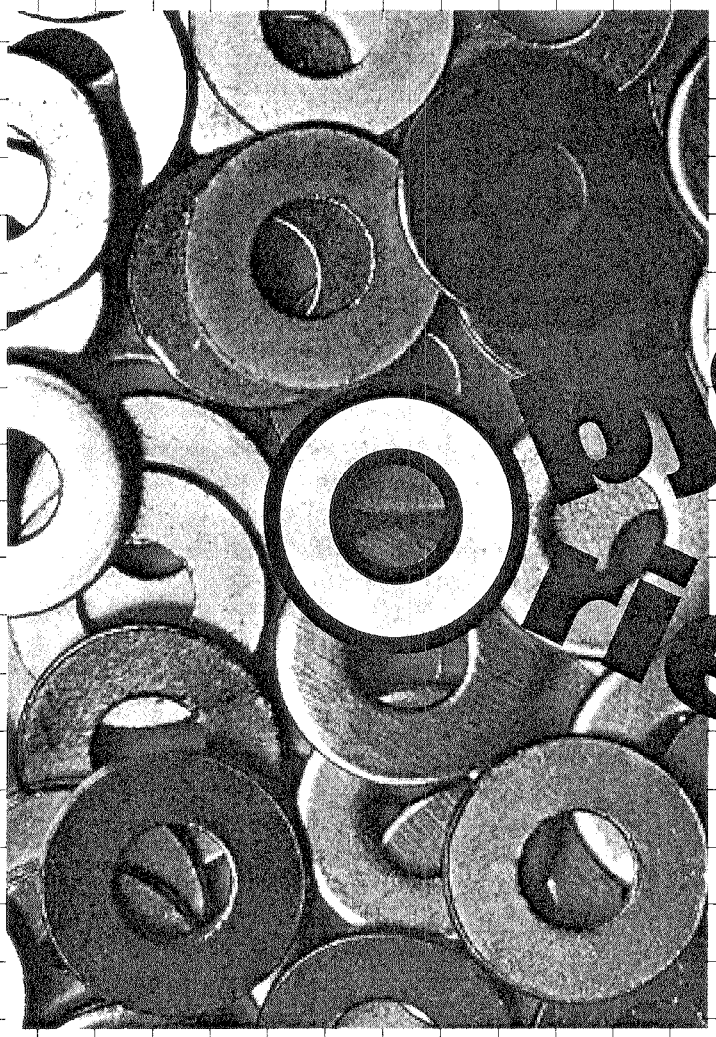
AMERICAN PROGRAMMER
OCTOBER 1994 Vol. 7 no. 10



Object Orientation

AMERICAN PROGRAMMER

CONTENTS

OCTOBER 1994 VOL. 7 NO. 10

EDITORIAL BOARD

October is the month that our attention turns to objects. Why? Simply because it's the time of year when the object faithful in North America flock to the annual OOPSLA gathering (in Portland, OR, this year). Even though there are now a number of terrific object-oriented conferences, OOPSLA is the granddaddy of them all, and it's been our tradition to schedule our object-oriented issue to coincide with it.

Our authors this month examine several important issues associated with object technology. Brian Henderson-Sellers, for example, emphasizes the need for "(1) new analysis and design techniques to replace traditional structured techniques; (2) a re-evaluation of software development project management guidelines. . . ; (3) new OO metrics to complement and, in many cases, supplant the older metrics; and, preferably, (4) use of a new (OO) programming language." He goes on to review the characteristics of several popular OO methodologies, with some recommendations for implementing them in today's application development organizations.

John Dlugosz follows with a look at the "dark side" of object technology, which should be required reading for any IT organization that has fallen under the spell of vendor promises of object miracles. As he points out, "OOP itself is not bad, but getting there can leave you wondering whether the trip is worth it. . . . Understanding that there is a road to OOP that needs to be taken — rather than simply declaring, 'Today we start using OOP' — can make all the difference."

On the other hand, we are beginning to see more and more evidence of "industrial-strength" applications of object technology and more "real-world" success stories. Robert Pemberton gives us one, with a case study of a software development organization that has applied Smalltalk and object technology to an

AS/400 environment. There are no miracles reported here, but lots of good advice and practical experience from which you can learn.

Next, Andrew Wozniewicz examines one of the hot debates in the object world today: the difference between class-based environments (such as the environment provided by Smalltalk and C++) and instance-based environments (such as the environment provided by Visual BASIC, with its VBX "custom controls"). As many *AP* readers are aware, some popular computer journals, such as *Byte*,[1] have recently proclaimed that object orientation has failed and that "components" (the building blocks of the instance-based environments) are the new successor. Wozniewicz provides a balanced review of the two camps and predicts that "the class-based and instance-based approaches will eventually converge into a cohesive, all-encompassing environment. This indeed is what is badly needed if we are truly to reap the benefits of OO technology."

With all of the claims and promises being made about object technology, it's time that we started seeing some credible measurements. Capers Jones, one of our *AP* Editorial Board members and one of the world's leading metrics gurus, provides a set of interesting measurements of object-oriented software in his article. His basic conclusion is that "OO languages [do] offer substantial economic productivity gains compared to third-generation procedural languages, but these advantages [are] hidden when measured with the LOC metric."

One of the central issues of *any* object-oriented methodology involves "discovering" the appropriate objects for an application. Many developers have a tendency to fall back on traditional analysis techniques, such as data flow diagrams, entity-relationship diagrams, and state-transition diagrams, to aid in this discovery process. Our next author, Carl Argila, provides an interesting alternative: an object-discovery approach based on a linguistic analysis of the users' description of their application domain. Argila and I are working on a case-study book that uses this approach, and I have been quite impressed with its success. One of its interesting features is that it doesn't require expensive CASE tools.

Finally, Jim Thomann explores a common dilemma in many business-oriented IT shops: the temptation to believe that object models are simply extended entity-relationship models. Thomann, who works with DBMS vendor Software AG and has watched many organizations transform themselves from SA/IE shops to OO shops, argues that "these two methodologies are very different, and OO needs to evolve further in order to deal adequately with the management and use of business data."

As this issue reaches you, we'll be hard at work on the November issue, which deals with the tidal-wave phenomenon of client-server technology. We're also organizing the final issue of the year, which will focus on "peopleware" issues of recruiting, training, and motivating software professionals. And perhaps even more important, we're beginning to plan the editorial calendar for 1995; if you have ideas or suggestions, please share them with us by phone, fax, e-mail, snail-mail, carrier pigeon, or smoke signals.

Ed Yourdon ★
CompuServe: 71250,2322
Internet: yourdon@acm.org

[1]See, for example, the May 1994 issue.

American Programmer™ (ISSN 1048-5600) is published 12 times a year by Cutter Information Corp., 37 Broadway, Arlington, MA 02174-5539 (617/648-8702 or, within North America, 800/964-8702; Fax 617/648-1950 or, within North America, 800/888-1816). American Programmer covers the software scene, with particular emphasis on those events that will impact the careers and jobs of programmers, systems analysts, and data processing managers around the world. Editor: Ed Yourdon. Publisher: Karen Fine Coburn. Managing Editor: Karen Kunkel Pasley. Production Editor: Rosanne DePasquale. Client Services Manager: Beth O'Neill. List Manager: Doreen Evans. Reprint Manager: Carolyn Licata. Subscription rates are US$395 a year in the United States and Canada, US$495 elsewhere, payable to Cutter Information Corp. Reprints, bulk purchases, past issues, multiple subscription rates available on request.

Finding and Keeping Good Obj

by Carl A. Argila

The formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill.

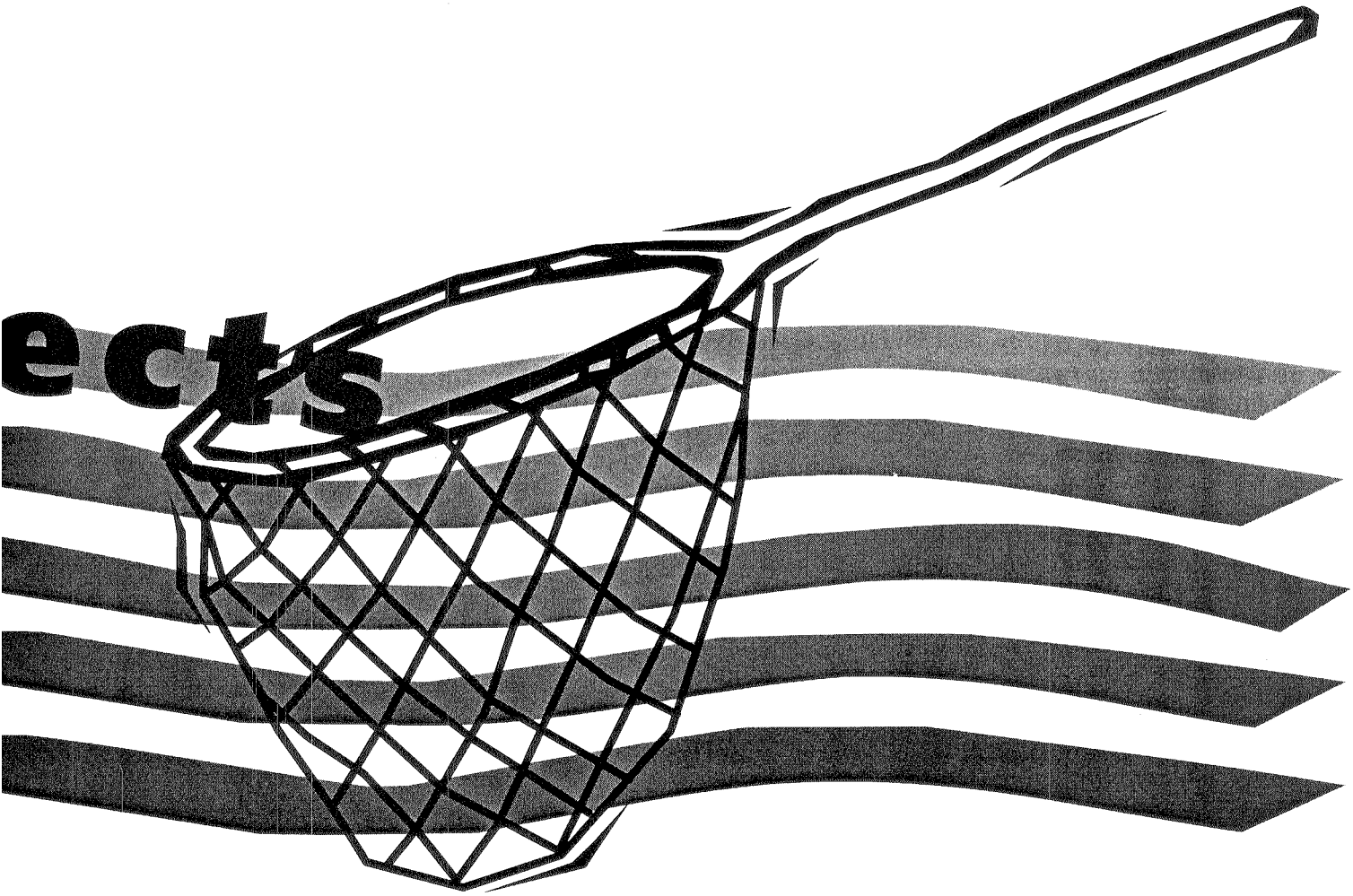
Albert Einstein and Leopold Infeld, *The Evolution of Physics*

True story. A small European country with a large European social system installed a new information system, replacing an existing legacy system that had become enormously expensive to maintain. The massive new system tracks and reports on all of the various government benefits, pensions, programs, and so on. The analysts who were responsible for the specification of the new system decided to use an object-oriented (OO) approach. Their analysis led them to identify objects with names such as "citizen," "pension," "beneficiary," and various other "appropriate" object names.

It was not until some time after the delivery of the system that the failure of this object-oriented analysis became apparent. As one of the systems developers explained, most of the maintenance changes involve changes in so-called "legislative rules." Nowhere, however, is there a legislative rule object — or anything similar. The "secrets" of legislative rules are embedded throughout the system. Therefore whenever one of these legislative rules changes, it usually requires maintenance programmers to make major system-wide changes.

THE RIGHT OBJECTS

OO techniques hold the promise of very significantly improving both the quality and productivity of software development. However, the benefits of object orientation can be reaped only if the "right" set of objects is



identified. An appropriate set of objects, for a given application domain, assures reusability, promotes extendibility, and helps to ensure that the quality and productivity improvements inherent in the OO paradigm are realized. Without a formal method for establishing what the objects are, software developers risk simply "hacking" at the object level.

In this article, I will introduce a number of practical techniques for finding and keeping good objects. Much of what I present in this article is akin to a "chaos effect" for software; that is, identifying "small" things at the start of a process that can have a big impact in the future.

MOTIVATION

As a software engineer, I am concerned with "making things work." It has been frustrating, therefore, not to find practical, down-to-earth object-finding techniques — techniques that can be readily applied to real-world problems. Although there are numerous books on OO analysis (and more coming out all the time!), few go beyond presenting terminology, notation, and the structure of their own unique models. And when it comes to the genesis of the process — establishing an initial set of objects — guidance is superficial at best. Yet the wrong choice of object-classes can have a profound impact on the success of a project.

In grappling with this problem, two insights have revealed themselves. First, objects "know things" (i.e., store data) and "do work" (i.e., have services). Since virtually all of the traditional "tools" of systems analysis are concerned with defining data or specifying processes, traditional

systems analysis tools should, in some way, be very useful for object finding.

In our work together, Ed Yourdon and I have found three traditional systems analysis tools to be particularly useful: data flow diagrams (or variants, including context diagrams), entity-relationship diagrams (or variants), and state-transition diagrams (or variants, including event-response models). These tools capture three different, independent "system views" — process, data, and control. We refer to the application of these tools to the software systems analysis process as 3-view modeling (3VM).

The second insight deals with the fundamental nature of objects. Clearly objects are inextricably bound to application-domain concepts. To find truly good objects, we must be able to identify and define application-domain concepts precisely. Although an understanding of the application domain has always been crucial to software systems analysis, coming to this understanding has typically been an informal and subjective process.

In attempting to learn more about the human conceptual process, I have been led from Aristotle to Ayn Rand [5]. In software systems analysis, humans deal with concepts primarily by means of natural languages, both written and spoken. There is an existing branch of science that deals with the study of language, namely linguistics. Therefore we investigated the use of linguistic tools and techniques for object finding.

There have been some efforts in the past to apply some linguistic principles to software systems analysis [1, 2, 4]. One CASE product, for example, is based on G.M. Nijssen's Information Analysis Model [6]. We refer to the application of linguistic principles to the software systems analysis process as linguistic-based information analysis (LIA).

APPROACH

Figure 1 shows the approach we have taken in applying both 3VM and LIA to the object-finding process. (Both of these techniques will be discussed in greater detail below.) At this point, it should be noted that in Figure 1, 3VM and LIA are activities separate and distinct from OO analysis (OOA). In fact, both of these techniques are useful for traditional systems analysis. It should also be noted that, although not shown in Figure 1, the application of these techniques is iterative. Our principal objective in devising this approach has been to reduce, to the maximum extent practical, the subjective nature of object identification.

CASE STUDY

The techniques presented in this article have been refined through nearly three years of application. For this article, I will illustrate these techniques within the context of a simple case study. The Small Bytes Subscription System (SBSS) maintains subscription, editorial, and special orders information for a "mom and pop" software journal. A complete description of the SBSS may be found in [7].

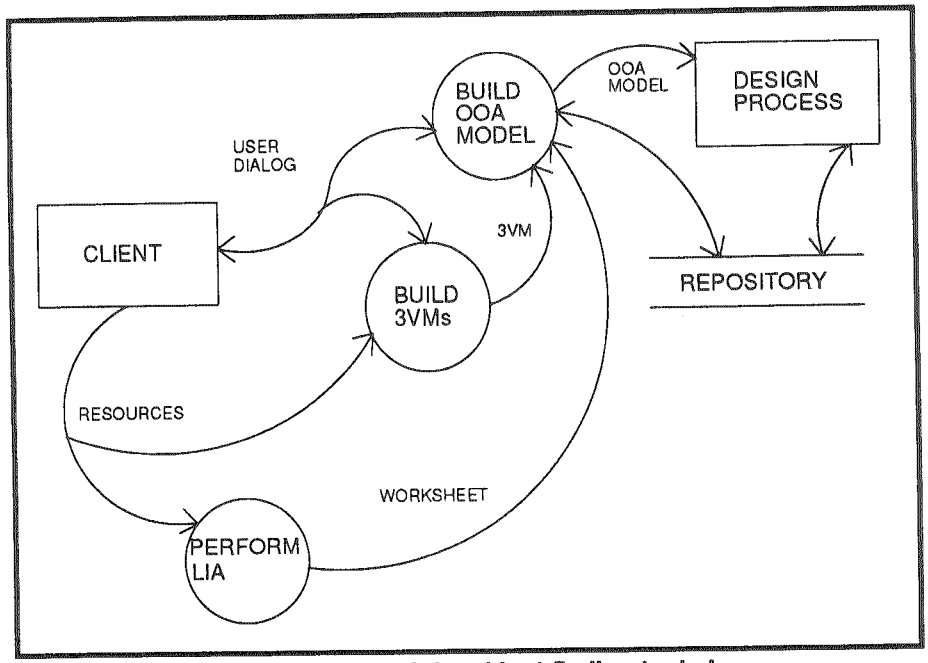


Figure 1: Overview of the object-finding techniques presented in this article

3-VIEW MODELING

We have found the construction of the various 3VMs of a proposed system useful for object finding, as I will discuss below. The use of entity-relationship models, data flow models, and state-transition models in software systems analysis is very familiar, however, so I will not discuss it in depth in this article.

Entity-Relationship Models

It is well known that entity-relationship diagrams (ERDs) are a useful precursor to OOA. The entities strongly suggest objects, and the attributes of those entities represent data that ultimately must be stored by objects. The relationships between entities may suggest the creation of "associative objects." The so-called "cardinality" and "conditionality" of relationships may suggest services that maintain these relationships.

Although the ERD is a very useful object-finding tool, we have found some inherent problems with its use. First, the identified entities may not be relevant application-domain concepts. This is especially true if the analyst has attempted to create entities in a "normal form." Also, the ERD is not useful for identifying objects that do not store data. Objects that recognize the occurrence of events or perform a controlling function, for example, frequently fall into this category.

Data Flow Models

We have found two forms of data flow models to be useful object-finding tools.

First, the context diagram establishes an overall system boundary, which is useful from a systems analysis perspective. The external entities identified in the context diagram represent the ultimate source or destination of the data flows. As such, the external entities are object candidates. The context diagram flows represent the proposed system's inputs and outputs. Any set of objects must therefore account for how these context diagram flows are received, processed, and produced.

When appropriate, a leveled set of data flow diagrams is produced; this model represents a "functional decomposition" of the proposed system into primitive units. These primitive units are referred to as minispecifications or primitive-process specifications (PPSs). The PPSs ultimately must correspond to object methods or services.

State-Transition Models

We have found two forms of state-transition models to be useful object-finding tools.

First, the event-response model (or simply an event list) has proven to be a tremendously useful object-finding tool. This model identifies each "happening or occurrence" the proposed system must recognize and to which the proposed system must produce a preplanned response. The event component of this model helps to identify a set of "event-recognizing" objects; the response component helps to identify a set of "event-producing" objects.

Second, in some very specific cases, it has been useful to create one or more state-transition diagrams (STDs) for a proposed system. (These should not be confused with STDs produced at the object level — so-called "life-cycle" diagrams.) In addition to identifying event-recognizing and event-producing objects (as transition conditions and actions, respectively), such STDs help to identify attributes that maintain "state information."

It should be noted that not all of the 3VM tools are useful for every proposed system. Clearly an analyst would want to use different tools to model an elevator control system than, say, an inventory control system. As an illustration of one of these techniques, Figure 2 shows a partial event list for the SBSS.

LINGUISTIC-BASED INFORMATION ANALYSIS

We have found the 3VM techniques to be quite useful for identifying components of objects; however, they offer no guidance in specifically identifying the "right" set of objects for a proposed system. Judgment, intuition, and insight still rule the object-finding process.

1. Subscription requested
2. Payment received
3. Time to send renewal notice
4. Subscription expires
5. Subscription renewed
6. Time to review "comp" list
7. Special order received
8. Article accepted for publication
9. Article published
10. Time to release issue

Figure 2: Partial event list for the Small Bytes Subscription System

Linguistic-based information analysis, on the other hand, offers considerable guidance in the object-finding process. LIA also helps to identify object components; thus there is some "overlap" with 3VM.

The goal of LIA is to identify application-domain concepts and relationships between these concepts. In this article, I will discuss the two LIA techniques that have worked well for us: phrase frequency analysis (PFA) and matrix analysis (MA).

Both PFA and MA require the identification of a "resource base" or "resource repository." The resource base includes relevant documents, models, software, people, and whatever other resources have knowledge of the application domain or the proposed system. If the application domain has a body of reference materials (textbooks, practices, procedures, etc.), these materials should be included in the resource base.

Other information in the resource base may include transcriptions of interviews, formal or informal system specifications, user's manuals of existing or related systems, printed forms, and even logs. System change requests (or "trouble reports") are an example of the latter; we have found these to be an invaluable resource. All of these resources contain a base of text to which we can apply the LIA techniques.

These techniques are usually applied to some subset of the resource base, depending on what "view" into the application domain or the proposed system the analyst desires. In general, resources that relate to the application domain will yield different results than those that relate specifically to the specification of a proposed system.

Phrase frequency analysis involves searching the selected resource text to identify terms that may represent application-domain concepts. Figure 3 shows a partial listing of the results of such an analysis on the SBSS textual description given in [7]. The creation of a PFA list is a nearly objective process (in fact, it can be almost completely automated). As you can see by reviewing Figure 3, most of the concepts identified might well prove to be irrelevant. We feel that one of PFA's great virtues is its identification of a broad set of application-domain concepts, which allows the analyst to evaluate them and then decide which of them are irrelevant. This is vastly preferable to simply

accepted subscription	constituent copies
accompanied payment	continued subscription
accounting department	contributing author
actual expiration date	converted subscription
additional subscription	copies, constituent
address, corporate	copy
address, correspondence	corporate address
address, home	corporate department
address, subscription	corporation
advisors, board of	correspondence address
agency, subscription service	cost, shipping
agreement, distributor-publisher	country
annual subscription price	country, foreign
article	credit card order
associated site	credit card payment
author	current author
author, contributing	customer
author, current	database
author, past	date, actual expiration
author, prospective	date, expiration
author, would-be	date, expired
author-article track	deleted complimentary subscription
back issue	department, accounting
board of advisors	department, corporate
brown wrapper, plain	direct subscription
bulk shipment	discount, subscription
bureau, subscription service	distributor
check payment	distributor territory
commission, subscription service	distributor, exclusive
company subscription	distributor-publisher agreement
complimentary subscription	division
complimentary subscription query	
complimentary subscription review	
complimentary subscription, deleted	

Figure 3: Partial list of application-domain concepts resulting from phrase frequency analysis (PFA)

overlooking such concepts in the first place.

On the surface, PFA may seem similar to a technique that has been widely used in data modeling, that is, identifying nouns and verbs as candidate entities and attributes. In fact, PFA is quite different. We have found that noun/verb identification is very subjective, both in terms of what is a noun or a verb (since words in English can frequently be interpreted as both), as well as the fact that the perceptions of the analyst usually determine which nouns and verbs are found. PFA, on the other hand, identifies concepts rather than grammatical units. We have found that the creation of a PFA list is not seriously influenced by who prepares the list.

A PFA for any significant application-domain resource may result in the production of a long list of concepts. Many (if not most) of these will be identified as irrelevant and ultimately discarded. Others will become components of an OOA model, including objects. We have found it useful to convert the PFA list into an OOA/OOD worksheet (see Figure 4). This worksheet has been tailored to the Coad–Yourdon version of OOA [3]; it could be adapted, of course, to any version of OOA.

ITEM	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	COMMENTS
Accepted subscription										
Accompanied payment										
Accounting department										
Actual expiration date										
Additional subscription										
Annual subscription price										
Article										
Associated site										
Author										
Author-article track										
Back issue										
Board of advisors										
Bulk shipment										
Check payment										
Company subscription										
Complimentary subscription										
Complimentary subscription query										
Complimentary subscription review										
Constituent copies										
Continued subscription										
Contributing author										
Converted subscription										
Copy										
Corporate address										

(0) Not applicable. Possibly irrelevant, outside the context of the system being specified, etc.
(1) Possible object-class.
(2) Possibly part of subtype/supertype structure. Includes gen-spec and whole-part relationships.
(3) Possibly describes an object-class attribute or instance relationship.
(4) Possibly describes an object service.
(5) Implementation specific. Possible problem domain component item.
(6) Possible human interaction component item.
(7) Possible task management component item.
(8) Possible data management component item.

Figure 4: Portion of OOA/OOD worksheet resulting from the application of linguistic-based information analysis (LIA) techniques

The OOA/OOD worksheet provides a systematic approach for reviewing a lengthy PFA list and identifying an initial set of OOA components. It is at this point that various OOA criteria are applied to the PFA list.

The second LIA technique I will discuss is matrix analysis. MA is a more sophisticated technique, and it is more difficult to perform than PFA. In addition, MA is usually performed only after an initial object identification.

A portion of a two-dimensional MA for the SBSS is shown in Figure 5. The rows and columns of this matrix represent application-domain concepts, usually an initial set of identified objects. The cells in this matrix represent associations between the corresponding row and column concepts. Although an initial matrix could be mechanically constructed from resources, we have found MA to be most useful as a discussion tool. Analysts review and discuss the matrix, cell by cell, identifying application-domain relationships ("business rules"). Analysts may also find new objects — objects that may not have been revealed during the initial PFA.

Clearly an N-dimensional version of this matrix is possible. In practice, however, we have found only two-dimensional matrices to be feasible. As with PFA, we have found the principal virtue of MA to be its systematic and methodical nature.

OBJECT-ORIENTED ANALYSIS

Referring back to Figure 1, both 3VM and LIA are precursors to the OOA process. It is during OOA that the results of 3VM and LIA are consolidated into an OOA model. Using the OOA/OOD worksheet as a guide, the analyst examines the various application-domain concepts vis-à-vis the various components identified by 3VM.

As an illustration of the technique, refer back to the partial event list given in Figure 2. Notice that a number of these events are temporal in nature;

	ARTICLE	AUTHOR	BOARD	COMPLIMENTARY SUBSCRIPTION
ARTICLE		monthly issue consists of 5-10 ARTICLEs, each written by one or more AUTHORs in the software engineering field . . .management is concerned with keeping track of this information, for it wants to avoid publishing more than one or two ARTICLEs from any one AUTHOR in a single year		
AUTHOR	the AUTHORs receive no payment for their ARTICLEs. . .			. . .they do receive a year's free subscription as a token of appreciation for their efforts; if they already have a subscription, then the expiration date is extended for a year
BOARD	The editorial BOARD reviews submitted ARTICLEs. . .	and also makes suggestions to Small Bytes's publisher and managing editor about. . . prospective AUTHORs who should be contacted. . . . . .editorial BOARD of advisors, some of whom may also be AUTHORs from time to time		the editorial BOARD normally serves for a one-year or two-year term, and they too receive a COMPLIMENTARY SUBSCRIPTION to the magazine
COMPLIMENTARY SUBSCRIPTION				

Figure 5: Partial matrix of application-domain relationships

for example, "time to review comp list." We turn to the OOA/OOD worksheet and look for an application-domain concept associated with this event. Although there are several possibilities, we select Complimentary Subscription Review as most appropriate. We decide that Complimentary Subscription Review will be an SBSS object; it is an example of an event-recognizing object. That is, it is an object that encapsulates the knowledge of how the time-to-review event is recognized. It is also an example of an object that is not revealed through entity-relationship modeling, as it stores no data. Other objects identified for the SBSS are shown in Figure 6.

It should again be noted that this is an iterative process; although 3VM and LIA are enormously useful in creating an initial OOA model, this model must still be validated against user requirements. Refinements of this model may well result in the elimination of existing, or the creation of new, objects.

Accepted article
Address
Author
Complimentary subscription review
Customer
Distributor
Expiration warning
Monthly issue
Payment
Published article
Recipient
Renewal
Special order
Subscriber
Subscription
Subscription expiration

Figure 6: List of objects for the Small Bytes Subscription System

SUMMARY

In summary, we have found the combined techniques of 3-view modeling and linguistic-based information analysis to be enormously useful as precursors to OO analysis. We have found that our clients and students readily grasp and apply these tools, and — equally as important — these tools allow analysts to initiate OO analysis in an objective and systematic fashion.

REFERENCES

1 Barbier, F. "Object-Oriented Analysis of Systems Through Their Dynamical Aspects." *Journal of Object-Oriented Programming*, Vol. 5, no. 2 (May 1992), pp. 45ff.

2 Bird, C. "Modeling Gaining Ground but Support Still Patchy." *Software*, Vol. 12, no. 1 (January 1992), pp. 67ff.

3 Coad, P., and E. Yourdon. *Object-Oriented Analysis*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, Yourdon Press, 1991.

4 Nijssen, G.M., and T.A. Halpin. *Conceptual Schema and Relational Database Design.* New York: Prentice Hall, 1989.

5 Rand, A. *Introduction to Objectivist Epistemology*, expanded 2nd ed. New York: New American Library, 1990.

6 Taylor, A., "InfoModeler 1.0." *DBMS*, Vol. 7, no. 8 (July 1994), pp. 30f.

7 Yourdon, E. *Object-Oriented Systems Design: An Integrated Approach.* Englewood Cliffs, NJ: Prentice Hall, Yourdon Press, 1994.

Carl A. Argila has worked for nearly 30 years as a practicing software engineer. Dr. Argila is president of aLigra Systems, Inc., a consultancy specializing in object-oriented methods and CASE training and implementation. He is coauthor with Ed Yourdon of Case Studies in Object-Oriented Analysis and Design *(Prentice Hall, forthcoming).*

Dr. Argila can be reached at aLigra Systems, Inc., 240 N. Jones Boulevard, Suite 205, Las Vegas, NV 89107 (800/347-6903; CompuServe: 71033,2454; Internet: carl@acm.org). ★