CONF-9010307--Suppl.

PNL-SA--18945

DE91 006724

"FINDING THE OBJECT" PROCEEDINGS ADDENDUM

October 1990

M. A. Whiting
D. M. Devaney

Presented at the
Object Oriented Programing Systems
Language and Application / European
Conference on Object Oriented Programing '90
Ottawa, Canada
October 22, 1990

Work Supported by the
U. S. Department of Energy
under Contract DE-AC06-76RLO 1830

Pacific Northwest Laboratory
Richland, Washington 99352

Finding the Object

An invited-participation, full-day workshop

OOPSLA/ECOOP '90
October 22, 1990
L'Orangerie, Chateau Laurier
Ottawa, Canada

Coordinators:

Mark A. Whiting
D. Michael DeVaney
Pacific Northwest Laboratory
PO Box 999
Richland, WA 99352
phone: (509) 375-2237
fax: (509) 375-3641
email: whiting%snuffy@pnlg.pnl.gov

Introduction

The purpose of this workshop was to discuss *finding the object* -- that is, how software engineers imagine, invent, design, or recycle objects and their behaviors for object-oriented software engineering. The workshop organizers (and, as we subsequently discovered, several of the workshop participants) felt that this issue is crucial to successful object-oriented software engineering (after all, finding objects is what the process is all about, isn't it?). Unfortunately, when previous workshops have had the opportunity to review and discuss techniques practitioners use to find objects, too often the results were heated debates on "what *is* an object?" which becomes all consuming. We believed that, given appropriate control over the question of which kind of "object" is being discussed (which meant *tell us* what object you are trying to find, then tell us your method), a workshop to concentrate on techniques for finding objects would be quite appropriate to the ECOOP/OOPSLA forum[1].

Two situations at the organizers' workplace, Pacific Northwest Laboratory, motivated the workshop proposal. First, we have recently been attempting to hire persons versed in object-oriented technology. Having interviewed several candidates, we found some recurrent themes in our interactions:

- Most candidates had a very difficult time explaining how they went about finding objects, given their own definition of what "objects" they used.
- Most candidates could quote the Booch method [Booch 1991] for finding objects via the noun, verb, etc., extraction from requirements specifications. However, most then turned around and admitted that either the specifications usually weren't sufficiently complete to allow this process to be effective, or that they intuitively (thought they) knew how to pick out objects for the first implementation or prototyping attempt.
- Most candidates decried the lack of reference material that would have assisted them. Generally, they only had available early Booch reference material and Shlaer and Mellor's analysis work [Shlaer and Mellor 1988].

[1] While discussing the workshop with a colleague, however, he quickly noted that there seemed to be certain similarities between "Finding the Object" and *Where's Waldo* [Handford 1987]-- a children's book where the object is to find the intrepid traveller Waldo in various situations: on the beach, at the railway station, on the camp site. Not to carry a facetious statement too far, the analogy becomes relevant when you consider Meyer's statement: "the objects are just there for the picking" or Shlaer and Mellor's statement: "Identifying objects is pretty easy to do. Start out by focussing on the problem at hand and ask yourself, 'What are the *things* in this problem?'"

The second situation involves the type of application our software engineers usually work on. We are often called upon to create innovative solutions to information processing problems our clients present. This implies a high degree of conceptual design in our work. The conceptual aspect of object-oriented software engineering (and how it fits into an integrated SW methodology) is of great interest at PNL [Whiting 1990] and in the technical community overall.

Participants

The participants represented interests focussing on several different aspects of the software engineering lifecycle and brought experiences in several different design methodologies and documentation schemes. They are:

Sam S. Adams, Knowledge Systems Corporation
Erik Altmann, Carnegie-Mellon University
Bruce Anderson, University of Essex
Kent Beck, MasPar Computer Corporation
Hassan Gomaa, George Mason University
Sanjiv Gossain, Nokia Telecommunications
Richard Helm, IBM T.J. Watson Research Center
Ian Holland, Northeastern University
Norm Kerth, Elite Systems
Stevan Mrdalj, Eastern Michigan University
Joachim Schaper, Digital Equipment GmbH
Edwin Seidewitz, Goddard Space Flight Center
Regan Wilkinson, Object International
Russel Winder, University College London

Agenda

The morning session consisting of five presentations, followed by question and answer sessions, set the tone for the workshop. The presenters and their position paper titles were:

Kent Beck, Finding Objects with CRC Cards
Sanjiv Gossain and Bruce Anderson, Approaches in Object Determination
Hassan Gomaa, Criteria for Structuring a System into Objects
Richard Helm and Ian M. Holland, Language Driven Refinement and Abstraction of Objects
Sam S. Adams, Object Discovery Process

The complete set of position papers and related information may be obtained from the workshop organizers. Following the presentations, three topic areas were identified for further discussion during lunch and early afternoon, and results of the afternoon working groups were presented following the discussions.

Working Sessions

The group very quickly arrived at the three discussion areas for the afternoon. These issues represented recurring themes or questions of interest raised throughout the morning. The areas identified were:

Social Aspects - How can we describe the human interaction aspects of Finding the Object?

Strategies - How do you answer the question, "How do I go about finding the object?"

Roles/Kinds - What part do object "roles" or "kinds" play in finding the object?

Explanations, motivations, and focus points are expanded upon in the following sections.

Results

Group 1 - Socialization (Sam S. Adams)

During the morning sessions, it was noted that several of the "finding the object" procedures depended on the successful human interactions. For example, in their discussions of CRC (class-responsibility-collaboration) card use, both Kent Beck and Sam Adams noted that the vitality of the process was largely drawn from getting the people personally involved in the process. Having someone adopt the identity of an object and role play was very effective in exploring that object's behavior. Also, working out misunderstandings and role confusion created an atmosphere where the entire groups' understanding of the problem domain was raised. The Social Aspects group was formed to define and understand the social parameters that affect object-oriented analysis, object-oriented design, and "finding the object" in the context of the object-oriented software lifecycle. The three areas the group concentrated on were "people parameters", the infrastructure, and the products, all with respect to the social element. The caveat was raised that some issues would be best handled by those who have studied human interaction processes--it was acknowledged that we were not specialists in those areas however it would be beneficial to cautiously proceed.

"People Parameters"

Initial questions: What is so different about the object-oriented software development process? How does it affect the life-cycle? What social implications, therefore, come into play? The group noted that, initially, a domain focus is implied. To obtain a domain focus, you can either ignore the people involved (not a good idea) or exploit[2] the people involved (domain experts, users, analysts/technologists, facilitators, skill specialists) to achieve understanding in order to build better systems for users. The converse of the statement is also true. To build better systems, understanding must be gained, so people can be exploited, which implies more of a domain focus!

Other important people parameters generated by the group include:

- Types of people
- Number of people
- Problem complexity
- Flux in group make-up
- Need for documentation and communication
- Recording/expressing group processes
- Involvement of facilitators
- Process environment -- logistics, room makeup, etc.
- Problem immersion
- Role playing -- anthropomorphism and scenario exploration
- Process activities -- wandering, shuffling

The importance of the changing nature of the group (the "flux") was raised. Different specializations are required at different points in the life cycle. Problem domain experts are required at certain points; solution domain experts (including implementation specialists) are required at certain points; facilitators, managers, and so forth; are all vital parts of the process.

Infrastructure

The focus for the infrastructure was on the process, tools, and, the development environment.

Processes used in object oriented development efforts seem to naturally lend themselves to enhanced interpersonal interaction. This can be very clearly seen in the CRC card approach ([Beck and Cunningham 1989], and as enhanced by various groups). The approach encourages prototyping. Role

[2] "Exploit" being used in a positive sense here.

playing is vital to the process. Groups can be split and merged and results compared to gain higher insight to problem domains. The "Field Trip" to the problem domain is a commonly used term to describe the process of exploring and learning about the problem domain as a group.

Tools are required to record, express and facilitate. The process must be captured, yet the tools should not dictate the process. Group memory and the learning process should also be captured. Decisions should be recorded as well as the rationale behind them. This information should be continually recorded, maintained and made accessible throughout the entire life cycle.

The environment provided to the object-oriented software engineering team must support the process in several dimensions. Not only should the trappings of a process be provided (e.g., appropriate room space, tables, CRC cards, markers, and so forth), facilitators should be present, automated support tools should be available, an appropriate development environment should be available, and so on.

Products
Object-oriented development is a process that is evolutionary with the goal of product delivery. However, intermediate products, those generated by one phase to be used by the next phase (or by an iteration over the same phase) must be increasingly people-oriented. Object oriented specifications, for example, must be organized to accommodate change, and there is now a higher emphasis on reuse. Libraries and frameworks are including the results of implementation, plus analysis, plus design. There is an enhanced focus on using previous results here as well.

Products are generated via the process with appropriate tool use. The following shows the relationship between tools and products:

Tools	Products
Express--->	Results
Record---->	Rationale/History/Audit Trail
Facilitate->	Group Interaction/Team Building

The benefits of optimizing the social interaction parameters seem to be applicable over the entire SE life cycle. Products must be able to be communicated and understood over time.

Conclusions
Object-oriented processes change the focus of software engineering activity. With approaches focused on modelling the problem domain, the development team is led to consider issues of concern to a wider range of people than in traditional approaches. Increased involvement by different parties in the development cycle leads to social interactions that should be exploited. Object orientation is the unifying concept that can be applied throughout the development cycle. Object orientation leads to social interaction, and social interaction leads to object orientation.

Group 2 - Strategies (Stevan Mrdalj)
The goals of this working group were to discuss and compare opposing strategies for "finding the object", i.e., for developing an object-oriented model for an application. The two main points of discussion revolved around:

1. Prescriptive versus Exploratory methods, and
2. Top down versus Bottom up system decomposition approach.

The group started the discussion about prescriptive versus exploratory methods with the following definitions of the terms used.

The term "methods," above, refers loosely to any mechanism (technique) that assists in discovering (finding) the objects in the systems.

The term "prescriptive" implies detailed guidelines for system modeling, given in terms of tasks, checklists, and deliverable. It also implies a restricted set of options available to the developer at each stage. Prescriptive methods allow easier tracking and control of the development process, and benefit development by promoting completeness and correctness.

The term "exploratory" implies an open, less formal approach. An exploratory method guides development with a minimum number of constraints with some high-level guidance obtained from the emphasis on "look and feel" and observation. Also, the output at any stage of exploration is a refinement of or addition to the output of some previous stage. Exploratory methods do not facilitate the control of the development process. However, they do support the process of coming to understand the domain and the requirements of the system.

The following are examples of the methods presented at the workshop with the various levels of prescriptiveness:

1. Gossain and Anderson's approach uses five categories of objects (classes) and a domain analysis process which consists of six well defined steps. This method is outlined in their position paper.

2. Helm and Holland's Demeter/Contracts method guides modeling with seven pairs of questions any of which can be asked at each stage of the exploration. The answers to the questions are phrased in terms of specific constructs of the language and result in a new abstraction or refinement of the existing model. This method is outlined in the position paper and represented in the membership of the working group with Richard Helm and Ian Holland.

3. Class/Responsibility/Collaborator (CRC) cards method, which has only a simple, flexible, and expressive specification machinery (vehicle) associated with it. But it provides very little guidance for modeling. It was represented in the working group by Kent Beck.

In the first example, each stage in the process dictates particular activities, and what the following stage is to be. The second example represents a less formal method in which questions can be variably intermixed and can occur in cycles. The last example is the most exploratory method in which the exploration is driven by test cases and execution scenarios.

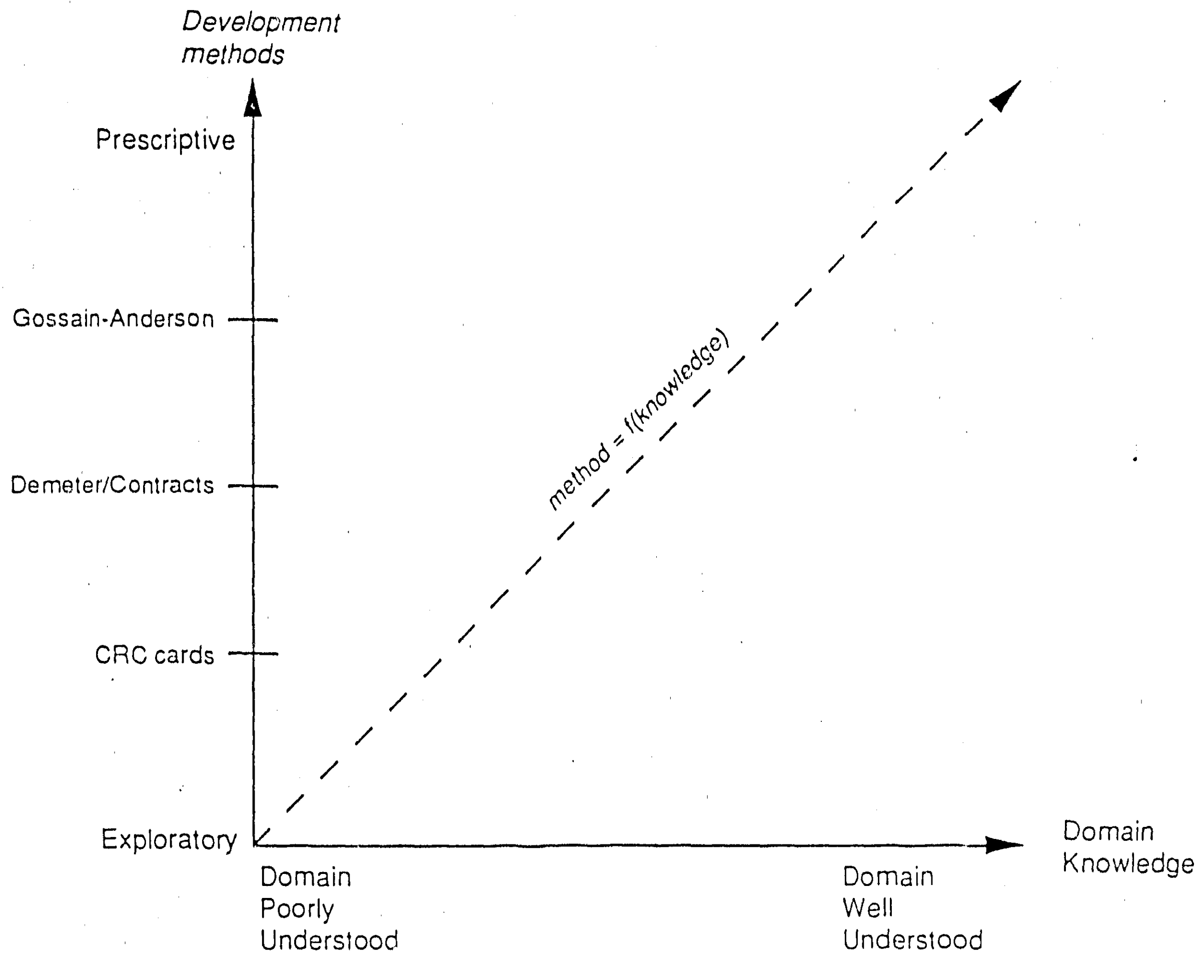
Rather than decide on "the best" way to discover the objects, the group attempted to relate the suitability of the alternatives to the level of knowledge held about the domain and the application. (The level of domain knowledge was also taken to increase as a function of the experience of the development teams, assuming transfer of generalized knowledge acquired in other domains.) The graph proposed by Erik Altmann, shown in figure 1, represents the consensus reached by the group. The x-axis represents the amount of knowledge held about the domain, and the y-axis ranks development methods by prescriptiveness. The dashed diagonal line is a guideline that relates prescriptiveness of methods to the amount of knowledge held about the domain.

The main points interpreted in the figure are as follows:

1. If the domain is well-known then previous experience will likely provide much of the structure of the domain model and the requirements of that model. In this context, a more prescriptive approach would be suitable, as it would lead "finding the object" along a well defined path and prevent omissions and unnecessary reinvention.

2. If the domain or the application is unfamiliar, then an exploratory approach is appropriate. With exploration the domain knowledge required to build a sound model can be discovered and refined.

3. The guideline suggested in the graph can be applied recursively to all aspects of the model and at all



Development methods versus Domain knowledge stages of the development.

The group also reached a consensus on the question of Top Down versus Bottom Up approaches to object oriented system decomposition. Kent Beck suggested that neither of these really applies, rather that modeling progresses from the "Known to the Unknown". Whether the "Known" portion of the model is some concrete object or a high level abstraction depends on the experience and bias of the developer. Each stage, after either abstraction from the concrete or refinement of an abstraction, yields some new known information. Stevan Mrdalj demonstrated that both Demeter/Contracts and CRC can be used in Top Down as well as in Bottom Up fashion and that the side from which system decomposition should begin depends on the level of domain knowledge.

Group 3 - Roles/Kinds (Norman Kerth)

During the workshop, it was noted that four of the participants had identified the casting or assignment of object roles as a distinctive element in the process of "finding the object" (FTO). It was also observed that object-oriented software engineering (OOSE) book authors (e.g., Booch, Coad/Yourdon) have also identified role casting as part of their FTO processes. It was further mentioned that there appeared to be similarities and overlaps between the identified roles. "Roles/Kinds" was chosen as the third area to explore by an afternoon working group session.

Purpose

The "Roles/Kinds" group's primary objective was to compare and contrast the three sets of roles identified in the participants' position papers/presentations. Other objectives were to review the evolution of roles from the participants' experiences in developing their FTO approaches, to characterize roles, and to identify the *Why*, *When*, and *How* of roles ("*What*" being part of the primary objective). More specifically,

Why - Why use roles? What support do they provide to OOSE?

When - When should you introduce roles to the FTO (or OOSE) process? Can we provide any heuristics for this?

How - How do you assign roles? How should roles be used?

Definition

Working definitions are the most desireable asset a working group can have, but also the most difficult to reach a consensus upon. Object-oriented jargon provided us the worst stumbling block in analyzing "role." However, there did seem to be an intuitive agreement among the working group members as to what extent we understood role (i.e., we had a good feel for what we were talking about), what clearly was not a role, and where the fuzzy areas of our understanding lay.

$$\text{Role} = \text{Kind} \neq \text{Class} \neq \text{Type}$$

Roles are groupings to assist in the conceptual processes in OOSE. Roles may change depending on several different influences, e.g., the stage of OOSE you are in, the application type you are attempting to construct, the view you are considering, and so forth. An object's role assignment may, therefore, also change as these factors change.

Background

Norm Kerth opened the discussion with a brief sketch of how roles evolved in his approach to FTO. His method involves a combination of Information Modelling [ala Shlaer, Mellor] as applied to understanding the knowledge within the user's environment, plus Event Partitioning [DeMarco, McMenamin] for understanding the transition of data throughout the system, plus his own 3D-HIP process to model the human interface to be developed. He found that understanding the differences between the kinds of objects in the system to be built was crucial to the overall understanding of the system to be built. Identifying these kinds led to his creation of the roles as outlined in his position paper.

His experiences prompted some discussion of role evolution from the other participants . The following (paraphrased) summaries were elicited from the participants in response to: "How did roles become part of your approach to FTO?" Kerth - "I needed to capture objects from several requirements documents from several different views--roles helped to organize this..." Gossain - "We needed to look at roles to assist in getting through the design; particularly in the transition from problem domain to solution domain..." Gomaa - "Roles assist in the simulation of the real world during analysis -- also for the move toward implementation..."

What Roles Have We Identified?
The following role sets were examined by the working group. Explanations of each role can be found in the particular author's position paper.

Gomaa		Gossain/Anderson	Kerth
Problem Domain		Domain	Foundation
	External Interface	Factor	Migration/Knowledge
	Data Abstraction	Application	System Domain
	Entity Modelling	Bridging	Application Domain
	Control	Basic	
	Algorithm		
	User Role		
Solution Domain			
	External Interface		
	Data Abstraction		
	Entity Modelling		
	Control		
	Algorithm		
	User Role		

Attempts at a written consensus of the roles was not as fruitful as our attempt to graphically portray the overlaps (due both to time constraints and the jargon problem). Figure 2[illegible] illustratesthe graphical consensus reached by the group comparing roles. The considerable degree of commonality discovered became known as the "Kerth Surprise Factor" and was shared by the group as a whole.

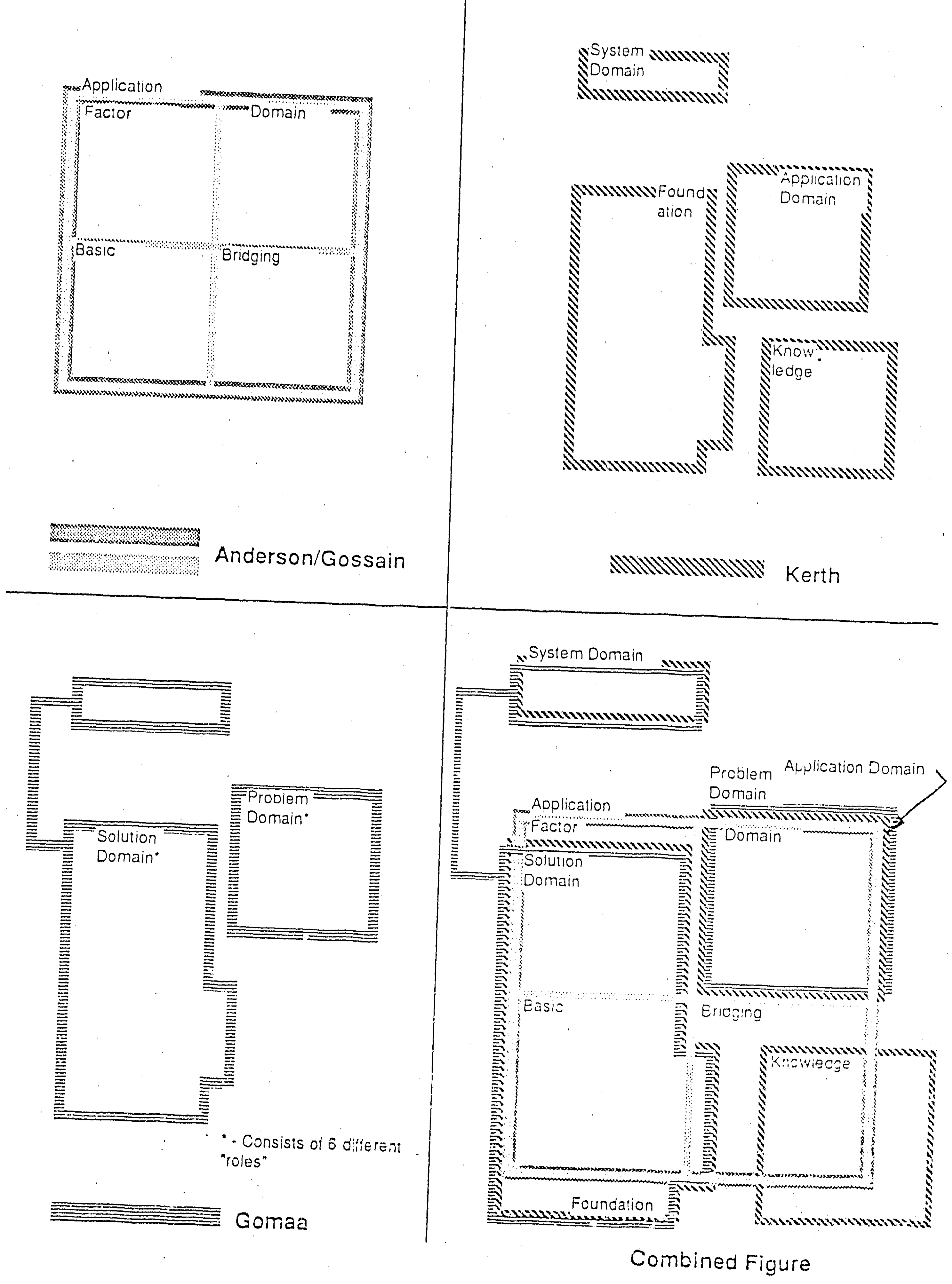
Why Roles?
Russel Winder succinctly stated that "roles provide compartments for thinking." These compartments may be orthogonal -- functionality may be a division, as may be view. But each may be required to enhance understanding of the problem domain.

When Should Roles be Introduced?
A heuristic agreed upon by the group was as follows:

Roles may or may not be introduced to an OOSE effort depending upon the size of the problem, the size of the team handling the problem, and "comfort level" of the individuals working the problem. Small efforts probably won't need roles. In larger efforts, the application size may require the introduction of roles to assist in the overall organization of the objects found. A large OOSE team may require the introduction of roles to assist in discussing and controlling the object set. Roles may also help individuals to think about the problem and solution.

How are Roles Assigned?
This was not analyzed in depth, however, the question was asked, "When a role set has been decided

Figure 2 - Consensus on Roles/Kinds



upon, can you find objects to fulfill particular roles?" The consensus seemed to be positive, and that you can iterate through a role set guided by your understanding of the problem.

Relationship of Roles to Other Workshop Working Groups

The use of roles is a consideration when developing any overall strategy to finding objects. The "When" heuristic is applicable here. However, the discussion of role did not come out in the Strategy group's (Group 2) discussions (which was not unexpected as it was beyond the scope of their topic set). The idea of role development with respect to group size and socialization aspects of an OOSE team in developing understanding of a problem was touched upon by the Socialization group (Group 1), and fits into their overall framework of object use in the software engineering process.

Issues for Further Study

The group agreed that it is not the appropriate time to attempt to create a standardized set of roles; the concept should be further explored overall before a first try. Further analysis and comparisons of other role sets that have been described in the literature may be very worthwhile. There was also a suggestion in the group that different role sets may be common to different application type (e.g., real-time oriented vs. data-base oriented vs. modelling oriented, and so forth), but this was not explored in any detail.

References

Beck, K. and W. Cunningham. 1989. "A Laboratory for Teaching Object-Oriented Thinking." *OOPSLA '89 Conference Proceedings*. ACM Press, New York.

Booch, G. 1991. *Object Oriented Design with Applications*. Benjamin/Cummings Publishing Company, Menlo Park, California.

Coad, P. and E. Yourdon. 1990. *Object Oriented Analysis* Yourdon Press, Englewood Cliffs, New Jersey.

Handford, M. 1987. *Where's Waldo?*. Little, Brown, and Co., Boston, MA.

"Proceedings of the Workshop on Finding the Object, OOPSLA/ECOOP," October 1990, Ottawa Canada. (Available from the workshop organizers)

Shlaer, S. and Mellor, S. 1988. *Object-Oriented Systems Analysis*. Yourdon Press, Englewood Cliffs, New Jersey.

Wirfs-Brock, R., B. Wilkerson, and L. Wiener. 1990. *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, New Jersey.

Whiting, M. 1990. "Conceptual Object-Oriented Design" *Pacific Northwest Software Quality Conference 1990 Proceedings*. October 29-31, 1990. Portland, Oregon.

Acknowledgements

This work was supported by the U.S. Department of Energy under contract DE-AC06-76RLO 1830.