The OO Software Development Process
PANEL

Dennis de Champeaux, HP-Lab, moderator
Bob Balzer, Information Sciences Institute
Dave Bulman, Pragmatics
Kathleen Culver-Lozo, AT&T Bell Laboratories
Ivar Jacobson, Objectory
Stephen J. Mellor, Project Technology

Co-organizer: **Pankaj Garg, HP-Lab**

Main Topics

A (software) development method has at least the following components:
– A language core for representing systems and their components,
– An (informal) procedure for guiding the development,
– (Informal) criteria for measuring progress and a "decision procedure" for terminating the development process.

We see an abundance of notions and notations in the OO paradigm: multiple OO programming languages and multiple "formalisms" for OO analysis and design. Procedures for guiding the development process are sparse. The sequence in which notational apparatus is presented in the literature is often the default process. In Rumbaugh et al, we find (only) a check list of development activities.

The literature gives us some guidance with respect to major activities of the development process. The waterfall, spiral and fountain model all agree upon distinctions such as: requirements capture, analysis, design, implementation, testing, maintenance. They also agree in that they do not provide a finer granularity of the inside activities of analysis and design.

We postulate that this omission is a key inhibitor to a more widespread acceptance of the OO paradigm by *project managers*. They do not know how to *plan* a non-trivial OO software development project.

The purpose of this panel is to take the first steps towards developing such process models. More specifically the panelists have been asked to address the following questions:

- How does the Structured paradigm compare with the OO paradigm on the process dimension. Qualify, if necessary, the answer with respect to the analysis, design, implementation, maintenance phases. Is there a qualitative difference between the two? For instance, is there a difference with respect to divide and conquer tactics? Can you substantiate your claim with metrics?
- Can you envision a generic OOA/ OOD development process? If so what are the major

steps and deliverables of these steps. If not, why not?

- If a software development organization had to choose between either doing a paradigm shift to OO, or improving its rating on the SEI maturity scale what would you recommend? What are the relationships between the two? Qualify, if necessary, the answer with respect to an extant maturity level.
- Large development tasks require teamwork. Is there a qualitative difference between the Structured development process versus the OO development process regarding the complexity of teamwork, subtask assignment, subtask interference, subtask granularity, etc.?
- The transition from a Structured development shop to an OO development shop doesn't happen automatically. Are there "invariants" in this transition? Can you give recommendations? For example, who will have to be trained first: programmers? designers? analysts? managers?
- What aspects of the software process are actually independent of the OO vs. Structured paradigm? I.e., what is the core of a software process which does not depend on whether you are using OO, Structured, or whatever.
- While providing support for process automation in a Software Engineering Environment, can the environment support both the OO and the Structured paradigm at the same time?

Position Statement of Bob Balzer

Bob Balzer was unable to produce a position statement.

Position Statement of Dave Bulman

A well-developed OO process of software development improves on a "Structured" process in each development phase. It is hard to decide which phase will be most affected, because the improvement will be so broad. Maintenance is greatly improved by the encapsulation of objects and the likelihood of reuse. If a good OO Analysis is done, both the design and the implementation is simpler due to more complete separation of concerns resulting from partitioning into more independent modules. Better partitioning will also provide a greater opportunity for groups of classes to be developed in parallel. In addition, implementation will be able to take advantage of much more reuse – both via inheritance and due to the fact that the objects themselves will be more reusable.

If a choice must be made between shifting to OO and improving the SEI rating, the shift to OO should be done first. As the organization moves up the maturity scale, it will be with the better method. If an organization chose to first move to a higher level on the SEI scale, and then change to OO, this would cause a temporary drop back down the scale, because different parts of the organization making the shift at different rates will result in inconsistency.

For problems so large that a large staff is needed, OO leads on each facet of the required teamwork. Once the initial (even tentative) definition of classes is started, teams can start development on separate groups of classes, with less ripple effect between classes when changes are needed. The granularity of development tasks has at least two opportunities in the case of OO. Each class has a natural strong fire wall between it and most other classes. In the rare case of a very complex class, some of the class operations will be sufficiently modular and sufficiently large to allow assignment as separate subtasks.

The right way to make the transition to an OO development shop is the same as for any transition and is obvious; unfortunately, this obvious truth has only very rarely been honored during any transition. The order of training should be managers, analysts, designers, and then programmers. The probability that many organizations will follow this path is somewhere between zero and nothing.

Managers are always too "busy." Analysts are always preparing the bid for the next huge contract. Programmers are always putting out fires caused partly by lack of training. Who knows if there are any Designers? Any new method should be used on a non-trivial project, whose late delivery will not be fatal to the organization. Most of this has been preached for many years, but apparently to a vacuum.

It is neither necessary, nor desirable, to change every aspect of the software process when making the change to OO development. The need to understand the users of the software, and communicate with them in a way they can understand does not change. This is the core of every development process, and must dominate the devising of new development methods. It simply does not matter how well you solve a problem, if it is not the one your customer wanted solved.

The development environment in any large organization *must* provide support for both an OO development process and the previously used process. No organization can commit suicide for the sake of a new "truth."

Position Statement of Kathleen Culver-Lozo

Theme: Defining an Object-Oriented Software Development Process

Before focusing on the object-oriented software development process, we must first decide how we want to represent a software development process. In many organizations, the de facto representation is textual descriptions. These descriptions are found in papers, text books, memoranda, and training materials. Other organizations rely on "gurus" and consultants to store information about the process and to spread their knowledge to novices. Using these informal representations to express the software development process presents many problems. First, comparing processes is difficult. How does a process using Yourdon's modern structured analysis compare with a process using Shlaer-Mellor's object-oriented systems analysis? Informal representations do not suggest parameters along which the processes can be compared. Second, informal representations do not suggest where a process may be incomplete or inconsistent. One description may include information about the outputs for each step in the process and another description may not.

At AT&T Bell Laboratories work on representing the software development process has been underway for the past two years. A methodology and software tool for modeling the software development process has been created and used to represent processes in AT&T. Processes based on both structured and object-oriented methods have been created and are being used to develop software. The representation of these processes is composed of a set of 'views' which organize information about the process. The views express such information as tasks, task inputs, task outputs, task dependencies, tool support for tasks, training required to complete tasks, task durations, etc.

Given a representation for modeling the software process, the impact of object-oriented methods on the software development process can then be studied. For example, some projects at AT&T are using Shlaer-Mellor's OO systems analysis and other projects are using Yourdon's modern structured analysis. A fragment of two processes that have been defined and that are in use based upon these analysis approaches are summarized in Tables 1 and 2 using a very small subset of the views.

By representing the two processes in this way, even partially, several observations can be made. User interface prototypes are needed in both processes because they are a proven approach for designing successful user interfaces. However, to develop a prototype, the human factors expert uses an information model and an event list in one process and an information model and state models in another process. Both processes produce an output called an information model although different tasks produce the diagram in the different processes (i.e., prepare entity-relation diagram and prepare information model). (Note that a complete

Task Name	Inputs	Outputs
prepare event list	background information	event list
prepare entity-relation diagram	background information	information model
develop user interface prototype	information model,event list	UI prototype

Table 1: Process Fragment based on Yourdon's Modern Structured Analysis

Task Name	Inputs	Outputs
prepare information model	background information	information model
define lifecycle of objects	information model	state model
develop user interface prototype	information model, state models	UI prototype

Table 2: Process Fragment based on Shlaer-Mellor's OO Systems Analysis

representation of the processes would describe the content of the outputs allowing the reader to compare whether the different information models actually include the same information.)

This example shows that when studying and communicating an OO software development process, we are influenced by how we represent the software development process. The example also shows that by carefully choosing how we represent the process, we can improve both our description and our understanding of OO methods.

Position Statement of Ivar Jacobson

The waterfall model, the spiral model, the evolutionary model, etc. are all special cases of a generic model for software development - the process model.

The process model views the software development organisation as a system (an enterprise or a business) and describes this organisation in terms of communicating processes such as analysis, design, implementation and testing processes. Using the process model a development project is a thread, a use case, through these processes and can be described using the waterfall notation; another thread through these processes is a "spiral" consisting of prototypes, incremental development steps, product releases, etc.; error handling is yet another very special thread through some, not necessarily all processes.

My position can briefly be summarized by:

1. A generic object-oriented development process can be a reality. This process is designed as a set of communicating sub-processes (objects) using an OO approach. The sub-processes can be specialized to meet different needs: business needs, implementation needs (e.g. programming language, computer architecture), organisation needs (e.g. geographical distribution), etc. They can also be designed to offer different threads for prototyping, incremental design, etc.

2. The generic process supports not only development projects (as the waterfall model, etc. does) but also supports the whole life-cycle of the developed software; it is product-oriented. Product-orientation means responsibilities are allocated to objects. Due to the object encapsulation property, clear responsibilities are achieved.

3. Level 5 of the SEI-scale means that a systematic process with self improvement is installed

in the organization being measured. To offer this ability the generic process has a special sub-process for process development. The process development sub-process measures the different activities in the other sub-processes and can change these sub-processes as better tools becomes available or as new implementation techniques evolve.

4. Each sub-process can be "formally" described using e.g. state transition graphs. A transition is invoked by a stimuli from another process and consists of a sequence of activities (some in parallel) including invokations of other processes. Each activity can be described in terms of input data, micro-design steps, criteria for "good" design, checklist and output data.

5. The process structure is important to the process developers (the methodologists) but must be made transparent to the people performing/ executing the processes, preferable using supporting tools. These people are e.g. product managers, project managers, analysts, designers, testers, customers which need different handbooks, user guides, manuals, etc. in paper or integrated in the tools.

Position Statement of Sally Shlaer and Stephen J. Mellor

Over the last ten years, we have been developing and using an object-oriented software development process known as Recursive Design. The process proceeds as follows:

Partition into Domains The system to be built is first divided into distinct, independent subject matters, or "domains": an application domain (the subject matter of concern to the end user of the system), several more general domains (such as a user interface and a sensors-and-actuators domain), and an architectural domain. The domains are organized in client-server relationships, so that a domain acting as client can rely on a server domain to provide commonly needed mechanisms and services.

Analyze Application Domain We then analyze the application domain using Object-Oriented Analysis [1]. This produces a set of formal models: An Information Model that defines the conceptual entities (objects) of the domain and the relationships between the objects, State Models that prescribe the lifecycles (behavior) of each object and relationship, and Action Data Flow Diagrams that factor the processing required in the State Models.

Extract Requirements for Server Domains Once the analysis of a domain has been completed, we extract from it certain requirements that must be fulfilled by other domains. These requirements often take the form of assumptions: The application domain assumes that mechanisms will be provided for storing persistent data, for transport of events, and for communicating with an operator. The requirements are then assigned to the various server domains in the system.

Analyze other domains We then proceed to analyze each of the other domains separately. An order is imposed by the client-server relationships: One must complete the analysis of all clients in order to extract all the requirements that must be fulfilled by a particular server.

Architectural Domain The last domain to be treated is the architectural domain. This domain is required to provide rules and mechanisms for managing data and control for the system as a whole. The conceptual entities in this domain are defined to support the type of design desired: In a fast process control system, the conceptual entities are likely to include periodic tasks and a universally accessible run-time database, whereas, an object-oriented design will be based on object-oriented concepts and include entities such as container classes and classes to implement functions such as finite state machines and timers. The architecture will also state the policy for assigning instances and classes to tasks, and dictate the protocols that communicating tasks must adhere to.

The architectural domain is specified in OOA and in a design notation that supports the con-

ceptual entities of the system design. Hence, for object-oriented designs we use a notation supporting OOD constructs [1]. No matter what kind of design is desired, the architectural domain is specified entirely independently of the application and other client domains.

Build Mechanisms and Templates The architecture is then implemented in the form of traditional complete components (tasks, interface functions, etc.) and as templates: structures of code and data that need to be completed by adding elements from the client domains. In a strong architecture, the templates prescribe virtually the entire design of the system.

Populate Templates The replaceable segments in the templates are then populated with elements from the OOA models of the client domains. This step may be carried out by a combination of automation and orderly manual procedure.

Implications:

This approach reduces and controls iteration in analysis by confining it to a single domain at a time. Iteration in design is similarly controlled: Modifications to the design are made entirely in the architectural domain and propagated to the entire system through the templates.

The approach systematizes and supports reuse of entire domains. Because domains are kept completely separate from one another until the final construction steps, they can be transported intact to other systems. This applies particularly to the architectural domain: This domain – including the mechanisms and templates – is commonly reused for other systems that have basically the same loading and performance characteristics.

Because the process is based on transformation of analysis models, it is highly susceptible to automation. We have seen a number of "transformation engines" built from common text manipulation tools and pre-processors that feed a "make" utility. Some projects have pushed the automation process as far back as the CASE tools, extracting the OOA elements needed to populate the templates from the CASE database.

[1] Sally Shlaer and Stephen J. Mellor, Object-Lifecycles: Modeling the World in States, Prentice Hall, 1992.