Object-Oriented Analysis—Is It Just Theory?

Roy Gelbard, *Bar-Ilan University*

Dov Te'eni and Matti Sadeh, *Tel Aviv University*

System modeling using object-oriented analysis (OOA) and UML diagrams fails to attract practitioners because the costs of engaging in OOA outweigh the benefits.

Research[1–3] and commercial surveys[4] suggest that the object-oriented (OO) approach strongly supports the technical design and coding phases of software development but poorly supports the functional analysis phase. In other words, "the design is good, the analysis is poor."[4] The source of this weakness is often attributed to the fact that "UML representations have not been effective in large-scale projects for context and communication."[4]

Ritu Agarwal and Atish P. Sinha analyzed programmers' assessments of UML's convenience and report that they perceived only the class diagram and the interaction diagram as user friendly.[1] If this is the case for programmers, the perception of UML diagrams' convenience and clarity is even lower among nontechnical systems analysts and customers. Ian Sommerville reaches an even more radical conclusion, noting that it's necessary in many cases to combine UML diagrams with functional processing and flow diagrams for customers to understand the UML diagrams.

> *Developing object models during requirements analysis usually simplifies the transition to object-oriented design and programming. However, I have found that end-users of a system often find object models unnatural and difficult to understand. They may prefer to adopt a more functional, data processing view. Therefore, it is sometimes helpful to supplement object models with data-flow models that show the end-to-end data processing in the system.*[5]

These considerations lead to these basic questions: What is object-oriented analysis (OOA) to the practitioner? How does it differ from functional approach analysis? Has OOA theory, which had a prominent place in the literature of the 90s, taken hold in practice as well? Or has it remained theoretical, and if so, why?

Defining OOA

To address these issues, we first need to define OOA. Although in practice OOA is perceived as an analytical operation that uses UML diagrams, we can't take this as the definition of the theory behind the OOA methodology. Indeed, OOA products are usually UML diagrams, which have become the de facto standard for object modeling and have even been mechanized in various computer-aided software engineering (CASE) tools.[5] Theoretically, though, we distinguish between the methodology and the tools with which we implement it.

CASE tools, as well as the software engineering literature, emphasize the benefits of using UML diagrams during systems analysis to make an easy and smooth transition to design and coding.[5,6] Although researchers assess UML as efficient and effective for OO programming, they raise doubts as to its efficiency and effectiveness for functional analysis.[1,7–11]

Roger Pressman addresses the question of OOA's significance and provides this definition:

> *Any discussion of object-oriented analysis must begin by addressing the term object-oriented. … The intent of object-oriented analysis is to define all classes (and the relationships and behavior associated with them) that are relevant to the problem to be solved.*[6]

In addition, as Martin Fowler and Kendall Scott mention in their prize-winning book *UML Distilled: Applying the Standard Object Modeling Language*, "The UML is the successor to the wave of object-oriented analysis and design methods."[12]

This article examines the practice of OOA in light of Pressman's view. It not only depicts the situation but also points to potential pitfalls and suggests directions for improvement in terms of organizational cost-benefits.

Approach

Pressman provides an operational definition of OOA's required elements:

> *To accomplish this, a number of tasks must occur:*
>
> 1. *… requirements must be communicated between customer and software engineer.*
> 2. *Classes must be identified.*
> 3. *A class hierarchy is defined.*
> 4. *Object-to-object relationships should be represented.*
> 5. *Object behavior must be modeled.*
> 6. *Tasks 1 through 5 are reapplied iteratively until the model is completed.*[6]

Pressman's definition therefore requires three main operations: first, identifying and defining classes—that is, the attributes, methods, hierarchy and relations among the classes; second, modeling potential object behaviors using UML notations; and finally, an iterative life cycle.

Our examination of the actual application of OOA thus focused on

- class-objects, as the first phase in system analysis,
- using UML diagrams to describe the dynamic aspect of class objects (the methods, messages, and relations between classes), and
- exploiting engineering capabilities to achieve an iterative life cycle.

We examined the actual use factors involved with OOA, focusing on organizational cost-benefit considerations. The costs included total time required to complete work, licensing, training and support, and gap solutions (mechanized or manual). The benefits covered time saving at each phase of the life cycle as a result of team support, version management support, change management support, clarity for the customer, clarity for the development team, quality (closure and completeness of the product), and ease of mechanization.

Earlier research highlighted these components in an isolated fashion and didn't link them to the larger picture of cost-benefit considerations.[2,4,7,8,13] In contrast, we incorporate these features while recognizing the putative impact of external variables such as policy, which can be dictated by the customer, business partners, or the organization itself as part of the organizational or project culture.

Method

We conducted semistructured interviews with software project managers, senior systems analysts, and IT developers concerning the system analysis of 54 software development projects, which were all planned to be programmed in OO languages and tools. All projects had already accomplished system analysis. Semistructured interviews let interviewers adjust the questions according to the terms and terminology particular to various projects. This lets us analyze projects from a wide variety of fields, system types, and development procedures.

Table 1 presents the distribution of the 54 samples. Although the projects differed in terms of the developing body's internal best practice, CMMI, ISO 12207, or MIL-STD 498 development

Researchers raise doubts as to UML's efficiency and effectiveness for functional analysis.

Table 1

Project distribution

Sector	High-tech (37%), software house (29%), governmental (12%), start-up (12%), services (10%).
Project size (person-years)	Up to 1 (25%), 1–3 (10%), 4–10 (33%), 11–50 (16%), more than 50 (16%).
Team size (members)	Up to 3 (31%), 4–10 (42%), 11–50 (21%), more than 50 (6%).
Product type	Tailor made solution (69%), off-the-shelf product (31%).
System type	Massive user interface and database (59%), real-time (23%), combined (18%).
Programming language	.NET (42%), C++ (31%), Java (25%), Delphi (2%).
Development standards	ISO (62%), in-house best practice (33%), CMMI (5%).
Use of computer-aided software engineering (CASE) tools	CASE tools available but not used (45%), CASE tools not available (28%), CASE tools used (27%).
Object-oriented background	Professional course (30%), academic and professional courses (26%), self-education (23%), academic course (19%), none (2%).

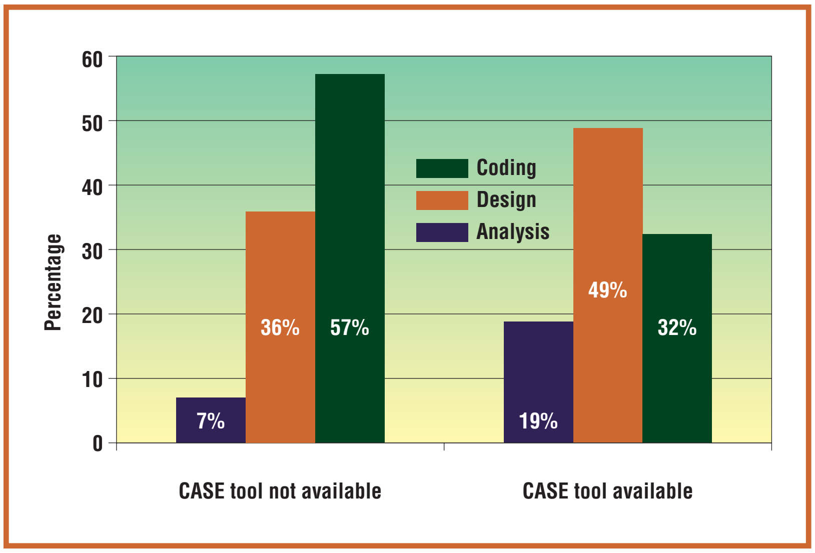


Figure 1. Class identification and definition. The chart shows the earliest phase in which we identified or used classes. When computer-aided software engineering (CASE) was available, classes were defined earlier, but even then only a fraction of the projects identified classes at the analysis phase.

standards, all were defined as OO projects, from the initiating phase to the analysis and development phases.

Each interview had four stages. First, we presented the study and the interview process to create a relationship with the interviewees, reduce their concerns, and assure confidentiality. Second, we asked general questions about the project: the nature of the project, phases and products, concepts, and terminology. At this stage, we refrained from posing questions that could have been perceived as overly invasive. Third, we asked what-how-why questions based on the phase of the project and the required questions at each phase. We proceeded gradually from "what" and "how" to "why," delaying questions interviewees could see as judgmental as much as possible. This gradualness also helped deal with the interviewees' natural tendency to present a "pretty picture." In the final stage, we focused on questions about the project and interviewee that could be perceived as personal and invasive. However, by this stage, the interviewees had made some investment in the survey process and found it easier to answer more probing questions.

Results

Our study focused on three basic requirements of OOA: class identification and definition, use of modeling language to characterize objects; and iterative life cycle.

Class Identification and Definition

Figure 1 presents the phase in which we identified and defined the analysis, design, and coding classes in each project. We found a significant difference between projects that used CASE tools and projects that didn't. So, Figure 1 presents the distribution of class identification and definition in these two categories separately.

We found no correlation between this distribution and other project distributions, except project size (in person-years). The projects with a 3–10 person-year scope identified and defined classes at earlier phases, while smaller and larger projects left identification to later.

Use of Modeling Language

Table 2 presents the projects' use of UML diagrams and additional modeling devices, including flow charts and data-flow diagrams. Each row shows the percent of projects that used the specified diagram. We made no distinction between basic or advanced diagram use.

It appears that users or developers might not find symbolic language—for example, modeling diagrams—sufficiently clear, requiring supplemen-

Table 2

Diagram use

Diagram	Use (%)	Remark
Use case	6.7	
Class	56.9	
Activity	25.5	Logical level
State transition	17.6	
Sequence	25.5	Physical level of activity, isomorphic to collaboration
Collaboration	13.7	Communication diagram according to UML 2
Package	33.3	Logical level, isomorphic to component
Component	—	Physical level, isomorphic to package
Deployment	—	
Flow charts	25.5	
Data-flow	9.8	Supports use case functionality

tal textual descriptions. The degree of accuracy that diagrams achieve defines to what extent text is needed to complete the description. Table 3 presents the level of text support used to complement the modeling language's visual components. The table refers to the entire set of diagrams each respondent used. These results show that diagrams aren't clear enough to the user or to the developer and that substantial textual clarifications usually support visual modeling—for example, diagrams including UML diagrams.

Table 3

Level of textual support

Textual support level	Use (%)
Diagrams with a few remarks	2.0
Text and graphics presented together	35.3
Mostly text diagrams when needed	29.4
Text with a few diagrams	25.5
Text only	7.8

Table 4 presents the distribution of modeling diagrams into six categories according to their functionality in system modeling, noting the degree to which the surveyed projects used each category. We found a negative correlation between the degree of use and the amount of textual description. In other words, the higher the degree of use, the fewer textual descriptions. We found the greatest amount of verbal description in use case and data-flow diagrams (the main-functionalities category), and the lowest in the data item category, which includes class diagrams and entity relation (ER) diagrams. A closer examination of ER and class diagram use indicated that most projects actually used class diagrams as ER diagrams, modeling only the static aspect, not the dynamic aspect. Another finding regarding the configuration category is that nearly all projects were modeled during design and not during analysis.

Iterative Life Cycle

The results show that 84 percent of the projects were carried out in a single iteration—that is, according to the waterfall model and contrary to the OO approach. Only 16 percent of the projects were developed in a process that defined multiple iterations.

Cost-Benefit Considerations

We asked the respondents to assess the effectiveness of the methodologies and the tools they used during various phases of work, such as

- the degree of clarity of the various UML diagrams to the customer, analysis team, and development team.;
- the degree of accuracy of the description achieved with the diagrams, and to what extent text was needed to complete the description;
- the amount of investment needed for method use, ease of learning, and description using the diagrams;
- the amount of team support, in change management and version management;
- the amount of support for various software engineering operations, code generation, and reuse; and

Table 4

Percentage and type of diagrams used in the six modeling categories

Modeling Category	Use (%)	Diagrams used
Data item	56.9	Class diagram, entity-relation diagram
Process	39.2	Activity, sequence, and collaboration diagrams
Logic	25.5	Flow charts
Actions	17.6	State transition diagram
Main functionalities	13.5	Use case, data-flow diagram
Configuration	33.3	Package and component diagrams

- the reason for using the method: the availability or unavailability of the supporting CASE tool.

The following quotes represent the most typical responses to these questions:

> *Drawing diagrams takes a long time and the project must follow a strict and busy schedule, which leaves no time to invest in diagrams.*
>
> *The customers don't understand the complex diagrams anyway, and even if they do understand the outline of symbols, they still have difficulty understanding the significance of the described functionality models.*
>
> *The diagrams alone never succeed in describing the system accurately.*
>
> *In any case, additional aspects are discovered during the coding phase that the analyst did not anticipate.*
>
> *A detailed description of the screens constitutes a sufficiently clear and precise definition of the system (in projects in which the user interface is a chief component of the system).*

Only 27 percent of the projects used CASE tools. CASE tools were available in 72 percent of the projects, but 45 percent chose not to use them. The remaining 28 percent didn't make the CASE tools available to the project. The most common answers regarding this point included these reasons:

> *The customer isn't interested in a formal explanation of the diagrams, and a detailed verbal explanation is necessary in any case.*
>
> *The customer's demands and our obligations to him are defined as a functional language, and no one cares how it's modeled at the analysis and design phases; the main thing is that the system provides the necessary functionality.*
>
> *Use of CASE tools doesn't shorten the process because the real difficulty is in defining each of the demands and not in the modeling method used to do so.*

Discussion

We made three observations regarding the data. First, only in 7 to 19 percent of the projects were the classes identified—partially or comprehensively—during analysis. In the other projects, class identification occurred during design and coding (32 to 57 percent during coding).

Second, during analysis, the practitioners surveyed used modeling tools to characterize five main components: data-item modeling (56.9 percent), process modeling (39.2 percent), business-logic modeling (25.5 percent), action modeling (17.6 percent), and hierarchic distribution according to system functionalities (13.5 percent). We found the first two components at not only a higher level of use compared to the three latter components but also a higher level of clarity. As Table 3 shows, diagrams aren't clear enough to the user or the developer. This is why massive amounts of textual description usually support visual modeling—for example, diagrams, including UML diagrams.

Finally, only 16 percent of the projects were developed in a process that defined the number of iterations. The rest of the projects were developed in a single iteration—that is, according to the waterfall model and not the object approach.

Thus the field, which is apparently fluent in OOA principles (only 2 percent of the interviewees had no training in OOA methodology), doesn't implement this approach in actual work situations, despite the availability of CASE tools in 72 percent of the surveyed projects. Is this due to the low percentage of CASE tool use in OOA methodology? Or, is it due to insufficiently developed technology? Or, does it arise from comprehension problems related directly to the approach and its representation using UML diagrams?

Brian Dobing and Jeffrey Parsons report that both customers and programmers experienced difficulties in understanding some of the modeling diagrams.[8] Customers need to critique the result of the characterization, whereas programmers must develop the programs according to the modeling diagrams. Several researchers have attempted to overcome the lack of integration between objects and processes.[14,15] Still, David Avison and Guy Fitzgerald define the current state as a post-methodology era, in which methodologies lack a great deal of capabilities and are greedy exploiters of resources.[16] The latter feature directly impacts the cost-benefit ratio

Overall, the common perception is that CASE tools are at a reasonable level of technological maturity and user friendliness. So, the cost-effectiveness argument is more relevant to the approach these tools implement rather than to the technology. In software development, as in other fields of development, projects aspire to work effectively by reaching goals economically and in a "reasonable" time frame. In software development, human resources are the dominant cost factor. A clear goal definition is therefore crucial, but we could not find explicit statements of project goals. We elaborate on this matter in light of our survey results and suggest a number of practical recommendations.

Clarity, Integrity, and Completeness as Keys to Cost-Benefit Analysis

Unlike a construction project's architectural planning phase, OOA methodology lacks clarity and comprehensiveness (except for, perhaps, the data component).[17] Moreover, the products of architectural planning—lot plan, construction plan, electrical plan, water, air conditioning, carpentry, gardening, and quantities itemization—successfully meet these criteria:

- *Clarity and integrity.* Both the customer and the engineer can explicitly understand them. The customer will demand that they reliably correspond to his other desires, while the engineer will need to complete the engineering planning of the building.
- *Completeness.* An architectural plan promises full coverage of all building components, so there's no need for additions or explanations at later implementation phases.

OOA products don't meet these criteria.

Moreover, OOA products fail to provide controls and measures to assess whether the criteria have been met. The condition for clarity-integrity, as implemented in the database component (using the normal forms), is the elimination of uncontrolled data item redundancy. UML diagrams are beset with duplications, which not only threaten the clarity and explicitness of the object definition but also waste valuable time and human resources.[1,8,11,13]

UML has several redundancies, for example:

- activity diagrams and sequence diagrams both represent the same functional actors and the same functional chronology;
- sequence diagrams and communication diagrams both represent the same interactions among the functional classes; and
- communication diagrams and the operations section in class diagrams both represent the same functionality of the classes.

As regards completeness, both the functional and the OOA approach fail to provide visual-modeling tools for the user interface, business logic, hardware and software infrastructures, and technical constraints.

So, the low use of OO CASE tools and OOA methodology casts doubts on the methodologies more than on organizational policy.[2] First and foremost, organizations' policies in regard to methodologies and tools reflect cost-benefit considerations. In the absence of clarity, integrity, and completeness of all system components, and in the absence of decomposing and stopping rules for all system components, the ability to use these tools and methodologies effectively and economically has little appeal for the organization or project.

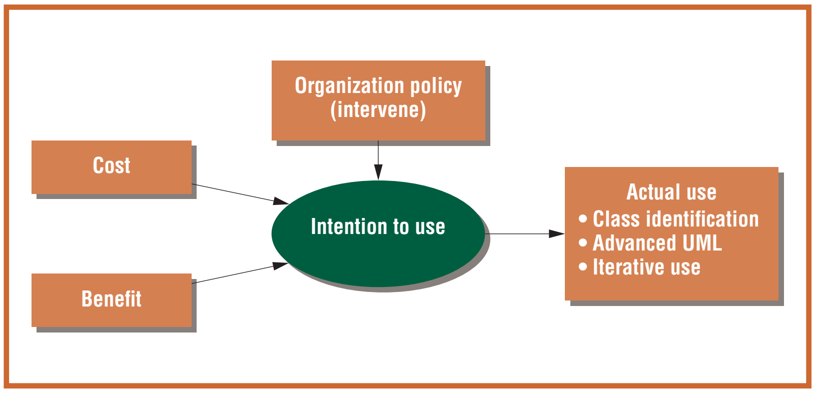
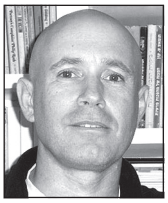


Figure 2. The proposed model. The actual use of OOA is an outcome of the intention to use OOA, which is affected by cost variables, benefit variables and an intervene variable related to the organization policy.

About the Authors



Roy Gelbard is head of the Bar-Ilan University Graduate School of Business Administration's information system program. His work involves several related areas of information systems: data and knowledge representation, data mining and recommendation systems, methodologies of system analysis, and integration of software engineering and project management tools. Gelbard has a PhD in information systems from Tel-Aviv University. Contact him at gelbardr@mail.biu.ac.il.

Dov Te'eni is a professor in the information systems department at Tel-Aviv University. He studies several related areas of information systems: human-computer interaction, computer support for communication, knowledge management and systems development. His research usually combines model building, laboratory experiments, and development of prototypes such as Spider and kMail. Te'eni has a PhD in IS and computer science from Tel-Aviv University. He coauthored Human-Computer Interaction for Organizations (Wiley 2001), and wrote a novel titled *Let's Congress* (visit letscongress.com). He has served as senior editor for the *Management Information Systems Quarterly* and associate editor for the *Journal of the Association for Information Systems, Information and Organizations*, and *Internet Research*. Contact him at teeni@post.tau.ac.il.

Matti Sadeh is a senior software engineer and systems analyst in a large software house. He has a MSc in information systems management from Tel-Aviv University. Contact him at sadematti@hotmail.com.

We suggest the model in Figure 2 for further exploration of the intention to use and the actual use of any analysis theory, methodology, or tool. Regarding OOA theory, we suggest that the "actual use" variable should be composed from the three core constructs: class identification, advanced UML (to clearly distinguish between "pure" OO use and other uses that are usually quite similar, such as ER diagrams and class diagrams), and iterative use.

Suggested Directions for Improvement—Less Is More

Given the need to meet cost-benefit constraints, the analysis phase should focus on only four components and exploit them by relying on previous experience and theory. The first component is the organizational-interactions. Practitioners rarely exploit use-case diagrams because these diagrams don't analyze crucial components such as organizational structure, knowledge authority and responsibility domains, production capacity, and workloads. It's no surprise that these issues are characterized chiefly by free text, which reduces clarity and consistency in the specification. For example, Dov Te'eni applied communication theory to match systems design to the particular organizational structure. Further research is imperative.[18]

The second component is data items. Modeling data items with class diagrams and ER diagrams is the most effective approach. Given the amount of detail found in the diagrams in practice, we suggest restricting the modeling during analysis to entities (without methods).

The third component is business processes. In OOA, business process modeling is scattered over a wide range of diagrams. Each type of diagram presents a separate aspect of the process, with no organized way of forming an integrated picture of the processes, the business logic, or the way they're merged into the organization's business-service chain. So, modeling this domain with the OOA components isn't effective. A project that meticulously completes the various OOA diagrams in this domain will find that it's performing the same analysis over and over, as many times as the number of diagrams used. Applied research that injects organizational theory into the modeling of business process is needed to produce useful simulations of the organization.

The final component is user experience. The OOA methodology doesn't include modeling of the user's interaction with the system. Yet, such modeling that is based on theories of human cognition, affect and behavior is necessary for determining effective design.[19] As we indicated earlier, the current form of use cases is too weak to represent the user's physical, cognitive, and affective requirements and provides no guidance to effective user interface designs.

We've argued that for OOA to be worthwhile, costs should be reduced by concentrating on four major components, namely organizational relationships and interactions, data items, business processes, and user experiences. The benefits of each component should be enhanced by going deeper into the analysis that builds on relevant psychological, organizational and social theories.

References

1. R. Agarwal and A.P. Sinha, "Object-Oriented Modeling with UML: A Study of Developers' Perceptions," *Comm. ACM*, vol. 46, no. 9, 2003, pp. 248–256.
2. J. Iivari, "Why Are CASE Tools Not Used?" *Comm. ACM*, vol. 39, no. 10, 1996, pp. 94–104.
3. D.J. Reifer, "Is the Software Engineering State of the Practice Getting Closer to the State of the Art?" *IEEE Software*, vol. 20, no. 6, 2003, pp. 78–83.
4. J. Duggan, "Modeling and Methods: The Keys to the Quality Kingdom," *Proc. Gartner Analysis and Design Summit*, Gartner, 2001.

5. I. Sommerville, *Software Engineering*, 8th ed., Addison-Wesley, 2007.
6. R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed., McGraw-Hill, 2005.
7. I. Davies et al., "How Do Practitioners Use Conceptual Modeling in Practice?" *Data and Knowledge Eng.*, vol. 58, no. 3, 2006, pp. 358–380.
8. B. Dobing and J. Parsons, "How UML Is Used," *Comm. ACM*, vol. 49, no. 5, 2006, pp. 109–113.
9. C.F.J. Lange, M.R.V. Chaudron, and J. Muskens, "In Practice: UML Software Architecture and Design Description," *IEEE Software*, vol. 23, no. 2, 2006, pp. 40–46.
10. D. Te'eni, R. Gelbard and M. Sade, "Increasing the Benefit of Analysis: The Case of Communication Support Systems," *Didaktik der Informatik*, Lecture Notes in Informatics, vol. 92, Gesellschaft für Informatik, 2006, pp. 13–27.
11. A. Zeichick, "Modeling Usage Low: Developers Confused about UML 2.0", *Software Development Times*, 15 July 2002; www.sdtimes.com/content/article.aspx?ArticleID=26637.
12. M. Fowler and K. Scott, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.
13. J.C. Knight et al., "Why Are Formal Methods Not Used More Widely?" *Proc. 4th NASA Langley Formal Methods Workshop* (LFM 97), NASA, 1997, pp. 1–12.
14. D. Dori, *Object Process Methodology—a Holistic Systems Paradigm*, Springer, 2002.
15. P. Shoval and J. Kabeli, "FOOM: Functional and Object-Oriented Analysis and Design of Information Systems: An Integrated Methodology," *J. Database Management*, vol. 12, no. 1, 2001, pp. 193–210.
16. D.E. Avison and G. Fitzgerald, "Where Now for Development Methodologies?" *Comm. ACM*, vol. 46, no. 1, 2003, pp. 79–82.
17. B.C. Bjoerk, "A Unified Approach for Modeling Construction Information," *Building and Environment*, vol. 7, no. 2, 1992, pp. 173–194.
18. D. Te'eni, "The Language-Action Perspective as a Basis for Communication Support Systems," *Comm. ACM*, vol. 49, no. 5, 2006, pp. 65–70.
19. D. Te'eni, J. Carey, and P. Zhang, *Human-Computer Interaction: Developing Effective Organizational Information Systems*, John Wiley & Sons, 2007.

ADVERTISER INFORMATION
JANUARY/FEBRUARY 2010 • *IEEE SOFTWARE*

Advertiser	**Page**
Seapine Software	Cover 4

Advertising Personnel
Marion Delaney
IEEE Media, Advertising Dir.
Phone: +1 415 863 4717
Email: md.ieeemedia@ieee.org

Marian Anderson
Sr. Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
Sr. Business Development Mgr.
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Recruitment:

Mid Atlantic
Lisa Rinaldo
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: lr.ieeemedia@ieee.org

New England
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Southeast
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest
Darcy Giovingo
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: dg.ieeemedia@ieee.org

Northwest/Southern CA
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Japan
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe
Heleen Vodegel
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com

Product:

US East
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: db.ieeemedia@ieee.org

US Central
Darcy Giovingo
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: dg.ieeemedia@ieee.org

US West
Lynne Stickrod
Phone: +1 415 931 9782
Fax: +1 415 931 9782
Email: ls.ieeemedia@ieee.org

Europe
Sven Anacker
Phone: +49 202 27169 11
Fax: +49 202 27169 20
Email: sanacker@intermediapartners.de