

What they ARE ... and what they are NOT!

OBJECT-ORIENTED METHODS

By Charles A. Tryon
Tryon and Associates

Some years back, a college professor asked me what I thought about Object-Oriented (OO) Methods. Without waiting for my reply, he gushed that he believed them to be “new, new, all new.” When he paused, I replied that, “if you don’t know anything about established, mature development methods, then OO Methods are indeed new.”

My response was not intended to be flippant nor condescending. It was and is simply the truth. Yet today, many people still harbor the false impression that OO Methods are to software development what penicillin is to medicine ... a breakthrough that changes everything and they are “new, new, all new!”

Don’t get me wrong. I am not against Object Oriented concepts. In fact, I’m just the opposite. I’m a strong believer in them. But I’ve also been around long enough to know that just because a variation on a method hits the software development arena, it doesn’t automatically make everything in place before it obsolete.

Adamant statements are often heard with many of them coming from people who have little, if any, knowledge of the software engineering methods that preceded the current incarnation. Often times, these arguments are fueled by authors who hope to hook onto the latest trend to sell a few books. So battle lines are drawn around misinformation and misunderstandings. Reason and facts are lost in the process.

The intent of this paper is not to completely resolve the debate that rages in many development organizations over OO Methods. But I do hope to bring some balance to the discussion. I believe there is ample middle ground, not as a political compromise, but as a logical and efficient strategy that should be pursued by modern software development organizations wishing to incorporate a complete requirements process.

Are OO Methods a “Fad?”

Software development became a formally recognized act in the late 1960s and early 70s. The industry began with highly technical programmers manipulating machine-level instructions on rudimentary processors. We now have systems engineers who capture complex business requirements and create automated solutions using multiple technology components operating in a variety of environments.

Along the way, a host of software development techniques have surfaced. Many were specific to the technology of the moment and some were just bad ideas. But along the way, a handful of methods survived to transcend both technology eras and skill changes to be true systems engineering methods. Object Oriented Methods are the latest addition to this elite collection of development tools and techniques.

OO Methods are a set of philosophically sound, proven, software engineering methods. There are a few distinct variations suggested by different authors and organiza-

tions, but at the core, OO Methods provide the greatest opportunity for reuse of software code and easier software maintenance of any approach to date.

OO Methods accomplish these lofty objectives by respecting the principles of *simulation* (using models to represent an understandable view of a system), *encapsulation* (bundling into a specific object the meaning and use of data associated with the object), *classification* (organizing objects that share some common behavior) and *inheritance* (attributing a lower-level object with the characteristics of a related upper-level object).

Objects that are identified, documented and refined based on these principles lead to clear initial implementations and simplified future software maintenance and enhancements.

OO Methods Heritage

The reason OO Methods deserve recognition as a significant software engineering discipline is due largely to its heritage ... something unknown or unrecognized by many current OO practitioners.

OO Methods, like other software engineering approaches, began due to a significant technology awakening. In the OO world, it was new languages like Simula, Modula, Ada and Smalltalk. Today C++ dominates the market. These languages provide a way to create graphical interfaces at a pace unimaginable with more traditional, procedural languages.

Various authors quickly proposed Object Oriented Design methods to leverage the power of these new languages. More recently, many OO thinkers have collaborated to establish UML or a Unified Modeling Language.

While the languages and designs are revolutionary, many of the core concepts found in OO Methods are anything but new.

For instance, the concept of encapsulation, a critical pillar of OO Methods, was proposed in the 1970s practice of Structured Design as defined by Larry Constantine and Meiler Page-Jones. To counter the global data treatment of COBOL, these authors suggested *data clusters* or *information hiding modules*. These software designs required special data definition in Working Storage and a variety of distinct program units that would create, use and remove the elements of this defined data. Further, Constantine's Structured Design principles of *coupling* and *cohesion* are now part of OO Design.

The very concept of what makes up an object was drawn intact from the Strategic Data Modeling definition of an *entity*. Data Modelers trained in strategies initiated by the likes of Matt Flavin, Ira Morrow and Peter Chen have at least two decades of experience refining information about entity identification and definition that are considered "discovered knowledge" by OO practitioners.

The most significant "new idea" of OO Methods is the *Use-Case* to provide a flow for related objects to solve a specific problem. Yet the author of Use-Case, Ivar Jacobson, attributes this "discovery" to Steve McMenamin and John Palmer's text on *Essential Systems Analysis*, material that is at the heart of Structured Analysis, a process modeling discipline first defined by Tom DeMarco in the mid 1970s.

OO Methods offer significant advantages for modern software developers. However, few of the methods and techniques are truly "new." They are extracted and derived from established systems engineering precedes-

sors. This should not be seen as any type of disadvantage. By understanding these earlier methods, OO practitioners may learn how to utilize current tools without constantly rediscovering old knowledge.

Are OO Methods Anti-Main Frame?

Software Development methods are commonly and unfortunately linked with the technology that prevails at the time they are introduced. Such is the fate of OO Methods. Many conclude that because OO Methods emerged during the GUI driven era of Client/Server implementations or PC-based products, then OO Methods must apply only to those environments.

OO developers know, however, that current OO strategies are targeted toward large computer installations as well as smaller hardware configurations.

By the same token, some OO zealots have been fooled into believing their methods represent such “new technology” thinking that all older approaches are as obsolete as a 10 MB hard drive.

This view, often reinforced by uninformed OO authors, vendors and classroom instructors, frequently ridicules and falsely associates more mature systems engineering methods with obsolete thinking. They attempt to justify OO Methods by tearing down the very techniques that gave rise to Object Oriented concepts.

This robs new practitioners of the history and foundation for the methods they are attempting to practice. It forces continued “discoveries” that are not new. And new developers are more susceptible to false directions that have already proven to be ineffective. Unnecessary conflict flares between development groups within an organization

as developers battle for the supremacy of their favorite approach.

Do OO Methods Cover the Total Development Process?

This depends on what you consider the total process and who you talk to. To some, this is an easy and resounding “yes,” but as they say on the football field, “Upon further review...” So here is my view on this rather contentious topic.

With every new method seems to come a desire, usually driven by supporting vendors and their technically challenged marketing organizations, to replace all existing knowledge with something new. “New and improved” has been a proven sales strategy for a long time. The OO industry is no different.

While introducing a truly new approach for designing and building software, OO authors, instructors and supporting organizations have also produced papers, books and seminars on such things as Object Oriented Project Management and Object Oriented Analysis.

Project Management for fast moving software construction organizations is quite different from the more stable version used for home construction and assembly lines. However, OO Project Management is no more valid than Project Management for College Students Who Are Left Handed. A few new items in a Standard Task List and a handful of important deliverable names do not justify “reinventing” an entire discipline. Declaring a new variant on an established method as the “latest and greatest” is a rather common and misguided marketing scheme.

Even more contentious is the legitimacy of training on Object Oriented Analysis. I be-

lieve that most of the confusion surrounding this topic stems from the definition of ... *analysis*.

For many, analysis is considered a brief activity that results in a limited collection of notes talking about general business needs, often with a predetermined solution in mind.

For more mature organizations, analysis has evolved into a highly rigorous discovery process that focuses on what a business is attempting to accomplish. Models are created that guide a rethinking of the uncovered business processes and data.

The goal of these efforts is to study business

requirements, in detail, free from the bias or constraints of *any* solution or implementation. Business requirements are evaluated from the initiator of the need, through the total business process and to a final set of responses. True analysis examines this information so that multiple, simultaneous and highly distinct solutions may be designed and created ... one of which may well be an Object Oriented implementation.

Attempts to assume an object perspective early during the study of business requirements biases the understanding of these business needs toward an OO solution. This serves to complicate any non-object components and limits scope to only those things

The Difference Between Analysis and Design

Analysis is not design. It took almost twenty years for systems developers to fully appreciate this distinction. True analysis is more than just studying a customer's problem. It is doing so without the bias of any solution or technology. Design is formulating a solution that solves the problem and works within the constraints of a specific technology or environment.

Making such a strong distinction between the two is not a limiting factor. In fact, it allows analysts to focus on the business requirements without attempting to simultaneously consider all the solution options. It protects the design process from poorly considered solutions that apply to only a fragment of the total answer.

OO Methods are design methods, by definition, as they are already collecting business requirements into design ready containers that will be constructed ... objects. This approach also considers a fixed scope of automation within the business where true analysis wishes to capture a larger study domain. Understanding a broader scope allows us to find critical components, not originally considered, that must be part of an effective business solution. Comprehending the business context for automation also allows developers to create more meaningful interfaces into the world supported by technology.

As a result, true analysis techniques should be applied before any type of solution specific strategy. Once true analysis is complete on a project, multiple design methods should be considered to determine which one best fits the solution for the business.

By creating true analysis models using Structured Systems Analysis and Data Modeling, designers, including OO practitioners, are able to focus on extracting critical design components and not discovering them.

OO Methods address.

A key point of maturity in OO Methods was the recognition of Use-Cases. Derived from the study of Business Events in Structured Analysis, this approach identifies an *actor* who initiates a request to an automated system, launching automated work and some automated by-product. Use-Cases have enabled OO developers to organize and structure relevant objects with their associated processes and data definitions to enhance clarity and completeness. To many OO developers, this is analysis.

What Use-Cases fail to address are that not all business requirements will be automated. A business system is composed of automated *and* non automated processes and data. Before defining *any* solution, we should capture detailed business requirements without the restricting bias of *any* technology or solution type. Failure to do so limits design choices available to a developer. OO Methods, by definition, cannot do this.

If the total goal of a project is to automate a predetermined subset of a business, without understanding the context of the solution, OO Methods may provide a limited approach for narrow application design and construction. But this is NOT true analysis. It is “design analysis.”

Challenges Facing OO Methods

Many believe OO Methods to be some form of panacea for all things related to systems building. An honest appraisal shows a far more realistic view.

Complex Models – The models required to represent the intricate structure of an OO solution are highly technical and difficult to interpret. While very rigorous, they rival

Data Model Diagrams for complexity. These technical blueprints are valuable for both software construction and documentation. They are, however, very challenging for a business customer or non-technical person to understand. Any attempt to simplify the diagrams actually reduces their effectiveness for design.

Unfamiliar to Business Users – Technical authors often attempt to sell a new strategy around the idea that it appeals to the user’s natural way of thinking. Numerous OO authors have tried to justify similar positions. What they ignore is that business organizations do not see their world as fragmented automation.

Business systems are not comprised of static, stationary data surrounded by isolated actions against that data. The subject matter experts who relay requirements to systems analysts view their world through the work they do. They see business as being something alive or active. They understand the use of data primarily in the context of what is happening in the business process. While customers often recognize data structures and the actions against these structures, viewing such information independent of the total business process is out of context and open to many misunderstandings and omissions. As a result, any type of user-friendly model must depict a business system as a flow of work along with needed information to complete the work.

That is why established process modeling methods such as Structured Analysis have proven effective and useful. They model the business through the eyes of the customer. OO Models and Data Models find this a significant struggle.

Limited Automation – A significant danger of using OO Methods exclusively during

systems development is the misnomer that a “system” is comprised exclusively of automated components. In fact, the true business system extends far beyond automation. Due to the automation bias of OO, the results easily become fragmentary and out of context to the business. This can also result in missing technical components.

Structured Analysis and Data Modeling make no early distinction as to which process or data components have been or even will be automated.

Premature Design Decisions - OO Methods assume an OO implementation. During analysis and external design, the job of a developer includes finding a solution for the business that naturally fits the stated needs. This may include a purchased package. It may be composed of non-OO components. It may also use non-automated procedures. It is highly questionable and limiting to use an analysis or design strategy that prematurely locks a development team into a specific type of answer, no matter how good that solution approach might be.

Repository Dependency – For decades, knowledgeable developers, authors and instructors have encouraged software reuse by capturing software components and then “reusing” them in future applications. OO Methods demand such an approach. An *object repository* provides a centralized location for all objects and related information. This information is then refined and updated when it changes or new information becomes available. All teams across an organization must commit to using this repository.

In practice, few do. Developers often redefine as “new” existing objects and store them in local repositories. When this oc-

curs, most of the expected benefits from OO Methods are lost.

Long-Term Yield – Using OO Methods are not initially faster than traditional development approaches. OO Methods represent a serious engineering strategy that requires time, discipline and talent. An initial pass that digs out objects and their characteristics will take as long or longer than traditional programming approaches. Any “savings” are found in creating future applications from objects previously stored in the object repository.

The Future of OO Methods

OO Methods will remain a dominant design strategy for years to come. Will OO Methods continue to evolve? I believe they can and will. Part of this evolution will include linking OO Methods to established analysis tools and techniques. UML is helping to shake the exclusive OO bias and move toward technology independent requirements.

OO practitioners may eventually “discover” and commit to analysis approaches that are solution neutral and useful for any solution type. Such a discovery will push the OO community out of their exclusive “OO only” world. Several books and papers, written by OO advocates, are already suggesting such an approach.

But true analysis methods already exist and are currently available. Structured Analysis with Business Events allow us to study the processes critical to the end customers. Data Modeling helps us find obscure entities along with the definitions and rules that govern the entities. I believe OO thinkers would be well served to find bridges from those methods and support a broader requirements management strategy.

Organizational Implications

All mature software development organizations strive to find methods that will guide the creation of high-quality software products. OO Methods must be included in an organizational toolkit. But this should not be approached as a replacement strategy. Technology advances have burst open the door to multiple, distinct answers for the same business problem statement. OO Methods support a significant portion of modern designs ... but only a portion.

Organizations must resist yielding to the latest and loudest voices for any method. It is often difficult to distinguish enthusiasm

from truth. Misinformation, when stated often and forcefully enough, tends to be accepted as fact. A broader view will prevent accepting an artificial standard that applies to only a subset of the challenges facing software development.

Success will be found by integrating a variety of methods, leveraging their natural strengths into a set of best practices that will result in quality solutions for customers' needs. These methods must cover the total development process, beginning with true analysis of business requirements, through design and into software construction.

The Tryon and Associates Approach

Tryon and Associates has offered Process Modeling training and consulting since 1985 to some of the largest organizations in the United States, Canada and Europe. This three part series of courses is intended for systems analysts and business analysts who are helping their clients define business requirements with the intent to improve the total business through the use of new technology.

The Tryon and Associates Process Modeling curriculum has been crafted for this audience with concise, focused material that provides the type and level of knowledge needed to study business processes. Modeling Business Processes provides an in-depth understanding of Data Flow Diagrams, Mini-Specifications and the Data Dictionary. This seminar also identifies how these tools fit together to create a rigorous specification. Reengineering Business Processes explains the transition from the current view of business requirements into a model of a new solution. The seminar uses Business Event Modeling as the primary means for defining business requirements independently of any particular solution. Designing Application Systems completes the transformation of business requirements into a certifiable model of the future automated component of the solution.

The right amount of education . . . and at an affordable cost.

This paper may be reproduced for internal, non-commercial purposes without the consent of the author. The paper must be reprinted in its entirety with this notice and all copyrights shown. Any commercial use of this paper must be approved by Tryon and Associates. Additional information on Tryon and Associates seminars is available from our website at www.TryonAssoc.com or by calling 918-455-3300.