# OO Methodologies: Process & Product Patterns

by Ellen Gottesdiener, President, EBG Consulting, Inc



*This article is published in the November, 1998 issue of Component Strategies, Vol. 1, No. 5.*

Now that distributed objects, Intranet application development, interactive development environments based on object-oriented models are all the rage, what similarities and differences can be found among current object-oriented (oo) methodologies? How has the Unified Modeling Language (UML) changed object-oriented methodology? How are methodologies adapting to object-orientation and components? This article answers these questions and discusses major themes of oo methodologies along with the factors that make them succeed.

## Methodology definitions and Object-oriented methodology elements

A 'methodology' is a systematic collection of techniques & guidelines for *how* to build, buy, maintain and/or enhance software products. A methodology provides a basis for communication, a toolkit of techniques and a basis for repeatable, reliable software engineering. The term 'method' refers to an approach to activities generally adhering to common principles. 'Methodologies' go beyond this approach to incorporate guidance for business analysis, project planning and management, project processes (e.g. estimation, metrics, risk management), quality assurance, testing, role and responsibilities, reuse and architectural design).

Methodologies partition work into organized phases and/or stages which further decompose into activities and tasks such as what do and the rationale for doing it, task inputs, task outputs (deliverables), predecessor activities, roles, techniques, metrics and tools. These sets of organized work patterns are known as templates, routes, work flows or process patterns. The work pattern is tailored to a specific project to become the basis for a work breakdown structure (WBS).

Methodology work patterns are characterized in several ways: the technology used (e.g. object-orientation, expert systems), by the objective of the project (e.g. install a package, refurbish an aged application) or by the software architecture being employed (e.g. distributed objects, Intranet). Examples of methodology work patterns include enhancement, package integration, object-orientation, knowledge-based, real-time, information engineering, and client/server.

A 'process model' is a manner of conducting work, which is oriented toward the delivery of the product. It is variously referred to as a 'process 'approach', or 'delivery framework'. Examples include: waterfall, staged delivery, incremental,

evolutionary delivery, and fountain. The waterfall process is a top-down approach in which a series of sequential activities are used to deliver the software in one major release at the end of the whole process. The waterfall process model does not consider the effect of iterative prototyping on the overall process, and was largely been abandoned when client/server architectures evolved (ref. my client/server article).

Object-oriented methodologies eschew the waterfall model. Instead, they prescribe *iteration*, *incremental delivery*, and *parallel* or *overlapping* phases. Generally, object-oriented methodologies contain similar stages or phases (see Figure 1). Iteration is accomplished via a spiral process model for each for lifecycle stage. The spiral process involves prototyping, risk reduction activities and successively deeper discovery of application requirements. This type of process is natural and standard in object-oriented methodologies. Object-oriented methodologies encourage carving out 'slices' of the whole software product and releasing them in succession. Slices may therefore be analyzed, designed and developed in parallel. Alternatively, lifecycle phases may be overlapped to speed delivery of phase artifacts and promote iteration (overlapping lifecycle).

These phases and process models emerged in mid 90's with client/server methodologies. Many object-oriented methodologies have their roots in these methodologies, since object-oriented applications use client/server and distributed computing styles. Object-oriented methods differ in subtle ways from client/server methodologies.

Common ideas between oo and client/server remain, including a high degree of end user involvement via JAD (Joint Application Design) – like workshops and prototyping, small cohesive teams, early testing and 'proof of concept', risk management, the need to partition, and doing usage analysis in order to make design tradeoffs during architectural design. Changes found in oo methodologies include the desire to exploit patterns and components, a shift from using OMT and Booch models toward UML, an emphasis on choosing a distributed object architecture, and in practice, a subtle shift from larger-scale CMM (Capability Maturity Model)-style process improvement to smaller scale team learning.

## Methodology Differences: Procurement, Scope, Principles and Packaging

Object-oriented methodologies may be acquired in different ways:

- commercially purchased (e.g. Platinum/LBMS, Rational, Client/Server Connection)
- purchased as part of consulting work (e.g. Ernst & Young, Anderson Consulting)
- provided as a byproduct of software development subcontracting (e.g. Geneer)

- developed/adapted from existing non-proprietary methodologies (e.g. OPEN, DSDM, Catalysis)

Object-oriented methodologies vary according to:

- breadth (scope of the lifecycle stages supported)
- how work products (deliverables) are specified (type of models, format of the deliverables, quantity of examples and templates provided)
- principles used in the methodology (e.g. union of logical and physical architecture, service-based, responsibility-driven, component-based, use case centric)
- depth of the techniques furnished (including examples and case studies)
- degree of integration with other tools such as process management, project management, testing, and modeling tools
- degree of emphasis on process management and improvement
- the format and delivery of the content (packaging)

Some object-oriented methodologies are delivered as software products that can generate a work breakdown structure (WBS) and have process management capabilities. These products are themselves "methodology engineering" tools. Others are provided as text user guides, hypertext or in books. Some include mini case studies, tips and examples, standards such as GUI and documents templates. Some offer a variety of WBS ‘templates’ or process frameworks. Some have tutorials built in.

Based on the representative methodologies reviewed for this article, oo methodologies are generally strong in the areas of requirements/analysis and design. However, the areas of technical design, architecture, and incorporating legacy applications could be improved. Most oo methodologies have business modeling added to the requirements process and now have techniques for user interface modeling, an improvement over client/server methodologies. Most have updated at least some of the deliverables to include UML-compliant diagrams. Little guidance has been provided for recognizing, identifying and cataloging patterns, although most have added facilities for reuse. None of the methodologies analyzed as exemplary methodologies for this article addressed organizing software and applications using intelligent agents.

## UML integration

The acceptance of the Unified Modeling Language (UML) as standard (by the OMG -Object management Group in late 1997) has changed only one aspect of the risky art and science of software development: notation. "Now, like other engineering fields with a notation that is widely taught and understood, we have standard notation with the UML, despite its’ complexity", says Tony Wasserman, Software Methods and Tools. Methodologist, consultant and educator James

Odell concurs. "85% of object-oriented methodology is the same. The modeling is different. We knew what to do with data, now we need to do this with process".

In the US and amongst a majority of software development organizations, the UML has become a de facto notation. The UML, however, is only a notation. Only recently has the "Rational Unified Process" (RUP) been announced for the end 1998 general availability as a superset to Rational's Objectory methodology. (Current Rational Objectory customers have access to the RUP beta). The beta version of RUP has facilities for business modeling, some user interface modeling, more project management support, some additions for components, and greater integration with Rational's suite of tools.

A rich, 'third-generation' non-proprietary object-oriented methodology is also available. OPEN (Object-oriented Process, Environment and Notation) actually contributed significantly towards a competing submission to the OMG for an object-oriented analysis and design modeling language. While the UML is a notation and a metamodel, OPEN contains both a full metamodel and notation (OML or the

OPEN Modeling Language). OPEN also goes beyond the UML by providing all the aspects of a full methodology, and can actually support the UML.

OPEN was developed beginning in the early 90s from a merger of numerous object-oriented methodologies and inclusion of ideas from over 30 methodologists and researchers. They form the OPEN Consortium.

Despite the wider acceptance of the UML notation, OPEN's modeling language, OML, continues to be enhanced and may be used as a superset to the UML for some features not currently provided in the UML (Reference #1). Further, OPEN has an abundant set of tasks and techniques to draw from in providing full methodological support. OPEN, however, has not yet been backed by the commercial forces that backed the UML. Thus OPEN has not yet gained much commercial attention. However, Brian Henderson-Sellers, one of OPEN's author's, believes "we are now entering into the realization by industry developers that they DO need a methodology and when they do, OPEN's mindshare is set to grow by leaps and bounds".

## Themes (patterns) for object-oriented methodologies

Patterns of similarity and difference emerge in analyzing a variety of commercial and non-commercial object-oriented methodologies. Similarities include an increased emphasis on requirements definition and traceability, the 'testing' work products beginning in the early stages (e.g. requirements), delivery in increments, the use of an iterative processes, parallel development of slices or increments, shorter delivery cycles, the use of use cases for functional

requirements, the identification of of team roles such as architect, tester (or QA analyst), and business analyst.

A simple, but sometimes confusing, distinction amongst oo methodologies are the terms used to refer to it's own elements. Deliverables or work products, which result from an activity, have different names. For example, 'work products' are used in CLIPP, while Rational Unified Process uses the term 'artifacts'.

Beyond those simple distinctions, some methodologies use more complex terms for deliverables and techniques that require team members to learn a new set of semantics. For example, James Martin + Co.'s *UML Client/server Object-Oriented Process* has an 'inheritable interface component' and 'patterns for data structure design' . Rational's beta version of Rational Unified Process (RUP) has activities such as 'identifying design mechanisms' and 'packaging boundary classes'. Other methodologies are more direct, like Geneer which uses simple terms like 'basic interactions between objects', 'system architecture' (subsystems, model, views, system utilities, IO), and 'proof of concept'. In addition, phase names also vary (see Sampler sidebar), although for the most part, the models produced in those phases have common names across methodologies.

## Architecture and enterprise-wide support

A significant area of difference is the depth and breadth of explanation, guidance and techniques for establishing software architecture and how usable the methodology is for large-scale applications. CLIPP, CS/10,000, OPEN and James Martin + Co. provide explicit methodology support in this area. CS/10,000 has a knowledge base of 120 client/server architectures, which guide the user to architectural design. Team Fusion, Select and the beta version of Rational Unified Process also provide solid guidelines and tips, although not as comprehensively (based on the documentation supplied for this article).

Architectural guidance is critical for distributed objects and an important feature of object-oriented methodologies when large-scale applications are being delivered. For example, Jeff Morrison, Director of Product Development at Caldwell-Spartin, a software vendor providing software for the Staffing industry, found CLIPP to be the most comprehensive methodology. "Our project is a 22 man-year project with 500+ objects, 200+ user interface specs, using Orbix, Oracle, and Persistence software. We write 'contracts', which are specifications for class methods for both client and server to use. We are now able to do 87 screens in 6 weeks."

This productivity arose after the intense period of architectural design, guided by the CLIPP methodology materials and mentoring. "We spent the first six months building the framework and architectural components. Our application has lots of layers and wrappers. For example, we use MVC [Model- View – Controller] on

the client to promote isolation and portability. The application server has communication, business object, business controller, and data access layers. The CLIPP architecture materials give us guidance. CLIPP is very comprehensive – for example, it has UI, architecture, requirements and testing and more. CLIPP has provided us with the core for our development process which we believe is one aspect that sets us apart from all of our competitors".

Each methodology has its own 'spin' on what architecture is and how it should be layered. In the Rational Objectory/RUP there are layers for application, business specific, middleware and system software layers. CLIPP' has layers for application, technical, physical and data architecture. CS/10,000provides 120 architectures – all variations of partitioning strategies derived from its knowledge base. OPEN defines layers as strategic architectural units of design corresponding to a traditional three-tier architecture and suggests these layers should address elements such as application model, domain, server interface and relational database wrapper. Architecture is visually modeled in different ways. For example, OPEN and CLIPP use several dynamic and static architectural diagrams and CS/10,000provides visual client/server architectures for it's architectural designs. Layering of application architecture generalizes into interface, business, and data storage (see Figure 2), similar to the Smalltalk MVC (Model – View –Controller) pattern or Objectory/RUP's use of the as Entity, Control and Interface class stereotypes.

## Using use cases

Use cases or scenarios are the most commonly accepted technique for modeling functional requirements when using object-oriented methodologies. As a practitioner and facilitator who works with requirements, analysis, processes and deliverables, I have observed a great deal of confusion and differences in how use cases are handled. This same cornucopia of use case techniques is available from object-oriented methodologies.

Some methodologies (Team Fusion, Select Perspective, CLIPP, RUP) have helpful guidance and tips for use case modeling. CLIPP is particularly strong with its' approach to use cases, having as many as 17 deliverables in 4 categories: use case diagrams, specifications, catalogs, and verification matrices. Select Perspective uses roles based on DSDM (The Dynamic Systems Development Method - a public domain Rapid Application Development method). For Andrew Scott, Principal Consultant, PA Consulting Group IT Consulting Practice in the UK, this coupling of clearly defined roles, such as the "Ambassador user", along with use cases has been essential in the web-site project for Blackwell's, a UK academic book retailer. It enabled the team to define the functionality in layers. "The *Ambassador User* role represents the customer or business within the team. They help to define the boundaries of responsibility between use case layers".

In other methodologies, use cases are not mentioned (James Martin + Co.'s UML client/server object-oriented blueprint or Client/Server Connection's CS/10,000). The beta version of Rational Unified Process extends use cases in two directions: upstream into business modeling and downstream as part of user interface modeling (in the form of 'storyboards'). The OPEN methodology provides multiple techniques for requirements in addition to use cases, including noun analysis, task scripts and CRC cards.

## Components

One of the hottest buzzwords is *components* and unfortunately the term is used to mean slightly different things. 'Components' requires a precise definition. Consequently, object-oriented methodologies differ in their treatment of components.

The Catalysis approach defines a component as "an independently deliverable unit of software that encapsulates its design and implementation and offers interfaces to the outside, by which it may be composed with other components to form a larger whole" (ref #2). Rational defines component as "a piece of software code (source, binary or executable), or a file containing information, (for example, a startup file or a reassume file); a component can also be an aggregate of other components." (ref #3). These definitions, then, are oriented toward components as *implementation* units meant to actualize the promise of reuse and Lego™ -like plug and play.

In other cases, the term 'components' is used to mean partitions, pieces or parts – not software components. Catalysis in particular is component-*centric*, while CLIPP and Select Perspective are highly component-*oriented*. Rational's new Rational Unified Process uses the term but the workflow and deliverables are not component-oriented, except when addressing the UML component diagram. Some examples of how components are treated in object-oriented methodologies follow.

In *Select Perspective*, objects can be grouped into component packages, which are cohesive to the needs of a particular set of services. This is akin to the idea of subsystem partitioning using responsibility driven design (Wirfs-Brock, 1990). The component package provides the required services through its interface, acting like a 'wrapper' for the objects within the component package. *Select Perspective* uses the idea of a *contract class* to model the component's interface. The contract class manages service requests, delegates services to the objects within the subsystem, and encapsulates access to the subsystem, allowing the subsystem to wrap its objects. It can then delegate services to those objects, including resolution of polymorphism.

The CLIPP methodology defines components as "a group of related data and operations, organized to accomplish a specific purpose, with a distinct boundary

and an abstract interface, behind which the implementation details of the component are hidden" (ref #4). Using this definition, CLIPP treats systems, subsystems and classes as varying degrees of granularity of components which have the same nature and properties (e.g. interfaces, encapsulation/abstraction/information hiding, participation in static relationships via association, aggregation, generalization and dependency). CLIPP therefore promotes 'conceptual unity' between logical and physical views by using the same class notation for systems, subsystems and classes, with a simple change to the graphic to distinguish them. CLIPP therefore avoids the UML's package concept and views all software as components, regardless of whether the software is application software (created as part of system development) or technical (pre-existing software used to support application software, such as DLL, device drivers or ORBs).

Catalysis' 'component-oriented' approach relies on the core concepts of collaboration (behavior of group of objects), type (external behavior of an objects; type model) and refinement (trace abstract model to realization using packages to separate different levels of abstraction). At its heart, Catalysis sees the word in terms of components, using these core principles for object-oriented design. According to the authors of Catalysis, "The Unified Modeling Language (UML) and metamodel has adopted significant modeling constructs from Catalysis, including types, behavior specifications, refinement, collaborations, and frameworks" (ref #2). (Note: Catalysis is not a full methodology, as it does not include full lifecycle support)

## Age before beauty or Respect the legacy

One of the least mature areas of support in the object-oriented methodologies sampled for this article is how to plug, refurbish, wrap and/or connect existing legacy application into object-oriented applications. Most new development requires hooks to existing 'legacy' systems, if not conversion from old applications. More guidance should be provided for selecting applications that are better candidates for wrapping, where and how to specify wrapping mechanisms, and how to evaluate the design of interfaces to legacy systems. We can presume that good design constructs like coupling and cohesion are in important for legacy wrapping, but this is one area of disappointment.

Some exceptions are notable. Excerpts from the Catalysis (not yet released at the date this article was submitted) approach appear promising. Wrapping requires the 'wrapped' layer to have a well-defined interface, which is how the Catalysis approach precisely maps its' components. In addition, CS/10,000's methodology provides specific guidance for how to design legacy system wrapper objects.

## Business Rules

Business rules have begun to get appropriate attention as a modeling approach, methodology and/or explicit artifact in software development work (ref. articles). Businesses operate according to thousands of business rules which themselves are composed of other business rules. Business rules are the fundamental, underlying policies and constraints in the business. The OMG defines a business rule as " declarations of policy or conditions that must be satisfied".

Business rules are at the very heart of the structural and behavior models we build during analysis and design – regardless of whether the analyst is taking a data-oriented, process-oriented, object-oriented or component-oriented approach. These business rules provide a foundation for the complex policies and constraints that govern the daily operations of a business, which become implemented in code. Software systems run most businesses, or at a minimum guide the humans who make the business run.

Unfortunately, software applications in different portions of the business value chain were built in isolation. Business rules change constantly at a policy level, while the software used to implement them at an operational level does not keep up. Consequently, business rules and systems are mismatched and the rules are enforced inconsistently across the enterprise. Non-standardized business rules enforcement costs can be staggering (ref #5). Worse, businesses have been robbed of the thing they need most from software applications – the capability to change rules constantly and consistently.

The UML has defined the Object Constraint Language (OCL) for specification of ‘constraints’, which is based on first-order predicate logic. This is not useable by business people who are, after all, the source of the business rules. As methodologist James Odell notes, "syntax is a problem. Natural language is difficult to automate." The UML has a metamodel place for business rules. However, *capture and modeling* of business rules, *isolating* them in the application architecture (for reuse within and across applications) and *exposing* them to facilitate change at the business rule –level has not been addressed in any object-oriented methodologies.

Capturing, modeling, validating and specifying business rules in oo methodologies is largely ignored with several exceptions. Some methodologies use the term ‘business rules’ as part of what a business object encapsulates, e.g. part of one or more method’s responsibilities. The CLIPP, UML Client/Server Object-Oriented and OPEN methodologies do some justice to business rules. CLIPP specifies capturing business rules during use case modeling, documenting and tracing them in matrices. In James Martin + Co.’s UML Client/Server Object-Oriented methodology, business rules replace action diagrams and structured English. Business rules are specified according to Jim Odell’s business rule types: structural (derivation and integrity) and behavior (pre-condition, post-condition, trigger). Odell is one of the first object-oriented methodologist to pay appropriate attention to the importance of business rules.

The James Martin + Co. methodology provides examples of standard ways to write business rules, from Odell's work.

OPEN use the concept of software contracts, based on Bertrand Meyer's work, for specifying business rules. Contracts are found by examining requirements specs or scenarios. Business rules are categorized as constraints (policies and conditions) or derivations (inferences or computations). The business rules are incorporated into an object model by specifying class invariants, contract pre-conditions and post-conditions. OPEN provides even deeper business rule classifications, as well.

In all cases, however, the problem of single definition of business rules, business rule reuse, transforming the business rules into the application and technical architectures and full traceability is missing.

## Metrics

Object-oriented methodologies have grown from client/server methodologies to be more cognizant of the key role of metrics. Metrics are used for prediction (sizing and estimating the work) and for tracking and understanding quality.

If software development is serious business, metrics is a serious part of the process. Geneer is a customer software development organization. Says Geener Vice President and Chief Methodologist Todd Wyder: "We measure areas of the process that we are trying to change. Throughout the development process, we are constantly evaluating what is going well and what needs improvement. We look for root causes of these things and then develop action plans to make sure what goes well will continue to go well and what needs work gets addressed. We apply Goal/Question/Metric to track how well we are doing." Geneer's core philosophy is for teams to learn how to learn through the use of metrics. This is based on the Sheward cycle (plan – do – check – act) developed by TQM guru's guru (Deming's teacher was Sheward who taught this PDCA cycle as part of statistical quality control).

Geneer client Eli Lilly & Co. has adapted this view. According to Marty Morrow, Manager of the Software Factory at Indianapolis-based Eli Lilly & Co, "We view software development as a business. You need metrics to run a business. You have to have metrics. We're going to use a lot the same metrics (actual vs. plan, milestone vs. actual) as well as quality metrics (# of defects, amount of outstanding enhancement at turnover, customer satisfaction). In addition, customer satisfaction and employee satisfactions (the developers and analysts) are important. We plan to return to our projects 6 months post implementation, and look to see if they are using the application and what the business value is. That hasn't been done in past. The metrics are part of the process".

A similar emphasis on 'process metrics' is used at Caldwell-Spartin. According to Jeff Morrison, use case revisions, business rule and user interface specifications are tracked so they know roughly the time needed to complete these deliverables. Caldwell-Spartin built its' project management tool on the Web. They employ use case sizing metrics as well.

Other object-oriented methodology users are slowly evolving to the use of metrics. For example, Rational Objectory Process user at Skandia in Sweden estimate size by counting use cases and methods, calling them 'use case points' (like Function Points for use cases). Today, according to Johan Plogfeld, Chief Designer, they are not mature enough yet in their use of metrics and are tracking only dates.

Some methodologies provide sophisticated support for metrics as part of the software. CS/10,000 has an Estimator module as part of its knowledge base, which requires the user to answer a variety of questions about the degree of reusable components, complexity of the data, process and GUI, team experience and user interaction with the project. The result of this process is an estimate by WBS stage. Platinum/LBMS Team Fusion comes with a variety of estimation techniques, including Function Points and Object-based (# and size of the 'objects' e.g. classes, interactions, etc.).

## Tool Integration

Software tools for lifecycle activities are critical. For Caldwell-Spartin, tools integration is an important aspect of successfully using their object-oriented methodology. A variety of tools are used. Platinum's Paradigm Plus is used for visual modeling and to generate the framework for getter and setter code in C++, but they have now evolved the tool usage by adding scripts to increase code generation to about 60%. Matrices for verification of models are done in Microsoft Access (after having used Excel to begin with). Microsoft Front Page is used to publish requirements and user interface specs, use cases and business rules on their Intranet. Microsoft's Visual Basic is used to prototype the screens, which in turn is pulled into Visual C++ for final coding of the interface screen and processing.

Similarly, a variety of tools are currently used at Hewitt Associates, a compensation and benefits planning, consulting and Human Resources outsourcing concern. According to Sharon Seeder, Manager of the Development Methodology unit, some artifacts that they build are done in Word, Lotus Notes discussion databases, as well as the Rational Rose and Proforma Provision visual modeling tools.

## Roll your own

Some software organizations have found the ‘roll your own’ approach to be more effective than purchasing an ‘off the shelf" methodology. Commercial methodologies disappointed Sharon Seeder, of Hewitt Associates. "We found that most commercial methodologies were focused on system requirements and were missing business requirements. Without business processes, the use cases are too thin and don’t make sense. We don’t believe you should start at the system level." Hewitt had purchased a client/server methodology several years ago and found it to be "too theoretical; not enough ‘show me’ and ‘how to’s’", according to Seeder. At Hewitt, a more at ‘grass roots’ approach, supported by her unit with JAD facilitation and methodology, is "progressing at a steady pace", said Seeder.

These experiences are echoed by other large software organizations. For example, Marty Morrow at Eli Lilly comments, "The idea of buying a methodology as a way to a capability level and world class solutions delivery is not where we are now. It failed for us – we tried twice. We are trying to focus on establishing teams. Yes, we need a method and process, but the team will evolve and the method will be ours and it will work well. The critical path will evolve and as more reusable components emerge, that will move us forward. As our knowledge of the new tools evolve and our customers feel the difference in how we deliver solutions, we will evolve."

## Team Learning

A methodology must be ‘engineered’ by a project team to be a success. The capability of creating a fit-for-use methodology is not something which can be dictated to a team, nor is it intuitive to people. It requires time, experimentation, and team learning.

At Skandia, where the Rational Objectory process has been adapted for the past two years, new models called ‘program processes’ and ‘system processes’ were created to handle building new mainframe Cobol applications using the Objectory process, according to Johan Plogfeldt. Handbooks with examples and sample work products were created. On a weekly basis one of the five members from the methodology infrastructure group meets with a project team to find similarities in architectures, encourage adherence to Skandia standards, and to capture and show information which is stored in their library of patterns and source code.

Team members, according to Jeff Morrison, from Caldwell-Spartin must experience understanding how each role is vital to get the job done. He has each team member take on another team member’s responsibilities as a brief learning technique. "This enables them to see the impact of their work product on the full lifecycle," says Morrison.

At Geener, a ‘post-portem’ is done after each iteration is delivered, according to Wyder. The QA team member solicits input from each team member about the

three things that went well and didn't go so well during the iteration. These are collated and returned back to the team, which then decide which ones to focus on during the post portem meeting. During the meeting, the team analyzes the root cause for both successes and failures, decides what needs to be improved, establishes improvement goals and then using the Goal-Question-Metric (GQM) technique, the team establishes metrics for the next iteration. Lessons learned are captured in the Geneer best practices database for reference by all employees. "We believe this is the key kernel to our process – the outputs of those meetings. Process is organic and how it changes is bottom-up. With our methodology, there are no sacred cows. The enforcement comes through pain. That's how learning happens", says Wyder.

## Conclusions

The topic of methodology often generates debate, not just about which is 'best' but also about the utility or futility of using a methodology. "We've had a 40-year struggle between hacking and disciplined development", observes Tony Wasserman, of Software Methods and Tools. "The

'let's just build it' approach is winning now, due to the Internet and speed driving projects. We're under-sourced, for which Year 2000 is a factor, and companies are saying – 'here's a ship date' and they need to get the software into the distribution channel despite bugs. In fact, the cost of screw-up in software is not high – they can send out a patch via Internet! So fewer companies are willing to use disciplined development".

It is true that getting organizations to consistently use an organized software development process is a battle. Success in this area is found, as Wasserman points out, in companies who are building systems, not software (e.g. systems that run cars, cell phones, jet engines). These organizations *must* have mature processes, because their business depends on it. Similarly, companies that build software for other companies must also have a mature, repeatable process.

This situation, however, is juxtaposed by the reality of scalable systems. As Wasserman points out, object-oriented concepts are now common in young software developers (many universities are teaching Java now). However, these new developers don't understand what it takes to build a scalable, system in a real production environment with lots of legacy applications to draw from or interface with.

The process of software developments continues to become more complex. Distributed systems, use of the internet, the need for multi-layered architectures, the attraction of modeling software like business itself and the constant change in software product content, features and ownership, make software development hard work. Object-oriented and component-based methodologies may offer evolutions in this capability to many IT organizations. Selecting a methodology is

complicated by the continual changes the contents undergo, but major considerations are the cultural fit, the lifecycle support that is required and the availability of mentoring, training and consulting support. The sensible thing to do is pick a decent methodology, learn it, and learn how to adapt it. The end product will ultimately embody the quality and stability of the process from which it was developed.

## References:


1. Henderson-Sellers, Brian, "OML: Proposals to Enhance UML", Proceedings <<UML>> '98 Conference, Mulhouse, France, June 1998.
2. D'Souza, Desmond F. D'Souza and Wills, Alan, Objects, Components and Frameworks with UML: The Catalysis Approach, Addison-Wesley, 1998.
3. Rational Software: Rational Unified Process, beta version.
4. Platinum Technology/Vayda Consulting: CLIPP™ : A Scalable Approach to System Modeling, Release .5, version 6.
5. Gottesdiener, Ellen, "The Value of Standardization of Business Rules", Object Magazine, March. 1998, 8 (1).


## For other Information:


Allen, Paul and Frost, Stuart, Component-Based Development for Enterprise Systems: Apply the Select Perspective™ , SIGS Books/Cambridge University Press, 1998**.**

D'Souza, Desmond F. D'Souza and Wills, Alan, Objects, Components and Frameworks with UML: The Catalysis Approach, Addison-Wesley, 1998.

Odell, James, "Is Method Unification Possible?" Distributed Object Computing, May 1997.

Gottesdiener, Ellen "Business Rules Show Power, Promise", Application Development Trends, March, 1997, vol. 4, no. 3.

Gottesdiener, Ellen "Methodologies Evolving: What's New in Client/server Development Methods", Application Development Trends, November, 1994, vol. 1, no. 12.

Gottesdiener, Ellen "What's Your Development Maturity?", Application Development Trends, March, 1996, vol. 3, no. 3.

Graham, Ian, Henderson-Sellers, Brian, and Younessi, Houman, The OPEN Process Specification, Addison-Wesley, 1997.

Firesmith, Don, Graham, Ian, and Henderson-Sellers, Brian, The OML Reference Manual, SIGS Books, 1997, now available from Cambridge University Press.

Henderson-Sellers, Brian, Simons, Tony, and Younessi, Houman, The OPEN Toolbox of Techniques, Addison-Wesley, 1998.

Software Methods and Tools Web site:

www.methods-tools.com

Hewlett-Packard's Fusion web site:

www.hpl.hp.com/fusion/file/teameps.pdf

Advanced Development Methods SCRUM methodology web site:

http://www.controlchaos.com/

## OO Methodology Sampler Sidebar

| Vendor | Product/Methodology Name | Website &/or References | Availability | Phases |
|---|---|---|---|---|
| Platinum Technology | Process Engineer/Continuum: CLIPP: Complete lifecycle incremental process | http://www.platinum.com/ | commercial | Requirements, Architecture, Analysis, Design, construction, SQA |
| Platinum/LBMS (LBMS is now a subsidiary of Platinum Technology | Process Engineer/Continuum: Team Fusion | http://www.platinum.com/ | commercial | Project Initiation, Solution Definition, Evolutionary Delivery, Distributed Deployment |
| Geneer | Code Science™ | http://www.geneer.com/ | Part of partnering with Geneer on customer software development projects | Analysis Workshop, Architecture, Construction, Delivery |
| Rational Software | Objectory; now called Rational Unified Process, RUP (RUP to be commercially available by end 1998) | http://www.rational.com/ | commercial | Inception, Elaboration, Construction, Transition |

| | | | | |
|---|---|---|---|---|
| | | | | |
| Icon Computing (now a Platinum Technology subsidiary) | Catalysis | http://www.iconcomp.com/ | Book, | Requirements, System Specification, Architectural Design, Component internal design |
| James Martin & Co. | UML Client Server/Object Oriented Process | http://www.jamesmartin.com/ | Commercially as standalone (passive) 'Processware' or along with the Platinum/LBMS Process Engineer engine sold by JM + Co. as the Architect product | Cs direction (object activity matrix is really crud matrix), outline Business Analysis, Client/server Design, Client/server Construction, Deployment |
| Select Software | Perspective | http://www.select-software.com/ | Book, processware, process with CASE tool | Feasibility, Analysis, Prototype, Plan Increments to Deliver, Design & Build |

| | | | | |
|---|---|---|---|---|
| | | | | Increment, User Acceptance of Increment, Rollout of Increment |
| OPEN Consortium | OPEN | http://www.csse.swin.edu.au/cotar/OPEN/OPEN.html<br><br>Several books, including: Graham, Ian, Brian Henderson-Sellers, Brian, and Younessi, Houman, The OPEN Process Specification, Addison-Wesley in October 1997. | Public domain | Project Initiation, Requirements Engineering, Analysis & Business Model Refinement, Build, Evaluate, Implementation Plan |
| DSDM (Dynamic Systems Development Method)Consortium | DSDM: Dynamic Systems Development Method | http://www.dsdm.org/ | Public domain RAD methodology | feasibility study, business study, functional model iteration, design an build iteration, implementation |
| Client/server Connection | Client/server 10,000 | http://www.cscl.com/ | commercial | Define, Plan, Prototype, Implement |

Figures follow:

Figure 1:

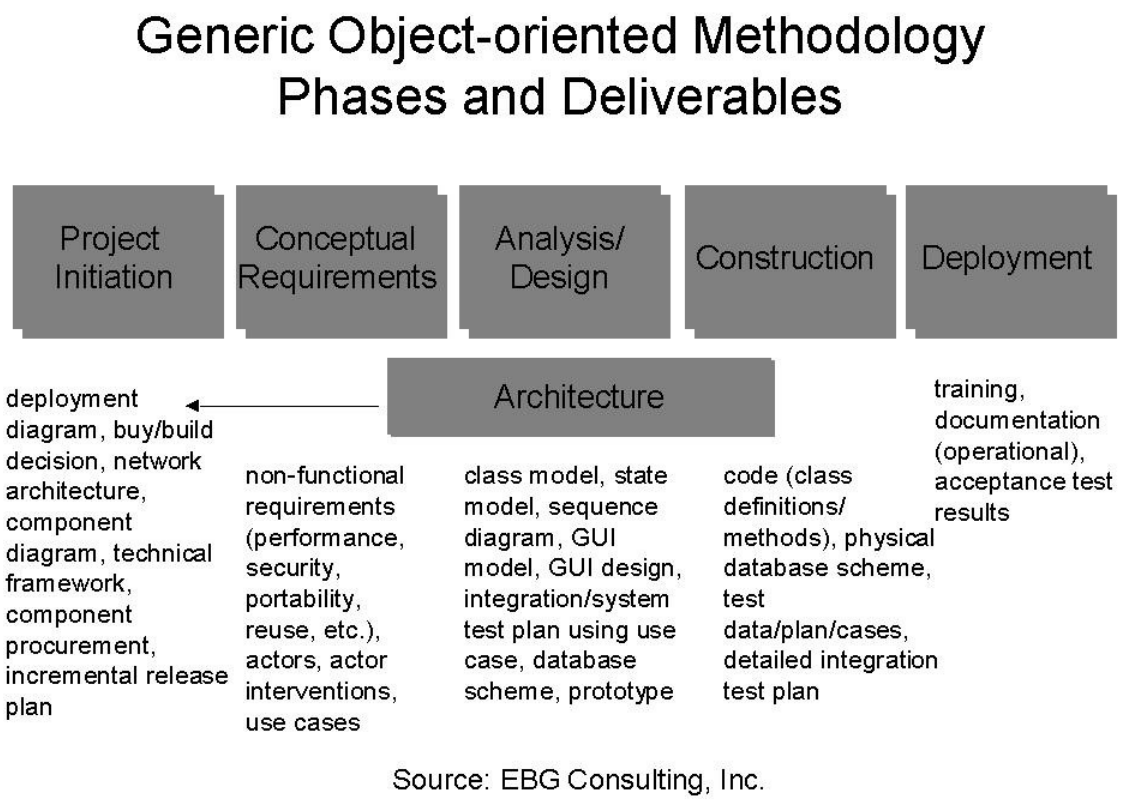
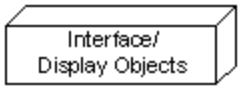
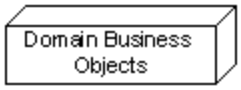



Figure 2:

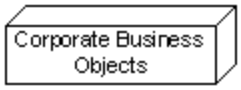
# Generic OO Architectural components



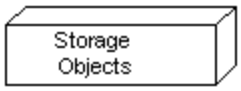

Screens, menus, dialogs




Conceptual objects specific to application domain




Cross-functional (reusable) conceptual objects




Encapsulated data (RDBMS,OODBMS, legacy data);

Source: Select Software