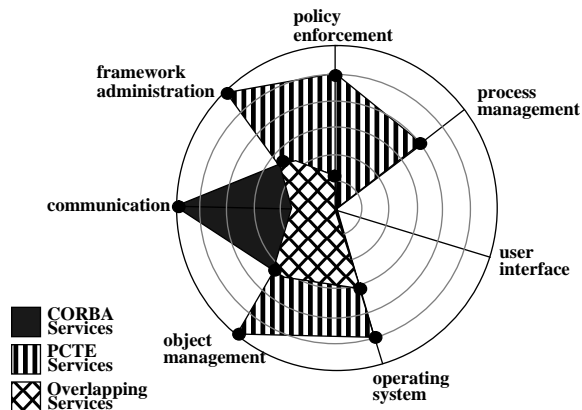


Figure 6: Overlapping and Disjoint Services

However, neither the reference model nor mappings derived from it are sufficiently rich in semantics to support or refute these conjectures. This suggests the design of experiments (in the form of compatibility studies) that are focused on determining the kinds of feature interactions

that might arise between the technologies in a hybrid CORBA/PCTE system (e.g., resource contention), or experiments for determining how features might be effectively combined. Determining how features of the OMA and PCTE can be combined might have practical importance in a large-scale applications that has to satisfy a wide-range of sometimes competing requirements (real time vs. distributed processing, interactivens vs. fault tolerance, etc.). In such cases, knowledge of how and when to delegate features among a range of overlapping technologies can be crucial to meeting application requirements. For example, a real-time or highly secure application might use an ORB to broker communication links that make use of specialized communications hardware and software that are not managed by the ORB.

#### Feature Benchmarks

We found feature benchmarks to be useful for understanding performance variations among different vendor implementations of the CORBA specification, and also for understanding factors in the implementation of object request brokers (ORBS) that influence performance. We also found it extremely useful to compare the performance characteristics of these CORBA implementations with a commercial implementation of RPC, in order to test our assertion that RPC and CORBA are both peers and competitors. Details of these benchmarks and an analysis of the results can be found in [21]. Figure 7 illustrates one such feature benchmark, where we compared the performance of three CORBA implementations and Sun/RPC. Several other performance-oriented benchmarks were created that varied the number of objects, the size of messages, etc. In still

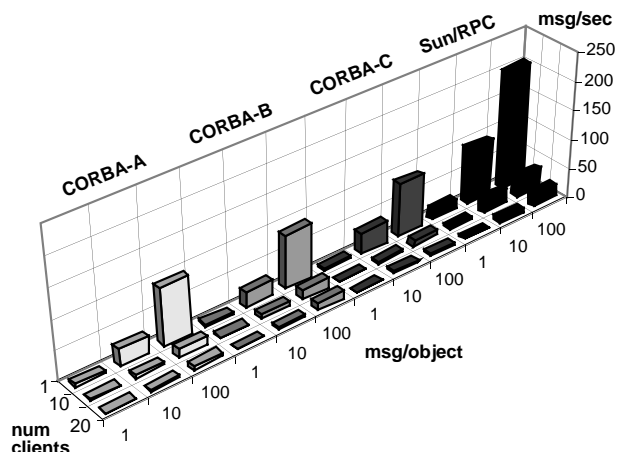


Figure 7: Sample Feature Benchmarks

other benchmarks, additional interprocess communication technology was introduced, such as Sun/Socket.

We also note that there are some practical benefits to writing feature benchmarks quite apart from the quantifiable data produced by the benchmarks. The most significant of these benefits is that feature benchmarks can be viewed as very simple applications (in terms of structure and algorithms). As such, they provide a relatively low-cost means for the analyst to acquire “hands-on” experience with a technology. We found these simple benchmark programs to be quite revealing of several dimensions of the CORBA specification, including code portability, inconsistent concepts visible to client writers and object service providers, deficient administrative and security aspects of ORBs, and issues of the robustness of the ORB and ORB-based applications, interactions between the ORB and the host platform and operating system, associated software development tools, and so on.

### 4.3. OMA/CORBA Experimental Evaluation

The descriptive models and comparative anatomies provided us with a good idea of the OMA feature delta: those features that distinguished OMA and CORBA from PCTE, ToolTalk and RPC (to name a few). In addition, we also had formulated several hypotheses concerning how this technology delta would behave in our selected problem domain (i.e., integration of large-scale, off-the-shelf software components). The next step was to define experimental scenarios that would allow us to apply the OMA feature delta under controlled circumstances, and to test our hypotheses. These experiments and their results are described in high-level terms in the following paragraphs.

Before proceeding we note two points. First, we designed quite a few more experiments than we had the resources to ultimately carry out. We suspect we are not alone in having resource constraints—however, the experiment design process is useful in and of itself, and even those experiments not conducted were revealing. Second, in the interest of space we do not describe all of the experiments we conducted, only a sampling that we believe are revealing of the technology delta method.

#### 4.3.1 Model problems

Model problems can be derived from a number of sources identified in the genealogy and habitat. For example, the CORBA genealogy asserts that CORBA *is-a* architecture description language (ADL). A pre-existing set of model problems for software architecture is available [29]; similarly, a set of model problems for software architecture description languages<sup>9</sup> is also available [30]. Problem contexts are also a ready source of model

problems: interprocess communication and distributed systems problem contexts have many known model problems. For our investigations the model problems of interest were derived from the tool and software component integration problem context.

#### Experiment Overview

The model problem we describe here concerns *architectural mismatch* [31]. The essential problem is that reused, off-the-shelf components embed many integration-time and run-time assumptions about their usage contexts, and often the assumptions made by one component are inconsistent with assumptions made by others. One common manifestation of architectural mismatch concerns the locus of control in an application comprised of many different large-scale components, i.e., who is in charge? There are many more kinds of manifestations.

#### Hypotheses

Architectural mismatch comprises a wide-range of experimental scenarios and hypotheses, and this is still an active area of experimentation within our project—for example, to classify and catalogue techniques for removing architectural mismatches. From past experience, we knew that removing architectural mismatches often requires intricate, low-level code. Given the claims that CORBA could be used to encapsulate and integrate legacy applications, we wished to examine the effect of CORBA on this code. In particular, we wished to determine the sensitivity of architectural mismatch solutions to ORB-vendor features. Our hypothesis was that vendor-specific features would have a minor effect on implementation strategies to architectural mismatch. Supporting or refuting this hypothesis is important for our project so that we can determine to what extent our documented techniques for architectural mismatch-removal are vendor-specific.

#### Experiment Design

Our experiment design involved the integration of a CORBA object implementation with a graphical user interface (GUI) into a single executing process.<sup>10</sup> CORBA object implementations typically have event loops that accept requests for services from arbitrary clients; GUI components have event loops that manage display events. For our experiments we used *Tk/wish* [32] as the GUI component; the object service integrated with the GUI was a simple two-dimensional array that allowed clients to put and get values at specified locations. We used two different commercially available ORBs, and developed a model solution using one ORB and attempted to “port” this solution to the second ORB. This experiment illustrates the essential characteristics of the problem without introducing

---

9. The OMA Interface Definition Language (IDL) is sometimes referred to as an architecture description language—this is the meaning of the placement of ADL’s in the OMA/CORBA genealogy.

---

10. Other experiments were performed where the components did not share a process address space.

extraneous details—the functionality of the final integrated application is trivial while the integration requirements are not trivial and are a valid representation of a potentially challenging integration problem.

#### *Experiment Results*

In brief, the experiment convincingly refuted the hypothesis. We discovered that component integration within a single address space exposed a wide range of ORB vendor-specific idiosyncracies, and the solutions we selected for one ORB were not implementable in the other. Although we knew that the OMA does not support source code portability for object implementations, the extent of the variation between our two model solutions was more dramatic than we had anticipated. In postmortem analysis of the experiment, we determined that it would have been possible to implement a model solution that, with some minor re-programming, would work on both ORBs. However, this uniform solution would be considerably more complex than either ORB-specific solution, and we would only have low confidence that the uniform solution would be applicable to a third ORB with a different set of vendor features.

Regardless of the specific interpretations of our experimental results, a great deal of information about the OMA and its exemplar implementations were derived with modest cost and development effort (on the order of 200 source lines of C++ for each model solution). This level of effort is probably an indicator of a well-constructed model problem.

#### **4.3.2 Demonstrators**

Ultimately, the impact of a technology is felt over the full range of issues related to application development. Experienced software engineering practitioners are well acquainted with technologies that provide leverage early in a software life cycle only to introduce more compelling problems later in the development cycle. The model problem just described was intended to focus on relatively narrow aspects of the OMA. In contrast, demonstrators are intended to reveal the broader characteristics of a technology when it is applied to a representative “complete” problem in an application domain.

We have developed two OMA demonstrators: (1) integration and wide-area distribution of a collection of legacy software components, and (2) an open, distributed-object-based workflow process definition and enactment framework. For reasons of space limitation only the first is described here; details of the second are documented elsewhere [23].

#### *Experiment Overview*

This experiment was conducted collaboratively by the National Institute of Standards and Technology (NIST)

Manufacturing Engineering Laboratory (MEL), Sandia National Laboratories, and the SEI. NIST/MEL has long-standing interest in improving the state of manufacturing technology, in particular involving the use and integration of manufacturing-related software technologies. Sandia is involved in the design and manufacture of high-performance materials, and thus has similar interests from the consumer-side.

The manufacturing processes required to move a product from concept to realization are many and varied, often requiring highly-specialized computing resources. In many cases these resources are independent of the underlying application domain—manufacturability analysis for automotive parts and washing machine parts are quite similar. Some believe that a breakthrough in manufacturing efficiency can be achieved if these “horizontal” resources are freed from their existing “vertical” market confinements, and allowed to develop in the free-market. The challenge is how to re-assemble these horizontal specialties into *virtual enterprises*—specialty companies collaborating in a vertical manufacturing enterprise. Among other things, virtual enterprises require technology infrastructure to support:

- the integration of separately-developed, specialized, computer-aided manufacturing technologies;
- geographical distribution of computing resources, and support for heterogeneous computing environments; and,
- fee-for-service brokering of computer-based services to enable competition for specialized tools and skills.

In short, virtual enterprises will rely increasingly upon information technology such as distributed object technology. This experiment focused on the use of the OMA to support flexible integration and wide-area-distribution of legacy manufacturing engineering design components.

#### *Hypotheses*

In contrast to the model problem described above, our concern in this experiment was less on ORB vendor-specific issues and more on how OMA-defined features supported a specific class of integration problems. Our hypothesis was that, as compared to RPC-based approaches, the OMA feature delta would result in an integration framework that would:

- be more abstract and thus easier to maintain, evolve and standardize;
- be more flexible with respect to component location, coordination, and other run-time concerns;
- better address system-level issues such as data management, persistence, and transactions.

We also were concerned that ORB-vendors did not uniformly implement OMA features beyond CORBA, i.e., CORBA products need not be bundled with implementations of common object services or common facilities (refer to Figure 2). Therefore, our hypothesis further stipulated that OMA services not provided by an ORB vendor could be either partially- or fully-implemented by application developers with only a modest increase in application development effort<sup>11</sup>.

### *Experiment Design*

Our experiment design involved the integration of several Sandia-provided manufacturing engineering design tools. The integration experiment focused on the use of advanced OMA features, rather than a more primitive RPC-like use of CORBA, thus:

- an object model for the engineering design activities comprised by the tools was modeled in CORBA IDL;
- OMA services such as persistence and relationships were used for the object model to support long-running, wide-area distributed design sessions;
- the Sandia tools were “wrapped” to fit into the distributed object model, rather than wrapped to export tool-specific functionality, and distributed across different sites (at the SEI and NIST); and,
- an end-user virtual interface was implemented to make the location and identity of the Sandia tools transparent.

We sketched a paper design of this system using simple RPC primitives to establish a comparative (if hypothetical) baseline; we went so far to give RPC the “benefit of the doubt” that we used CORBA/IDL as an RPC interface specification language, rather than the more primitive specification languages supported by most existing RPC implementations.

As a whole, the experiment design addressed the hypothesis as well as the underlying application domain (virtual enterprises for manufacturing). The total effort devoted to the main design and implementation phase of the demonstrator was approximately six person months, applied by two software engineers over a period of three months. This was substantial enough to construct a non-trivial demonstrator.

### *Experiment Results*

The major elements of the OMA feature delta—object services and an object-oriented interface description—provided an excellent foundation for designing and implementing distributed component-based systems; in large

measure, the first part of our hypothesis was sustained. However, we discovered that in practice developers will require an even richer set of object services than currently defined; it remains to be seen whether vendors will as a rule provide implementations of these services. We were also able to convincingly refute the second part of our hypothesis: we demonstrated that custom implementation of OMA services such as relationship management is impractical, and the use of non-standard services, such as vendor-specific persistence mechanisms, introduces additional problems, such as coding complexity and non-orthogonality with OMA concepts.

As with all of our experiments, a wide-range of practical results beyond the immediate hypotheses were established. From this experiment we discovered a variety of software-architecture related facets of the OMA. In particular, we discovered the affinity of OMA concepts, when applied to wide-area component integration, to a hybrid repository-style [35] and structural style [36]. We were able to demonstrate how this hybrid style addressed many of the technical requirements for flexible, evolvable wide-area component integration; and, further, how it addressed the sometimes ambiguous, inconsistent or incompletely-specified runtime semantics of OMA specifications.

## **4.4. Summary of Key Points**

The OMA is an interesting example of a new class of technologies that many believe will have a major impact on the design and implementation of distributed systems. In order to investigate this technology we have carried out a wide range of studies and experiments that help us to gain insight into the technology’s strengths and weaknesses.

The technology delta framework played an important role in this technology investigation by:

- separating the different evaluation tasks into manageable pieces;
- suggesting an ordering, or methodology, for carrying out different kinds of evaluations;
- allowing the information collected to be considered as part of a larger picture that provides a more complete understanding of the technology.

## **5. Conclusions and Future Work**

While the evaluation of software technologies is seen as a vital task in many organizations, most organizations carry out those evaluations without clearly defining their goals and expectations, and are heavily reliant on the intuition and experience of personnel performing the work. In this paper we have presented the basis for a systematic approach to software technology evaluation focused on establishing the contribution of that technology in relation

---

11. We did not actually believe this, but it is often more useful to state a hypothesis in an affirmative tone and then falsify it, rather than the stating a negative hypothesis and attempt to prove it.

to its peers and predecessors. This approach has been illustrated through its application to the Object Management Group's OMA and CORBA technologies.

A number of limitations and weaknesses of the current framework remain to be addressed. Specifically, we recognize the need for further development of the technology delta framework in the at least the following areas:

- *Additional rigor in the modeling of genealogies and habitats.* The current semantic nets approach provides great flexibility, but at the cost of some precision and repeatability. We hope to further define the modeling language in these areas through further application of the framework.
- *Improved integration of metrics techniques.* The ultimate goal of any evaluation is to define and apply appropriate quantitative techniques that yield objective data on which to base a decision. We hope to examine the technology delta framework as an opportunity for facilitating greater use of quantitative metrics within technology evaluations.
- *Application to a wider set of technologies.* Currently the technology delta framework has only been applied to system integration technologies. While we expect the concepts to be readily transferable to other domains, this assertion must be validated through its application to candidate technologies.

The technology delta framework was created through analyzing the methods that we had been using for a range of technology evaluations in the area of systems integration, culminating with our evaluation of OMA and CORBA. Currently planned is further application of this framework for the evaluation of other classes of system integration technologies (e.g., scripting languages such as Java [37], Python [38], TCL [32]), and for the use of the OMA in domains other than component integration (e.g., real-time, reliable systems). It is expected that these future applications will lead to a more robust framework and evaluation methodology, applicable to a wide range of software technologies.

## Acknowledgments

The SEI is sponsored by the U.S. Department of Defense.

NIST/MEL sponsored the SEI participation in the demonstrator experiment described in this paper.

## 6. References

1. Basili V., Selby R.W., and Hutchens D.H., "Experimentation in Software Engineering", IEEE Transactions on Software Engineering V12, #7, pp733-743, July 1986.

2. Fenton, N.E., "Software Metrics: A Rigorous Approach", Chapman and Hall, London, UK., 1991.
3. Tichy W.F., Lukowicz P., Prechelt L., and Heinz E.A., "Experimental Evaluation in Computer Science: A Quantitative Study", Journal of Systems & Software V28, #1, pp9-18, January 1995.
4. Welzel D., and Hausen H.-L., "A Method for Software Evaluation", Computer Standards & Interfaces V17, #1, pp121-129, January 1995.
5. Information Technology — Software Product Evaluation — Quality Characteristics and Guidelines for their Use, International Standards Organisation (ISO), ISO/IEC 9126:1991, 1991.
6. IEEE Recommended Practice on the Selection and Evaluation of CASE Tools, P1209, 1994.
7. Poston R.M., and Sexton M.P., "Evaluating and Selecting Testing Tools", IEEE Software, V9, #3, pp33-42, May 1992.
8. Paulk, M.C., Curtis, B., & Chrissis, M.B. *Capability Maturity Model for Software*. Technical Report CMU/SEI-91-TR-24, ADA240603, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, August 1991.
9. Schumauch C.H., *ISO 9000 for Software Developers*, ASQC Quality Press, 1994.
10. Boloix, G., and Robillard, P.N., "A Software System Evaluation Framework", IEEE Computer, pp17-26, December 1995.
11. Bruckhaus, T., "The Impact of Inserting a Tool into a Software Process", in Proceedings of the 1993 CASCON Conference, pp 250-264, IBM Toronto, October 1993.
12. Bruckhaus, T., "TIM: A Tool Insertion Method", in Proceedings of the 1994 CASCON Conference, on CD-ROM, IBM Toronto, November 1994.
13. IIscoe, N., Williams, G., Arango, G., "Domain modeling for software engineering," in proceedings of the 13th International Conference on Software Engineering, Austin TX, 1991.
14. Simos, M., "Organizational Domain Modeling", STARS Informal Technical Report CDRL 05156, July 1993.
15. Cagan, M.R., "The HP SoftBench Environment: An Architecture for a New Generation of Software Tools", Hewlett-Packard Journal, V41, #3, June 1990.
16. Reiss, S.P., "Connecting Tools Using Message Passing in the FIELD Environment", IEEE Software. pp57-99, June 1990.
17. NIST, "Next Generation Computing Resources: Reference Model for Project Support Environments (Version 2.0)", NIST Special Publication 500-213, November 1993.
18. Carney, D.J., "Guidelines for Mapping to the Reference Model for Software Engineering Environments", SEI Special Report, CMU/SEI-93-SR-21, 1995, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA.

19. Object Management Architecture Guide, Revision 3.0, Third Edition, Soley, R. (ed), Stone, C., John Wiley & Sons, Inc., ISBN 0-471-14193-3.
20. The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995, Object Management Group, 492 Old Connecticut Path, Framingham, MA, 01701.
21. Wallnau, K., Rice, J., "ORBS In the Midst: Studying a New Species of Integration Mechanism", in Proceedings of International Conference on Computer-Aided Software Engineering (CASE-95), Toronto, CA, July 1995.
22. Wallnau, K.C. and Wallace, E., "A Robust Evaluation of the OMA: An Experimental Case Study in Legacy System Migration", submitted to OOPLSA'96, SEI and NIST technical reports in preparation.
23. Wallnau, K., Long, F., Earl, A., "Toward a Distributed, Mediated Architecture for Workflow Management," in proceedings of NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, May 8-10, see also <http://lsdis.cs.uga.edu/activities/NSF-workflow>.
24. Wakeman, L. and Jowett, J., "PCTE: The Standards for Open Repositories", Prentice-Hall, 1993.
25. OLE Management Backgrounder, Part No. 098-56456, June 1994, Microsoft Corp., One Microsoft Way, Redmond, WA 98052-6399.
26. Frankel, R., "Introduction to the ToolTalk Service", Sun Microsystems Inc., Mountain View, CA., 1991.
27. Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Design Patterns, Elements of Object-Oriented Software", Addison Wesley, 1995.
28. Wallnau, K., "Mapping of CORBA 1.1 to the NIST Reference Model for Frameworks of Software Engineering Environments", Draft SEI Special Report SEI-95-SR-022, Software Engineering Institute, Carnegie Mellon University, September 1995.
29. Shaw, M., Garlan, D., Allen, R., Klein, D., Ockerbloom, J., Scott, C., Schumaker, M., "Candidate Model Problems in Software Architecture", Draft Technical Report 15-675, the Software Architecture Group, Computer Science Department, Carnegie Mellon University, 1995.
30. Kogut, P., Clements, P., "Features of Architecture Description Languages", Draft Technical Report, Software Engineering Institute, Carnegie Mellon University, November 1995.
31. Garlan, D., Allen, R., Ockerbloom, J., "Architecture Mismatch: Why Reuse is so Hard", IEEE Software V12, #6, pp17-26, November 1995.
32. Ousterhout, J., "Tcl and the Tk Toolkit", Addison Wesley, 1994.
33. Christie, A., "Software Process Automation: The Technology and its Adoption", Springer Verlag, 1995.
34. Hollingsworth, D., "Workflow Reference Model", Workflow Management Coalition Specification TC00-1003, Jan. 1995 (email: d.hollingsworth@wsr0104.wins.icl.co.uk)
35. Garlan and Shaw, "An Introduction to Software Architecture," in Advances in Software Engineering and Knowledge Engineering, vol. I, World Scientific Publishing Company, 1993
36. *Structural Modeling: An Application Framework and Development Process for Flight Simulators*, Gregory Abowd, Bass, L., Howard, L., Northrup, L., SEI Technical Report, CMU/SEI-93-TR-14, 1993, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA.
37. Java Home Page, <http://java.sun.com>.
38. Python Home Page, <http://www.python.org>.