Whiteboards, Flip Charts, and JAD Workshops

Donald G. Firesmith

Donald G. Firesmith provides consulting and training in object technology. He developed and maintains ADM, a fourth-generation O-O development method. He is the author of OBJECT-ORIENTED REQUIREMENTS ANALYSIS AND LOGICAL DESIGN: A SOFTWARE ENGINEERING APPROACH and OBJECT-ORIENTED DEVELOPMENT METHODS, STANDARDS, AND PROCEDURES. He can be reached at Advanced Software Technology Specialists, 2910 Webster Street, Fort Wayne, IN 46807, 219.745.7928, 73664.3515@compuserve.com.

MOST OBJECT-ORIENTED modeling is currently done by *individual* analysts, designers, or programmers using informal sketches on sheets of paper, flip charts, or whiteboards. If they are lucky, developers may also be able to create and maintain their diagrams with an upperCASE tool that provides editors for the relevant object-oriented diagrams (e.g., semantic nets, modified entity-relationship diagrams, inheritance diagrams, interaction or collaboration diagrams, state transition diagrams).

Interaction with other developers is often limited to occasional meetings and peer reviews. When multiple developers perform analysis and design as a group, they may also (or instead) use Class-Responsibility-Collaboration (CRC) cards to capture information about individual classes.[1] In either case, the analysis and design are too often done by software engineers with little domain expertise and limited access to domain experts or the users of the system.

Most of the time, the requirements, design constraints, quality goals, and customer desires that developers need to properly model the application come via written requirements from customers who often do not know what they or the users really want until after the first version of the application is fielded.

This article recommends using whiteboards and flip charts to capture the initial object-oriented models during joint application development (JAD) workshops involving domain experts and users as well as analysts and designers. Although JAD workshops are reasonably common in the information engineering (IE) community, they are still only seldom used for object-oriented requirements elicitation, requirements analysis, and logical design. JAD workshops are now even more useful because of the directness and completeness of object modeling and the iterative and incremental nature of object-oriented development cycles. The approach recommended by this article has been proven to be very successful for the rapid development of quality object-oriented models.

O-O JAD REQUIREMENTS ELICITATION AND MODELING WORKSHOPS

Most object-oriented development methods do not adequately address requirements elicitation. Instead, they assume the prior existence of well-documented formal customer requirements to be analyzed by developers during object-oriented requirements analysis. Such customer requirements, if they exist, are almost always textual, vague, incomplete, inconsistent, out-of-date, and ambiguous. Unfortunately, any requirements models provided by the customer are often still in the form of functionally decomposed data-flow diagrams or traditional entity relationship diagrams that ignore inheritance and the encapsulation of data and operations. Even if they exist, such obsolete models are not optimal inputs for analysis and design, especially if the object paradigm has been chosen.

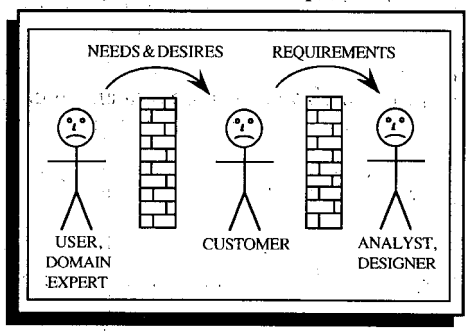

Figure 1: Traditional separation of concerns

The current document-driven process often separates requirements generation, specification, and analysis into disjoint user, customer, and developer activities. This is not only ineffective, it also promotes "waterfall" development, which is inconsistent with the incremental and iterative nature of the object-oriented development paradigm and development cycles. As illustrated in Figure 1, the resulting communication of requirements is inefficient and may easily become garbled between the domain experts, customers, and analysts/designers when they belong to separate organizations* and all information must be formally "passed over the wall" to the next group.

Incremental and iterative development is preferable, because many customers do not accurately know what they want until presented with functioning prototypes to elucidate, verify, and validate their desires, needs,

* When software is being developed for an external organization (e.g., the government), the customer is often an expert in contractual issues rather than in the application domain.

and requirements. In turn, this is one of the main reasons why "maintenance" typically requires 70%–90% of the total effort expended on many applications. An object is far more than merely an encapsulation of data (attributes) and functions (operations) on that data. It is a *model* of an application thing. To properly model reality, an object often must also encapsulate exceptions, rules (e.g., invariants), and component objects. It is impossible to develop good models of things without the necessary domain expertise of the thing being modeled. It is critical to have domain experts and users involved in the modeling process if good models are to be produced within a reasonable time frame. Otherwise, software engineers must all too often guess at the users' intent from written requirements in a domain they do not truly understand. Whereas the domain experts bring needed domain knowledge, analysts and designers can use object technology to recommend ways to perform business reengineering and ensure that inefficient manual processes are not merely automated as is.

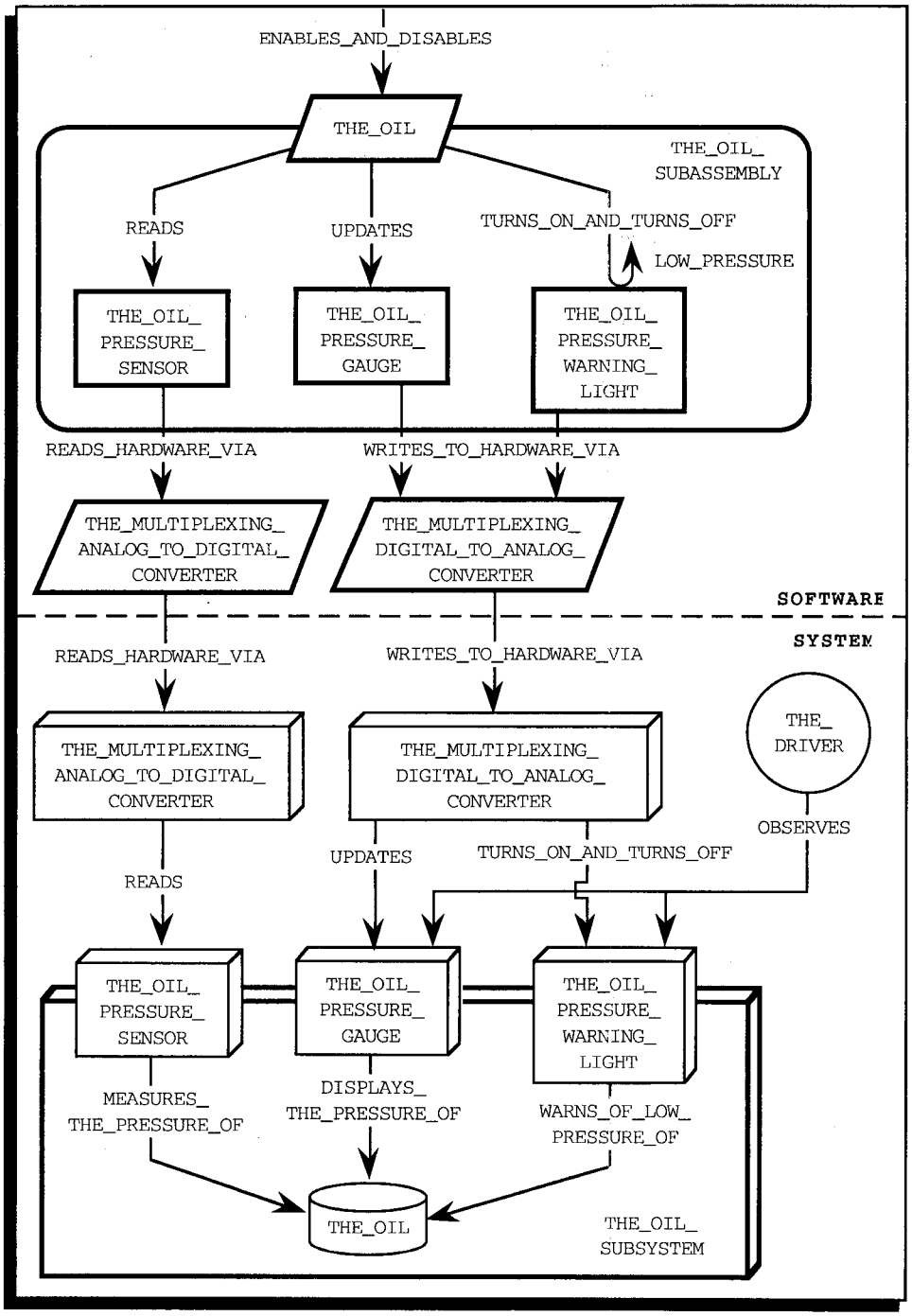

Figure 2: General Semantic Net (GSN) of THE_OIL_SUBASSEMBLY.

Much of the high-level modeling should therefore be done during JAD workshops in which JAD team members have adequate expertise in object-oriented modeling and the application domain. Often, they require systems and hardware expertise as well as software expertise. Thus, JAD team members should contain domain experts and users in addition to analysts and designers.

JAD teams should be neither too small nor too large. Experience teaches us that an optimum size should be between two and five total members. On the one hand, if the team is too small, it is difficult to ensure adequate expertise. Quality will suffer if the JAD team does not contain at least two members to ensure adequate peer-level review and iteration. On the other hand, having too many members produces design by committee and lowers productivity.

JAD workshops provide many advantages over using teams consisting only of developers. In addition to overcoming the previously mentioned problems, domain experts can get immediate feedback and answers to their questions. By reformulating domain expertise in the form of object-oriented models, domain experts develop a deeper understanding of their domain and have a greater confidence that the developers understand their domain and needs. Object-oriented analysts and designers also get ongoing "reality checks" of the models, which also increases their confidence. Different team members bring different perspectives on the problem, increasing the rate of iteration and making peer reviews more productive. The formal dog-and-pony-show reviews become more efficient and contain fewer surprises. By also including rapid prototyping with a language such as Smalltalk, domain experts can also obtain feedback from executable models in near real time.

Object technology makes JAD workshops more practical. Object-oriented modeling techniques are more understandable to the domain expert and systems engineer because objects better model understandable parts of the application domain. For example, good object-oriented diagrams are more readable than data-flow diagrams. Hardware engineers often prefer semantic nets or object-oriented entity relationship diagrams because of their similarity to traditional hardware schematics. Figure 2 is an ADM General Se-

mantic Net (GSN) from an automotive dashboard application,[2] one of several types of object-oriented diagrams that can be developed. The GSN shows four software objects, their corresponding hardware devices as indirect terminators, and the direct and indirect links between them.*

Once it is accepted that most of the high-level modeling will occur during JAD workshops, the next issue is how the modeling should be done to promote iteration and communication among team members who are not experts in object-oriented modeling.

PROBLEMS WITH CRC CARDS AND UPPERCASE TOOLS

No object is an island. Classes must be analyzed and designed in terms of the services they provide to their clients and the services they require of their servers.** Scenarios (e.g., use cases), idioms, mechanisms, and other patterns involve multiple collaborating objects and classes. Even an individual class consists of numerous interacting features, the two most important of which being its attributes and operations. Thus, during analysis and design, the relationships between things are often as important as the structure of the things themselves.

Because of this, almost all object-oriented diagrams consist of nodes (e.g., classes, states) connected by relationship arcs (e.g., links, messages, transitions). It is not enough to merely list the collaborators, as on CRC cards. A graphical structure consisting of nodes and arcs should be depicted graphically. A single object-oriented picture is literally worth a dozen CRC cards and pages and pages of code when it comes to human understandability,

A single object-oriented picture is literally worth a dozen CRC cards...

especially during JAD workshops when efficient communication between developers and domain experts is critical.

Thus, most object-oriented analysis and design methods include models that are largely graphical. These models (See Fig. 2) use different diagrams to capture different views (e.g., logical vs. physical or static vs. dynamic) of a system or its software. The scope of these diagrams also varies from the entire application (e.g., context diagrams) to an individual class or object (e.g., state transition diagrams, whitebox interaction diagrams) with the majority of diagrams documenting a small, logically-related collection* of collaborating objects, classes, and terminators.

UpperCASE tools exhibit several limitations as *initial* tools for object-oriented analysis and design. They may (or may not) support the specific diagrams of the project object-oriented development method. More important, they tend to inhibit initial requirements elicitation and interaction by primarily being tools for single developers rather than for groups. It is difficult to gather a group of domain experts and developers around a single monitor.

Even when the drawing is displayed by an overhead projector, spontaneity is lost when additions and changes to the drawing must be funneled through the single person at the keyboard. Developers tend to spend too much time and effort trying to make the drawing look good before it stabilizes rather than concentrating on the evolving content of the drawing. Those not doing the drawing do not feel as much a part of the development process as the person doing the drawing, who tends to view the drawing as his or her own rather than as the product of the group. Because one diagram often influences other diagrams in real time, it is important for several diagrams to be simultaneously visible to the JAD team. This is easy to achieve with multiple whiteboards but difficult when restricted to a monitor, even with multiple windows.

WHITEBOARDS

Most of these problems are solved by initially using whiteboards instead of CRC cards and upperCASE tools. Whiteboards are very cheap and easy for the entire JAD team to use, even for domain experts new to object technology. Whiteboards promote JAD team interaction and collaboration because all of the members of the JAD team can see and use the whiteboard simultaneously. Whiteboards promote iteration because little effort is invested in the diagrams. Team members feel free to jump up and make changes to the evolving diagram. Everyone feels themselves to be an active and valuable member of the team because they all take part in the development and modification of the models. By actively developing and iterating the diagrams, the domain experts, users, and customers understand and buy into the resulting models.

Large whiteboards are preferable to flipcharts and chalkboards for drawing object-oriented diagrams. Flipcharts limit iteration and are too small for typical diagrams, which may easily contain seven to fifteen nodes.* This is especially true if the labels of the nodes and arcs are to be easily read by JAD team members sitting across the room. Chalkboards are very similar to whiteboards in many ways but do not allow the use of node cards (see below), because chalk dust prevents the taping of the cards to the chalkboard's surface.

During JAD workshops, team members incrementally develop the diagrams, one node or arc at a time. Because modeling is an exploratory process, each diagram will usually evolve through two or three iterations over several hours before stabilizing. Nodes must often be moved around the diagram as initial relationships are replaced by new ones or the diagram is rearranged to avoid clutter and the crossing of lines.

* The *software* subassembly at the top of the GSN contains one visible concurrent object, the oil, and three hidden sequential objects connected by three links, one of which is conditional. These three hidden software objects are indirectly linked to the two hardware converters, which in turn are linked to the visible hardware sensor, gauge, and light objects in the oil subsystem, which also encapsulates the physical motor oil object. One of the important reasons to distinguish between software and hardware objects on the diagrams is that the direction of the links among (and control flow between) the software objects is often the opposite of the direction of the links among the hardware objects.

**This is perhaps an oversimplification, because messages may actually be used for four distinct purposes: to request a service, to supply data, to provide notification of an event, or to synchronize the threads of control of two concurrent objects.

* a.k.a. assembly, class categories, cluster, kit, subject, or subsystem.

* The number of nodes may be even larger during the initial brainstorming or later when design-level objects may be added.

I recommend *node cards* as a labor-saving device used to eliminate the need to constantly redraw and relabel nodes when they are moved around the diagram. Node cards are merely pieces of paper that have been preprinted with the icon* for a node used on an object-oriented diagram (e.g., a class or object). When a new node is needed, a JAD team member selects the appropriate card and uses a thick felt-tipped pen to label it in block letters that can be read from anywhere in the JAD workshop room. The card is then taped to the whiteboard and appropriate relationship arcs are drawn on the whiteboard, connecting it with other nodes.

I do *not* recommend listing responsibilities, attributes, or operations on the node cards. Initial modeling should be at a higher level of abstraction, a level that treats objects and classes as software blackboxes. Listing resources on node cards also clutters up the cards (and diagram), making the diagram difficult to understand and the cards difficult to read from across the room. It also makes it difficult to use a standard-sized card, because some nodes would require larger lists of features than others. Also, once one starts listing information such as encapsulated resources, when should one stop? Why not list exceptions, invariants, and parts? I have recently removed such information from most of the diagrams of my ADM method because such information is best captured using upperCASE tool dialog boxes. Instead, I recommend using flipcharts for informally capturing such information prior to its entry into a CASE tool repository.

Originally, I used large Post-it notes as node cards. Unfortunately, they had several drawbacks. One had to either draw the icon shape on each Post-it note which took time, or pay an inordinate amount to have the Post-it notes preprinted with the appropriate icon.* A more serious flaw was that Post-it notes are only sticky along a thin strip at the top. The next day or after returning from lunch, one often found part of one's design lying on the floor at the foot of the whiteboard. I briefly taped the Post-it notes to the board, but quickly realized that as engineering overkill. Currently, I use the drawing tool MacDraw to draw two large annotated icons on a standard sheet of paper. These I copy with a standard office copier before bisecting them with a paper cutter to produce pairs of new node cards.

In object-oriented JAD workshops, analysis and design should consume 50-60% of the effort

Node cards share many of the advantages of CRC cards. They are cheap and easy to use. Because little effort and time is invested in creating them, developers have no hesitation in replacing or modifying them, thus promoting iteration. Because they are physical, developers more easily think of them (and of the corresponding classes, objects, etc.) as things rather than functions. Unlike CRC cards, node cards are intended to be used on whiteboards to create object-oriented diagrams. They do not need to list collaborators, because relationships are drawn as arcs connecting the cards.

FLIPCHARTS

Flip charts should be used to initially capture analysis and design information that does not belong on the diagrams, prior to recording it via the project upperCASE tool. Flip charts can be used to list requirements, responsibilities, objects and classes, protocols of messages and exceptions, and resources such as attributes, operations, exceptions, invariants, and component objects.

RECOMMENDED JAD FACILITIES

When using object-oriented JAD workshops, analysis and design should consume 50%–60% of the effort of the project, and more than half of this effort should take place in specially designed conference rooms. Because many small (e.g., two to five member) analysis and design teams will be working in parallel, many *small* conference rooms will be required.* As illustrated in Figure 3, each room should contain a single table that is large enough to seat eight (i.e., the team members and any guests such as additional domain experts and representatives from other JAD teams).

Very *large* whiteboards should be mounted on three of the four walls. These whiteboards will be used to develop the graphical models, using the notation of the chosen object-oriented development method. Two flip charts will be used to list objects and classes, class

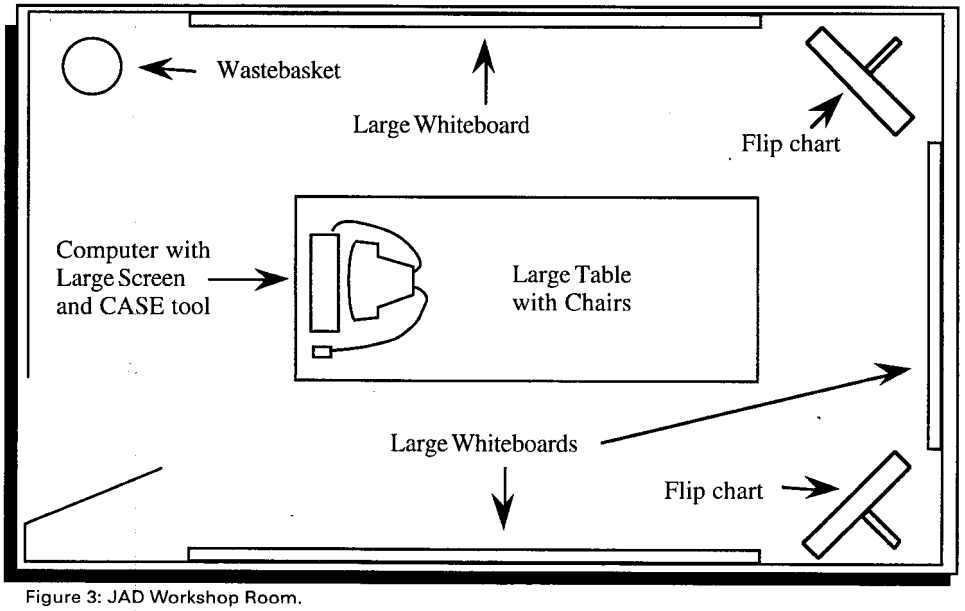

Figure 3: JAD Workshop Room.

* Often, the icon is also annotated with its meaning (e.g., class, object, state, operation) in small letters as a memory aid for team members who have not yet memorized all the icons.

* Many incremental, iterative object-oriented development cycles produce many small increments (e.g., less than the Miller limit of seven plus or minus two) classes. Often, more than one child increment can be spawned off of a single parent increment, producing multiple teams working in parallel.

Sometimes low-tech tools are the best, especially if used by the right people, in the right way, at the right time

resources, lists of things to do, and so forth. A personal computer or work station with a very large screen should be set on one end of the table so that its user can easily see the whiteboards and flip charts. The computer will be loaded with the upperCASE tool and used to capture the models generated by the team on the whiteboards and flip charts. A large wastebasket will be used to hold the results of iteration (e.g., the cards of classes that do not survive).

RECOMMENDED MODELING PROCESS

Object-oriented requirements such as elicitation, analysis, and design should be performed incrementally and iteratively using small increments (a.k.a. assemblies, class categories, clusters, kits, or subsystems). Systems analysis and design should typically be performed before software analysis and design, at least on a subsystem-by-subsystem basis. Once the context diagram(s) have been developed, initial increments (e.g., subsystems, subassemblies) should be identified.

On an increment-by-increment basis, the JAD team members iteratively develop the associated diagrams on the whiteboards.* For efficiency's sake, JAD team members may well prepare initial sketches of the diagrams prior to the workshops, but the main modeling should take place as a group.

Three whiteboards are recommended because it is usually useful to simultaneously have three diagrams on the boards: (1) the current diagram being worked on, (2) the previous diagram, which is still subject to significant iteration due to discoveries made while working on the current diagram, and (3) its previous diagram, which should by now be relatively stable and should be captured using the project upperCASE tool.

Capturing stable diagrams with the CASE tool is a good job for junior members of the team in that it requires less expertise, costs less, and is a good way for junior members to develop experience in modeling.

For each diagram, the main abstractions (nodes) are identified and associated node cards are chosen, labeled, and taped to the whiteboard. The cards are then connected with relationship arcs.

My ADM method recommends placing clients above servers, with the directed relationship arcs drawn from the client to the server. New nodes and arcs are added as necessary, and the diagram is iterated. Usually two to three iterations over the course of two to three hours are required before the diagram is reasonably stable. The next diagram recommended by the development method is started, and the JAD team members often jump from one diagram to another as insights gained in one diagram result in changes to another.

Flip charts are used to capture initial lists of textual information that is best left off of the diagrams. Stable information is captured using the upperCASE tool. Information that does not survive the elicitation, analysis, and design process is consigned to the trash can.

CONCLUSION

This article has recommended using whiteboards and flip charts to capture the initial object-oriented models during JAD workshops involving domain experts and users as well as analysts and designers. This approach has proven to be a very efficient, user-friendly way of eliciting and analyzing object-oriented customer/user requirements. Sometimes, low-tech tools are the best, especially if used by the right people, in the right way, at the right time. ⊠

* These increments may be identified and modeled in many orders: top-down or outside-in by message passing, bottom-up by inheritance, by horizontal layers, or by vertical partitions.

References

1. Beck, K., and W. Cunningham. A laboratory for teaching object-oriented thinking, SIGPLAN NOTICES, 24(10), 1989.
2. Firesmith, D.G. OBJECT-ORIENTED REQUIREMENTS FOR ANALYSIS AND LOGICAL DESIGN: A SOFTWARE ENGINEERING APPROACH, John Wiley and Sons, New York, NY, 1993.