Modular Programming Versus Object Oriented Programming (The Good, The Bad and the Ugly)

An Article Written by Stéphane Richard (Mystikshadows)

INTRODUCTION:

I'm sure some of your, by now, might be saying "what is that all about?". Allow me to begin this with a brief history of why I decided to write on this subject. You see, for decades already, when I hear talks about programming concepts and paradigms, It seems that Modular Programming techniques and object Oriented Programming have been under somekind of comparison by developers all around the world. A hidden feud between the two serving the soul purpose of boasting the advantages of one over the other. Also, It seems that with time, O.O.P. took on a very different role, one that radically changes the way newcomers to the world of development see and understand the concept. However, if you go on C forums, QB forums, and other languages too, and talk about O.O.P. you might be surprised at the answers you get from those communities.

Basically, these two items are what motivated me to write about the subject in question. NOte that the contents of this document represent my own point of view, my personal understanding of the concepts and my thoughts on what should be what. But i have seen the evolution of Modular Programming and Object Oriented Programming from their very beginning, I've also seen all the deviation that these programming methods took in the course of their evolution. I will be explaining these methods and share my views and insight on where each method can provide the best advantages on a project. This will give you the information you need to understand the rest of the document where I will be sharing what has been said about these methods and show you where the deviations that was bestowed upon the O.O.P. paradigm changed everything that O.O.P. is known as today.

WHAT IS MODULAR PROGRAMMING:

Modular (or Procedural) programming is a coding method that entails the use of routines, sub routines and functions in order to organize and execute specific sets of instructions pertaining to a given task to be done. The main rule of thumb in this technique is that if a set of instructions, that perform a specific task or calculation, will be called several times, it should

be moved to a subroutine (if it does not return a value) or a function (if it does return a value) and called from the needed other parts of the program.

Typically, to create a successful modular structured approach to solve a problem, you need to define the problem in terms of what actions or calculations the program needs to do in order to perform the task defined in the problem. You also need to see if there is any special connection between procedures and functions to establish an execution order for them. The other concept here is that if a subroutine or a function is quite big, it is probably due for a refactoring into two or more sub routines. This is essentially how modular programming manages the complexity of a program by breaking it down into smaller, simpler, more managable sections of code.

Over the years, structured programming has been given more than one coding style and as such gives a good foundation to adapt the programming method to fit one's minding. By that I mean that depending on a developer's background he or she might like to organize their modules, procedures and functions based on functional grouping (all string manipulation functions go in one module, all datatypes go in another module, all constants in yet another module, this is often the type of organization I like to follow as well. Others might prefer to organize their modular approach based on an entity structure (somewhat closer to O.O.P.). For example, all datatypes, procedures and functions that affect an employee's information goes in one module, all inventory datatypes and functionality goes in another module. This is one of the great advantages to the modular programming approach in that it allows to organize the code in a way that fits the developer's way of thinking. When you think about it, it can be a very strong point since everyone thinks in very different ways but everyone needs to be able to define their problem and bring solutions based on how they would solve them manually.

WHAT IS OBJECT ORIENTED PROGRAMMING:

Object Oriented Programming is a coding method that entails the use of objects and their relationships in order to describe, programmatically, the problem to be solved. The classic definition of O.O.P. was based on three founding pillars which were Encapsulation, Inheritance and Polymorphism. Here is a brief explanation of these founding blocks.

- **Encapsulation:**
  This term defines the fact that both the data, and the functionaly that could affect or display that data are both included under a unified name (the object name itself). In the classic definition, the data elements (or properties of the object) are not available to the outside world directly. Instead, methods would be created to give access to these values outside of the object. Quickly, most languaged that supported Object Oriented Programming added the ability to declare properties as being public or private which took away from the original definition of keep that data private to the object only.

- **Inheritance:**
  This feature allows developers to define objects in a hierarchy much like a taxonomy chart (like the animal kingdom classification chart). Each level of the hierarchy defines a more specific object than the parent level. Each level inherits all the properties and methods of it's parent object and at that point you define the more specific properties and methods need by the new level of object you created.

- **Polymorphism:**
  This is a very fancy term to say that at any level of an object hierarchy each object could have a method of the same name and because of the level at which the method resides in, it could know which of the procedures or functions to call. Hence you could have a Shape object that has a Draw method. Then you could define a Circle, Square and Triangle object as Shape objects and override the Draw method to draw specifically a Circle, Square or Triangle method respectively. All 4 objects would then have a Draw method but only the right method for the right object would be called.

In the classic definition of O.O.P. These founding blocks were created because of the birth of a need for code reusability without simply cut and pasting existing code into a new module. Hence you could create an object or a hierarchy of objects that pertained to a specific role. then give the compiled version of that object as a library to another person who could then create his own objects as being part of the originally defined hierarchy. Today, two new pillars have been added to the definition of O.O.P. They are Interface and Abstraction. Here are their definitions.

- **Interface:**
  This was added to O.O.P. in the goal to allow developers to provide an object template to control the access and definition of objects. An interface is simply a list of fixed method names that any object defined as being of this interface type must implement in order to be qualified as a valid object of the type the interface implies. This basically gave developers the ability to strictly control the naming conventions used in their objects as well as objects created by other developers designed to work in the current object model.

- **Abstraction:**
  As popular as this term now is in the world of Object Oriented Programming, this essentially goes against all programming techniques. Abstraction is the ability to not need to pay attention to the type of objects that are bring used if the developer sees fit to do so. For example, a Dog could be treated as a dog probably most of the time, but of you have wolves, foxes and other similar dog like animals in your hierarchy, you could treat them all as Canidae class objects if you'd need to apply changes to all 3 animal types (provided these animals are of declared as Canidae type objects first. From Abstraction was born the need for the know infamous Variant datatype and the ability to declare variable as the ANY datatype.

Essentially, O.O.P. is an evolving standard that adapts to new needed concepts as they are needed. This can be fun in some cases, but a real curse in most other cases. These last two pillars also created the concept of black box and white box programming techniques. The classic object was White box (which means that levels of the hierarchy needed to know what came at the parent level in order to be able to perform its task. In White Box development, a hierarchy defined levels of specializations of objects in a hierarchy where objects in a lower level had more refined programming compared to it's parent. In Black Box development, Each level of the hierarchy tends to define containment rather than refinement. For example: The white box would state that a dog is a specialized (or refined) type of Canidae. In Black box, a Canidae has a dog (and a wolf, and a fox as it's sub components. This can lead to some serious conflicts in my opinion, which i will share later in this document. For now, let's compare Modular Programming and Object Oriented Programming versus a typical problem scenario to get a glimpse of the real differences of both methods.

METHOD APPROACH COMPARISON:

The Scenario:

The scenario I'll be using is the following. The boss comes to see you and asks you: "I would like a program that can save information on the employees to a file, I would like to be able to enter the Employee's ID, his name, address, city, state, zip code, telephone number, email and hourly rate and it should also be able to to retrieve that information based on one or more criteria that I would enter in different fields. I would like to print that information so I can put the employee information in a physical file. The employee should be able to to enter the time they arrive at, the time the leave and come back from lunch and the time they leave at the end of the day in a timesheet program. The program should be able to calculate how many hours they have worked that week, the gross pay they have gotten and the amount of overtime they worked (as well as what monetary value the overtime amounts to. Employees are paid 1 1/2 times their hourly rate for any hours over the regular 40 hours a week and they are paid twice their rate on any holiday. Everything should be saved and retrievable on a per employee basis. I also want weekly, monthly and yearly detailed and summary reports on all this information for all or one of the employees at a time."

The Modular Programming Approach:

In the modular programming approach, the verbs in the problem description are the ones considered first. Typically, these verbs represent the work to be done and therefore the likely subroutines that we would have to implement in order to solve the problem. The names in the description represent the information (variables and datatypes) that will will be acting on or producing at the end. Modular programming is based on three things, which are: 1. The information needed before the process can start, 2. The work to be done with that information and 3. The expected output of a procedure. We will here answer these questions as per our scenario above:

- **What Information Is Needed:**
  By reading the problem description, we know already the information we need about the employee, so I won't define them here. However, we need to define the information about the timesheet itself. The timesheet should have the curent date, the start time, the lunch time start, the

lunch time end, and the days end start. All this information should be linked with the employee itself by adding an EmployeeID field to our definition since each employee will be entering their own time.

- **What Is The work to be done with that information:**
  Again based on the problem description, The program should allow us to enter and save employee information to a master employee file. It should also allow entering and saving of timesheet data into a timesheet database file. When needed, the system should allow to enter an employee ID or Name and show the user the week's total hours worked, gross earnings, the number of hours in overtime and the earnings from that as well. it should calculate the total hours and the total amount earned for that current week. The report should allow the user to specify a range of date for either a week, a month or the whole fiscal year and an optional employee ID to print out the payroll information for either the specified employee or all the employees (if none were entered).

- **What Is The Expected Output(s):**
  The first out is the employee's information that needs to be printable on paper. it is mentionned and should be coded for. The other printable information that are stated are the weekly, monthly and yearly reports on one or all the employees. There is another piece of printed output that is understated here but not literally mentionned, the timesheets themselves. It makes sense to give an employee his weekly timesheet at the end of the week and/or for the employer to request an employee's timesheet at the end of the week. So we would typically implement a procedure to print timesheet information on request.

As you can see, the modular approach is closely related to the functionality needed in the problem description. You think in terms of what needs to be done and usually can pretty quickly devise a workable complete solution to a problem by consecutively answering these three questions (some of these answer may require that you breakdown the system into smaller answers to a subset of these three question depending on how complex the system gets. This breaking down into smaller more specific procedures and functions is how Modular Programming offers to manage the complexity of a program.

The Object Oriented Approach:

In Object Oriented Programming, the names, not the verbs, are the first to be considered because in O.O.P. the first task is to define entities, not actions. O.O.P. starts by defining the players (or actors if you will) of a given scene (the problem) and them proceeds to defining how the actors are described (the properties) and what the actors can do (the methods). Once all these are defined for all playing actors, then the scene (the main part of the program controls what each players has and what it does. Only when you are defining the methods of the object can you actually start answering the same three questions that the modular approach lets you define right from the start. Again, as we did in the modular approach, I will define the same problem description using the Object Oriented Programming Approach instead of the Modular Approach I used previously:

The Entities:

By reading the problem description. We can clearly indentify three entities. The Employees, the Timesheet and the reporting system. This is how O.O.P. problem solving begins. You need to identify these entities first and the define how you describe the object as well as define what the object can do. So let's take each entity and define them so you can see how things work:

- **The Employee Entity:**
  The Employee is the main actor of the scene. Every other entity in the model revolves around what the employee is and what it can do. In any object model you define, it's always important to identify this key player because it is at the center of any object relationships that can exist. Here are the defining attributes (properties) and Related Functionality (The Methods) of the Employee Entity.

    - **Defining Attributes:**
      As mentionned in the description, the Employee entity will need an EmployeeID, his name, address, city, state, zip code, telephone number, email and hourly rate attributes.

- **Related Functionality:**
  The Employee needs to be able to do several things as far as it's defining attributes go as well as the ability to call the functionality of the other entities for timekeeping and and calculations. As such, the methods needed will be. EnterEmployee, SaveEmployee, LoadEmployee, FindEmployee and PrintEmployee.

- **The TimeSheet Entity:**
  The TimeSheet Entity should be made to work a week based format where all the days are there so that it can be easy to get a quick overview of what the week is like currently. We will add a method to our object to make sure that an employee's weekly data is completely entered before we go ahead and print the timesheets just as a precaution because of the importance of the data that is being handled.

  - **Defining Attributes:**
    When you think of a timesheet, it doesn't take too long to determine the general information that you would need. The properties are: EmployeeID, DayOfWeek, DayDate, StartTime, LunchStartTime, LunchEndTime, EndOfDayTime. EmployeeID is needed to connect a timesheet record to an employee record in the master employee data file. The rest of the properties specifically relate to the TimeSheet Entity itself.

  - **Related Functionality:**
    As mentionned in the problem description, the TimeSheet entity will need to perform several types of actions. The names of the methods described here should help state clearly what the methods which helps make the object definition that much clearer. Asu such, here are these methods: GetTimeSheetData, SaveTimeSheetData, LoadTimeSheetData, PrintTimeSheetData, WeeklDataCompleted, PrintWeeklyTimeSheet, CalculateRegularHours, CalculateOverTimeHours, CalculateHolidayHours, CalculateRegularAmount, CalculateHolidayAmount

- **The ReportingSystem Entity:**
  The ReportingSystem is present because on an entity based problem solving approach, every method needs to find it's place withing an object model. In most cases, printing related functionality is very often isolated into a seperate entity and sometimes even an independant application (so that it can be executed on a seperate system on the network and print the report while users can continue to do their other activities uninterrupted by the printing process).

    - **Defining Attributes:**
      Since the ReportingSystem entity creates no data files, all it needs is three attributes to perform it's task. These attributes are the EmployeeID, a StartDate and an EndDate properties so that it can accumulate all the TimeSheetRecords that fall between these two dates.

    - **Related Functionality:**
      Ultimately, we could have provided all printing functionality in the ReportingSystem entity, which means that the printing of the Employee Data could have also been added as a related functionality. For the sake of this example, I isolated them for the sake of keeping the objects as isolated and independant as possible. As such, this engine only has one method. PrintReport which will print either a weekly, a monthly or a yearly report on the timesheet data for either an employee in particular or all the employees in the data file.

As you can see, the Object Oriented Approach to problem solving is quite different from the Modular Approach. With all this information you now have on the two methods, you might be wondering if there are certain projects, or certain parts of a big project that could benefit from the Modular Approach and likewise for the Object Oriented Approach. Is there situations where you would be better off using one method over another. In this next section, I will discuss this subject, based on my own personal experiences with both methods.

WHEN ONE METHOD SHOULD BE USED OVER ANOTHER:

If we take into consideration both methods used in the example above. We can see that both of them managed to define the problem, and

possible organization adequately. In their own domain, we can say that they were both as successful as the other in doing what they are supposed to do. So where and when can and should you use one method over another? My experience has shown me that the best way to answer that question is to go by development domains they each have their own best way of describing their needs. Let's take a few examples industries to see how to select the best method.

- **Any Mathematic Oriented Industries:**
  In these we can include financial institutions, banks, accountant firms, statistical analysis firms and other related domain where the formulas prevail over the method. In these particular cases, chances are that problems will be defined based on the specific calculations that need to be performed. Since in programming, calculations translate to functions in code, the Modular Programming approach to problem solving can usually relate more closely to the description of the problem at hand. Therefore if you are working for such a firm, or you want to create an application for that specific industry, The modular approach would be the best way by which you can present your project to the people that know the industry well in such a way that they will understand you clearly and faster.

- **Most Science Related Industries:**
  In here we can throw any chemical engineering, Physics based Research and Development and any other related industries that depend on specific methods and order of things in order to perform at their best. Again in these industries, formulas are pretty important when describing the problem but usually not as much as when describing the specific order in which events and calculations need to happen. There's nothing quite like Modular Programming approach when describing an order in execution step that is required by the domain you are developing in and must be followed flawlessly. Therefore, people in this industry will tend to talk in groups of related steps needed to accomplish a task.

- **Database Native Industries:**
  In this category Falls alot of different businesses. For example Inventory management businesses, call management and many communication

related industries, essentially, any business who's business is related to data. Database Software has evolved, much like O.O.P. in order to answer the new, growing, more complex database needs of these companies. So today, when you encounter one of these industries in your career, chances are, when they describe a problem, they will be describing the problem based on Tables in a database, relationships between the tables and the role each table plays within the whole database structure. Also, when they give names to their tables, they will more than likely describe them precisely as they would an object, a table will represent an object, it's fields will represent the object properties. Because of this high resemblence to object oriented programming methods, O.O.P. stands ahead of modular program as far as implementing a solution that will make sense to the way people from these specific companies conprehend the way they do business.

- **Document Driven Industries:**
  Lawyer firms, sales force companies, publicity related enterprises all fall in this category. Even though these industries would probably need databases and talk about them like database native industries, To the people of these industries don't rely on a particular table, but more on everything that pertains to a particular client. When these people describe their needs, they will typically talk of a file in which different type of information (usually forms and contracts) need to all be part of the file when they leave. This is true whether they are talking about a physical file or an electronic data retrieval system (such as data replication systems). O.O.P. tends to play this role well. However I can advise that very specific names to the objects will be required. Especially as far as lawyer firms go since to them each form has a name, a number, something that makes the forms and contracts unique in contents and purpose. They typically worked hard to create their standards and procedure and will expect their standards to be followed even in an application.

- **Multimedia And Entertaiment Industries:**
  Everything that pertains to entertainment. For example, Music software designers, Recording studios, game designers, even book publishers, and video production groups. In these industries, database are not the primary concern, they are more concerned with the product they are

creating for their customer. And you'll probably notice that they explained things always based on the type of work they do, who they do it for and what elements they need to do their job right as per the work to be done. All in all, these industries today all talk in a very Object Oriented way because all they work with are physical things or physical representation of different things, especially the gaming industry since by today's standard the gaming industry is all about 3D worlds and 3D object representations. So when they want you to develop a game module for a specific game, chances are, they will describe them to you in terms of an object you will be adding methods to or creating from scratch. Of course object oriented is better suited, however, it's important to know that one of the reasons why O.O.P. was so propagated in these industries isn't because Modular Programming couldn't have done job, but rather a simple choice to go the O.O.P. way when they created DirectX and OpenGL standards and specifications. If OpenGL was created using the modular programming approach instead, today, Modular Programming would have been the gaming standard.

This covers the industry and which method you're more likely to want to employ when developing (and presenting) a solution to the people that work in these respective industries. There is another factor that can be a very big player in your decision to use one method over another. That factor is one of the size of a project and hence it's complexity. There are many cases where even a program, for an entirely O.O.P. oriented industry like Multimedia and Entertainment, just doesn't need O.O.P. or would fail to take advantage of the O.O.P. approach to bring a solution to that particular problem. Likewise, O.O.P. could very well be the only viable solution to a problem brought forth by industries that have never needed O.O.P. before. It really all depends on the way they can explain these problems based on what they know of their industry. One last factor could be that the company you are developing for already have tools and do not wish to purchase tools to accomodate for the missing method. In other words, companies could very well impose their development methods and expect you to be able to arrive at a solution using their method whether it's suited for the task or not. In those case you just need to make the best of what you have.

These represent how I would deal with problems in these specific industries. I'll take the time to see how a concept can best be described to the users of the program I am making and if the description is based on objects, unless there is a restriction that stops me from doing so, I will use

an O.O.P. approach especially when you are creating an entirely new project. When you think about it, there isn't just the program that you are making to consider but also how easy your solution will be for the users to understand and use. This means that you have to take the time, in making your decision, to see what kind of user you are dealing with to see how they might react to one or the other of the explanation methods and based on the findings of that little research, use the method that can be thought the fastest and easiest.

WHERE OBJECT ORIENTED PROGRAMMING FAILED:

When you talk about O.O.P. today, you'll hear some very different explanations by different software professionals. Since O.O.P. is found on the 5 pillars I mentionned about, how can these answers be so diverse and, in some cases, so out of touch with each other? The first answer I have to bring here is that depending on when developers started learning O.O.P. they were given some very different notions on what exactly O.O.P. is and how to efficiently use it for problem solving.

One of these new notions arrived when Java hit the mainstream programming industry. Which suddently told you that everything imaginable in programming is now an object including strings, integer and other datatypes. hence the base of all programming should be the object. The problem with this notion is that it's true only for languages that are designed around the same concept as Java where everything you can create is an object. This does not today constitute the majority of the popular languages you can use today. When you think about it, a datatype has no reason to be an object simply because of the way a compiler organizes memory. An object has an overhead that would represent a major waist of memory compared to storing the classic 4 bytes for integers. All these little things is what makes Java so slow and a perfect example of sloppy bloated code waiting for a faster processor to surface so it can make it look faster.

Another one of the confusions in O.O.P. definitions arrived when Objects Modelling techniques like U.M.L. (Unified Modeling Language) made surface. U.M.L. in itself is not a bad approach at all in problem solving. However, today, alot of what defines U.M.L. is confused, or atleast stated when talking about O.O.P. Often when I talk to people, they will mention such notions as agreggations and composition. The problem with this is that any language that can do O.O.P. doesn't have an aggregation or composition construct per se. But these people say that they do alot of

aggregations and compositions when they develop. I then have to explain to them that these are object modeling principles and have no direct equivalent in the language itself. Aggregation denotes a whole/part relationship between objects. For example a Circle is the whole object and a point, is part of a circle objects this is created as an aggregation of two objects. A Composition can be used to create a relationship between objects too but it has the added ability to have a source and a destination (Circle knows about Point, but Point doesn't know anything about circle). These are Object Modelling principles and cannot be directly coded in a language. but when translating the principle to the selected programming language, it could mean that the module that defines Circle will have an #include "point.h". But it does not mean that because this include statement is in the Circle Module that you have created an aggregation necessarily.

OBJECT ORIENTED MYTHS:

If you have been to forums and message boards that compare and debate modular programming and object oriented programming, you might have noticed that many claim O.O.P. is the only intelligent methods and start enumerating reasons that have absolutely no direct relationship to O.O.P. I've seen these too many times and it's a shame to see most of them are enumerated in an effort to bring down modular program and try to boast O.O.P. as a better approach. I will be listing here some of the more popular myths about O.O.P. that I have seen and tell you what I have experienced in regards to the myths themselves.

- **Object Oriented Programming is closer to how people think.**
  The first thing I can say about this is to take a look at the industries I've mentionned previously in this article and notice how the people working in these industries typically think. Some people are simply better suited to think in terms of modular programming, others seem to think that O.O.P. really does offer a clearer approach that as human beings, they can more naturally follow. But the truth of the matter is that everything depends alot on the working experiences of the individuals. Each method, if you were to do a survey, would probably be close to equal in results.

- **Object Oriented Programming makes debugging quicker and easier.**

I'm sure this isn't the first time you've heard of this myth. The previous myth is largely responsible for the fact that people believe that debugging O.O.P. based programs is easier and quicker to locate and fix a bug. The truth of the matter is, when a bug does occur it will usually do so either in one of the methods of and object which in most cases is equivalent to a procedure or a function. If you have implemented an elaborate hierarchy of objects, making proper use of polymorphism you might find yourself looking through more than one procedure called Draw and that could go double if Abstraction was used as well In essence, debugging is not about a programming method per se. It is rather about what you return to the user in your error management routines that can help locate a bug quickly or not. Once you have located the bug fixing it in either methods is equally easy and fast.

- **Object Oriented Programming gives you a better organization of your code.**
  This to me is debatable. Atleast I wouldn't phrase it this way. I would choose to say: "O.O.P. forces you to abide by a given standard" which results in all your objects being defined and implemented the same way. Differente developers visualize and comprehend programming concepts in very different ways. Some can instinctively break a process into a workable set of procedures and functions but just can't relate to the Object Oriented Approach simply because they skip that step of definition to go directly to what actions need to be performed. Modular Programming allows you to organize your code pretty much anyway you want and when you think about that concept. I could easily organize a user defined type (or more) and a set of procedures that display or manipulate elements of that datatype in such a way that it can be made to look quite close to how I would have done it using the Object Oriented approach.

- **Object Oriented Programming is more powerful than Modular Programming.**
  I really have to draw the line at this one. So far, in all of my 30 years of programming in both methods, I have yet to see O.O.P. be able to do anything that can't be done in a procedural language. In my opinion, this belief is because new comers to programming take a look at what's out there and see things like OpenGL, OpenAL, DirectX, Allegro and all

"popular" engines for making the "hot" games and think to themselves: "Wow, O.O.P. can do all that? Amazing!" The problem is that the newcomers will usually stop right there. They won't even think for a second that Modular Programming can do the same because they saw the O.O.P. version first. To that I say, just look at the METHOD APPROACH COMPARISON I did earlier in this article. Both methods defined the problem and could bring a solution to that problem. If you took the time to do both of these methods on any programming projects you have, you'll find that for every O.O.P. problem solving definition, there is an equal (sometimes shorter even) Modular Approach that can do the same job. This pretty much sets this myth rest on it's own.

I do have to say however that there are some myths that I found to be true however, again based on my own experience. And when you think about the reasons I give here, you'll find it hard to see it differently. This told me that O.O.P. can have it's specific place. Here are the two most widespread myths that have proven themselves to me as being true.

- **Object Oriented Programming is ideal for simulation projects:** As you know, a simulation is a program that reproduces the some parts of the real world. As such, it makes perfect sense to represent the real world using an object model. In a simulation you are also likely to see relationships between objects as well as interaction among these objects. O.O.P. is definitaly adapted for this kind of software development and can more closely mimic the real world object it is trying to simulate. This is true for both outside objects reacting and interacting with each other, or to explain and define all the inner components that make up a bigger more complex object. There's no denying that O.O.P. can be wisely used here for all these reasons. Note that Modular programming could do the job as well. But when you think about it, it would be hard to use the modular approach to explain objects, their properties and their methods as well as their relationships to other objects in a procedural way. It could be done, but in this particular case, it would be longer to do when you'd get to explaining the relationship between objects or between components of an object. This is in the analysis only however, the modular code that could do what the object oriented code could wouldn't be much different in size and could actually have a very similar coding organization in both cases.

- **Object Oriented Programming is ideal for Business knowledge development:**
  This one today is very true, but O.O.P. itself is not the main or only reason why it is so true today. As I mentionned in the database related industries, database systems have evolved to answer the growing needs of database administrators and businesses worldwide. As such, an Ojbect model and a database model are usually very closely related. So related in fact that it would actually be longer to seperated the model and break them down modularly to explain their functionalities and roles. As such, This is a perfect scenario where O.O.P. really has a distinct advantage over Modular Programming. Sure you could devise a set of procedures and functions and have them perform the designated task, however, in businesses, it is all about relationships between business objects.

THE BOTTOM LINE:

The bottom line is quite simple. It would not be wise to decide that everything can be made from one of these methods and stick to it alone for all your programming endeavors. In some cases, O.O.P. will naturally be able to represent a problem much more efficiently where in other cases, a modular approach could save you loads of time by being able to represent the problem much quicker, easier and cleaner than any O.O.P. model ever could. it's important to remember that Modular Programming and Object Oriented Programming are two problem solving methods and that both are designed to bring answers to questions and solutions to problems. They are two different means of implementing a solution and that is really all that needs to be remembered.

It's important that you remember that this article really reflect my own experiences with both problem solving methods and my own personal opinions about them. Others probably have a different point of view and may have all the right reasons to believe them. All I can say is so far, in my 30 years of personal and professional development, my judgement on these has not failed me once. It takes good common sense and a bit of logic to make the right decisions and opinions about them. I will finish by saying that it's not wise to destroy a problem solving method over another, they truely do have their advantages and could make your life easier if you didn't reject either of them. It is even worse to believe what others say without knowledge. This means that if you want to form your opinions, don't read comments from anyone and believe them blindly. Give Modular Programming and Object Oriented Programming atleast a fair chance,

find a project for each and see where each method takes you. It won't be bad for your career to know when each methods has advantages so it should be time well spent on your part. Learn them, evaluate them, compare them THEN when you have the knowledge you need, you can make a good choice.

As you now know, with all my writings, I am always opened to discussion and comments, suggestions and even debates. You can email me whenever you want to clear things that weren't clear to you when you read this. This article is the result of my own experience with the different programming methods. Your past experiences might give you a similar or different point of view on this subject. I'd like to hear of them because I know that everyone has his or her own thoughts on this subject, I see it every day on forums I visit. Until the next time I write, happy reading and programming no matter which method you like to use.

MystikShadows
Stéphane Richard
srichard@adaworld.com