Teaching Object-Oriented Programming Concepts Using Visual Basic .NET

Albert Dieter Ritzhaupt
Ron James Zucker
Department of Computer and Information Sciences
University of North Florida
Jacksonville, FL 32246
rita0001@unf.edu rzucker@unf.edu

ABSTRACT

This paper presents an object-oriented approach to Visual Basic .NET instruction to be delivered in a traditional academic semester for information system curricula. The paper first discusses some of the inherent problems with Visual Basic .NET instruction and then proposes an object-oriented approach. This approach includes a systematic set of programming projects to take students on a journey that traces the principles of the object-oriented, the event-driven, and the procedural paradigms into a coherent framework. The Unified Modeling Language Class Diagram notation is used to model an object-oriented system that is developed and enhanced throughout the duration of the course. Practical recommendations and programming exercises are provided and evaluated in the discussion. This course is intended to be at minimum a second programming course for information system students to satisfy IS 2002 guidelines.

Keywords: Object-oriented programming, information system curricula, Visual Basic .NET

1. INTRODUCTION

For many years now information system faculty at both the Associate of Science and Bachelor level has made the decision to adopt the Visual Basic programming language into the curriculum. This decision ranges in reasons from its ease in the development of windows applications to its forgiving development environment – Visual Studio. Some institutions have even selected Visual Basic as the first programming language for students to learn in the curriculum. The Visual Basic programming language, however, has made a fundamental change with the debut of Visual Basic .NET from the previous versions – 6.0 and less. This presents some interesting problems for information system educators. Visual Basic .NET (VB.NET) is now fully object-oriented rather than event-driven.

1.1 Problems in VB.NET Instruction

"Microsoft's .NET not only represents a major shift in the approach taken to software development and deployment, it also has the potential to change what is done in the college classroom" (Murray, 2003). The difficult process of integrating .NET into the college classroom has taken off in the past few years (Haiduk, 2002; Murray, 2003; Chaytor and Soleda, 2003). Some of the problems encountered are outlined in the subsequent discussion. Figure 1 capture the essence of these problems by showing the vast skills a VB.NET student must develop to proficiently utilize the language.

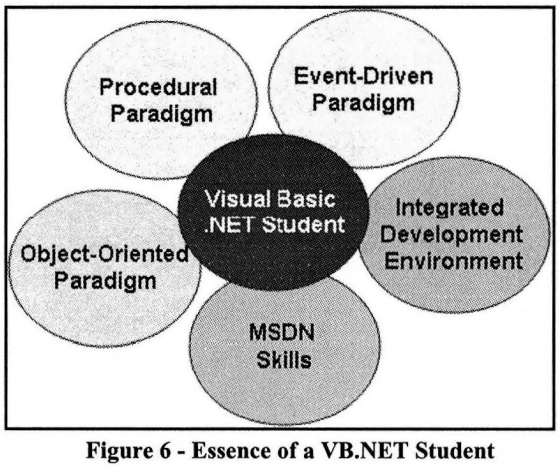


Figure 6 - Essence of a VB.NET Student

1) The .NET framework is a comprehensive application programmer interface providing services to many languages supported by the .NET platform. The services available are documented in the Microsoft Developers Network (Microsoft, 2005). At first glance, the MSDN can overwhelm the introductory student. However, the services provided by the framework are critical to the development of robust and

reusable applications. The incorporation of the MSDN is therefore critical to instruction in VB.NET. Knowing which packages to include in the course can be a challenge for faculty that has not been properly exposed to the MSDN and a further challenge for students with limited software development experience.

2) Visual Basic texts are having trouble outgrowing their roots. The language originated as a non-object-oriented language, to which object facilities were added over time. VB.NET is a re-creation of the language, fully object-oriented, which incorporates the same syntax as Visual Basic as a subset. Yet many textbooks still seem to use the approach developed within earlier versions of the language: a combination of event-driven and procedural programming. In Bradley and Millspaugh's "Programming in Visual Basic .NET," 4th edition, object-oriented programming is given a cursory glance in the first chapter and covered separately in chapter 6 (Bradley, Millspaugh, 2003). Koneman's "Visual Basic .NET Programming for Business," covers many of the advanced topics, such as Collections (chapter 8), without first addressing object-oriented programming (chapter 9) (Koneman, 2003). Object-oriented programming is treated little and later in the texts as a separate thought rather than the cohesive glue that binds all the concepts together.

3) The fundamental principles of program design are based on a few similar ideas, yet the approach to developing a solution differs depending on the programming paradigm. Three programming paradigms have emerged as the superior approaches to program development: procedural, event-driven, and object-oriented. Each paradigm exhibits unique characteristics that are important enough to differentiate the environments as three different worlds. The approach discussed in this section embraces an object-oriented software development perspective, while emphasizing the critical points of the other paradigms. As if teaching three different programming paradigms is not difficult enough, there is also a significant challenge associated with teaching active procedural and event-driven programmers the object-oriented paradigm because it requires a paradigm shift from long lasting procedural and event-driven principles and habits that are not acceptable practice in the object-oriented world (Turk, 1997).

4) Visual Studio .NET is the Integrated Development Environment (IDE) that can be used to develop and deploy VB.NET applications. This comprehensive utility provides many services that introductory students will not fully understand without substantial practice, such as the wide-ranging properties found for each control, the debugger, the form editor, et cetera. Learning Visual Studio .NET is a skill on its own because it utilizes many advanced technologies. The students will have to learn the IDE with some degree of proficiency to be able to develop VB.NET applications (Chaytor and Soleda, 2003). Students might easily get lost in the IDE and confuse the tool with the language.

Undoubtedly, these are some of the many problems faculty will or have encountered in the instruction of VB.NET. Consequently, this paper argues that the VB.NET programming language should be at minimum the second programming language learned by information systems students for the following reasons: (1) the language encompasses aspects of the procedural, object-oriented, and event-driven paradigms; (2) the development environment, Visual Studio, provides many advanced technologies outside the scope of a traditional introduction to programming course; and (3) the language and environment can unintentionally promote poor software development practices if students do not understand good software design principles (e.g., option strict off). Although this paper does not provide evidence to support this argument, we believe educators are aware of these problems and we believe that educators should consider the drawbacks of such a widespread technology as the first programming experience students encounter.

Thus, it is imperative that information system programs take a proactive stance in reviewing their curricula and making plans to align them with the object-oriented paradigm if the Visual Basic .NET programming language is to be taught. We believe that VB.NET instruction without a strong emphasis in object-orientation is a disservice to both the student and programming language.

1.2 Curriculum

This course was offered twice as an introductory programming course in Visual Basic .NET at the community college level with substantial success reported by the students. An advanced course is also offered that focuses on the .NET framework and its advanced component technologies, such as the ADO.NET. One of the many challenges facing community college instructors is the diversity of students entering the information systems courses, which include a combination of associate of science students in an information systems area, associate of art students transferring to a university, and practitioners retooling for the workforce. The curriculum for this approach is rigorous and quite adequate for an object-oriented programming course offered at the university level.

Students enrolling in the course are expected to be proficient in programming concepts and the procedural paradigm. This course is intended to build a strong foundation in object-orientation so that advanced topics can be more easily addressed in the subsequent course. Unlike similar approaches, this course tackles object-orientation by addressing the perspective earlier in the curriculum to build a foundation for advanced study (Chaytor and Soleda, 2003). Our perspective is that a student well-seasoned in object-orientation will more easily understand advanced topics in the .NET framework.

The course also fits quite nicely within the 2002 Information System Undergraduate Curricula Guidelines (IS, 2002). IS 2002.5 – Programming, Data, File and Object Structures should include the "traditional and visual development environments that incorporate event-driven, object-oriented design" (Gorgone et al., 2002). This course covers many of the topics in the scope and discussion of this course requirement.

2. APPROACH

This section describes the approach and system description for the course. A Unified Modeling Class Diagram is presented to capture the static relationships of the system to be developed. A figure is provided to show where th elearning objectives are covered in the programming projects while tracing through the three programming paradigms.

2.1 System Description

Figure 2 shows the complete software system to be developed in the duration of the semester as a UML class diagram. Each programming exercise adds more complexity to the overall software system in a piecemeal fashion where the concepts to be covered in the course are aligned with the new components added to the system.

The object-oriented system proposed here is a simplified investment application that computes future values for three different types of accounts. These accounts hold the attributes necessary to make the appropriate calculation and are derived from the generic account class: the interest, the principle or payment, and the term. The interface allows the user to create, update, and maintain many customers, each with their own accounts. The application can capture the state of the customers by use of serialization to the Simple Object Access Protocol (SOAP) as a file on secondary memory. The application also generates a report of the total value of each of the customers, the total value of all the customers, the total number of customers, and the average value of each customer.

2.2 Instructional Environment

The instructional environment for this approach utilizes blended learning. Traditional lecture and class exercises are supplemented with online discussion to enhance ideas and clarify points of interest via Blackboard, a course management system. All student questions related to the course material are directed to the discussion board to spur dialog and to streamline similar questions and answers. The discussion board has proven to be very helpful in that it triggers fruitful design and implementation dialog among the students and forces them to communicate their technical ideas in natural language. The use of the discussion board is restricted, however. Students are given clear instructions not to use source code from their programming projects but to provide similar examples. Students are allowed to discuss the design and implementation issues of their program – not provide a solution to programming problems via source code.

During lecture and class exercises, a parallel example is presented using a simple payroll application with three different compensation types: Salary, Hourly, and Salary plus Bonus. The application closely resembles the one being developed in the programming projects. First a lecture is provided to cover the ideas and answer questions, and then the class exercise is completed with the students practicing in the environment. Design discussion during classroom lecture is promoted by the instructor. The problems encountered in the class exercises map to the problems encountered in the programming exercises. The parallel example also includes the development of a Unified Modeling Language Class Diagram to capture the static relationships of the design.

2.3 Unified Modeling Language Class Diagrams

The Unified Modeling Language (UML) is the result of many years of software modeling experience. UML is a standard suite of many diagrams and is currently it its second version (Object Management Group, 2005). Class diagrams are one of the more popular conceptual models because they are rich with content, which attests to the easy transition from design to implementation. It is assumed the reader has a basic understanding of UML class diagrams.

Class diagrams clearly present composition, aggregation, generalization, and association between classes. Class diagrams also depict cardinality, multiplicity, abstraction, encapsulation, and specialization as well as the access modifiers of the attributes and operations. Programming projects require the delivery of a UML class diagram to develop the student's skills in software modeling and design.

2.4 Three-Tier Model

One of the major goals of software development is maintainability. Maintenance is historically an expensive cost to businesses; therefore techniques have been developed to support ease of maintenance. The three-tier model is one of these techniques. Each tier can be understood to be highly cohesive yet loosely coupled with the other tiers. Figure 2 shows an example of the three-tier model and the responsibilities assigned to each of the tiers. Consequently, modifications to the system will not necessarily require changes to each of the tiers. For example, a change in the appearance of the presentation of the system would not necessarily require a change to the business or data tiers and vice versa.

This model is embraced throughout the development of the system to discuss design considerations as well as implementation details in relation to the maintainability and

	Business Tier	Data Tier
Controls: Forms, buttons, textboxes, menus, etc. Data manipulation Data Entry and Validation	Business logic and validation Data rules Business classes	Primarily Database Data access components Stored procedures

Figure 2 - Three Tier Model

extensibility of the system. Students learn how to develop a system where the user interface is loosely coupled to the business tier – the customer and accounts. This model should be used as a frame of reference throughout the design and implementation phases of the system development life-cycle.

2.5 Programming Exercises

There are five phases to the programming project, each using principles of the three programming paradigms. Figure 3 shows the relationship between the programming paradigms and the programming projects as learning objectives. The course was offered in a 15-week semester, the amount of time placed on each of the projects is shown in the table as well. This section will briefly outline the specifications for each of the projects.

2.5.1 Programming Project 1: The first phase requires students to create an interface for a Future Value calculator with the functionality attached to the form. Although it is discussed as being the incorrect implementation for a robust solution, it is necessary to start here to understand why the three-tiered approach is a better implementation. The future value program calculates three different future value types: Compound Interest, Simple Interest, and Payment Compound Interest (an annuity). Students are encouraged to enhance the user interface with different texts, colors, and borders to broaden your understanding of these properties.

At this point, the students can assume that a user will enter valid data. Therefore, the system assumes it is impossible for a user to enter erroneous input. Students are also instructed that the interface should have a picture that relates to the topic. For instance, one might use a picture of dollar signs or money. As shown in Figure 4, many important programming topics are covered here, such as sequential logic, and arithmetic precedence.

2.5.2 Programming Project 2: In the first phase, students created an interface for a Future Value calculator with the functionality attached to the form. This is not the preferred approach because it is necessary to decouple the business logic from the user interface. Students create a class called FutureValue as part of the new solution. The class has three class methods, one for each type of future value calculation, given public access modifiers so that the methods can be used as services.

In this phase, students are required to check for erroneous input. Erroneous input at this point is defined as data that is nonnumeric or less than or equal to zero. If either of these errors occurs, the user should be notified with a message box indicating the type of error and how to fix the error. Students are asked to provide a flow chart for the logic of the data validation routine to achieve the most logically sound solution. Examples are provided in class using subroutines and functions to facilitate the optimal solution. During this phase, students are also instructed to deliver a UML class diagram of the software. It is important to note that there are two classes shown in the diagram, but no association shown between the interface and FutureValue class because the operations are class methods.

2.5.3 Programming Project 3: The focus of this phase is to learn to build a set of problem domain classes with instance variables and functions. Students are instructed to reuse the FutureValue class from the previous phase to make the calculations. Students are then instructed to create an Account class with the following protected instance variables:

- Term – Integer
- Principal – Double (also the payment)
- Interest – Double

The Account class should have accessors (getters) and mutators (setters) for each of the variables specified. The account class should be defined as must inherit, meaning no instantiations of the Account class are permitted – an abstract class. The Account class should have the following functions:

- MustOverride Function: getFutureValue() As Double
- MustOverride Function: getAccountType() As String

Students are then instructed to create three simple classes named: SimpleAccount, CompoundAccount, and PaymentCompoundAccount. These classes must inherit from the Account class emphasizing the concept of generalization and specialization (inheritance). Students are instructed to provide for each of the accounts a:

- Constructor: XXXXAccount(…) - to provide the data to instantiate an object of the specific Account.
- Overrides Function: getFutureValue() As Double- to override the parent class and return the future value (using the proper calculation of future value for the particular class).
- Overrides Function: getAccountType() As String - to return a String representation of the "Specific Account".

This programming exercise is a turning point because students learn two of the most fundamental object-oriented concepts: inheritance and polymorphism. The students are instructed to declare a class scope Account reference in the user interface class. In the new UML class diagram, a composite relationship exists between the user interface and account class because the lifetime of the account is dependent upon the lifetime of the user interface. The specific account class instances are assigned to the generic Account reference (polymorphism, and 'is a').

2.5.4 Programming Project 4: Upon beginning the fourth phase, students are building a full-scale object-oriented software system for both the presentation and business tiers. They are told to think of the Future Value System as an extremely simplified banking application.

The specifications are provided here. A bank has a collection of customers, and each of these customers has an array of accounts. The collection suggested is an ArrayList. Each of these accounts yields a different type of future value based on the type of account created. This phase uses the account classes created in the previous project. A class will have the following protected properties:

	Object-oriented paradigm	Event-driven paradigm	Procedural paradigm
Programming Exercise 1 (2-weeks)	❑ Primitive data types ❑ Abstract data types: Controls and forms	❑ User-triggered events ❑ Basic user interface ❑ Design ❑ Tab Orders ❑ Hot Keys ❑ Basic Controls	❑ Basic arithmetic ❑ Sequential logic ❑ Data types, declaration, initialization
Programming Exercise 2 (3-weeks)	❑ Class methods (Shared Functions) ❑ Class Creation	❑ User-input validation ❑ Intermediate controls ❑ Message boxes	❑ Functions ❑ Functional decomposition ❑ Decision logic using if-then-else
Programming Exercise 3 (3-weeks)	❑ UML class diagrams ❑ Inheritance ❑ Polymorphism ❑ Instance variables/ functions ❑ Method overriding	❑ Separation of presentation and business logic tiers	❑ Decision logic using case structures ❑ pass by reference versus copy
Programming Exercise 4 (4-weeeks)	❑ Abstract Data types: Collections Composition ❑ UML class diagrams	❑ Advanced user interface controls ❑ Advanced form validation ❑ Managing multiple data instances	❑ Introduce array processing ❑ Repetition logic using for and while loops
Programming Exercise 5 (3-weeeks)	❑ Serialization ❑ Sorting Using Collections ❑ UML class diagrams ❑ Multiple Inheritance	❑ Printing ❑ Separation of presentation, business logic, and data tiers ❑ File Dialogs	❑ Basic report generation ❑ Sorting

Figure 3 - Learning Objectives and Programming Paradigms

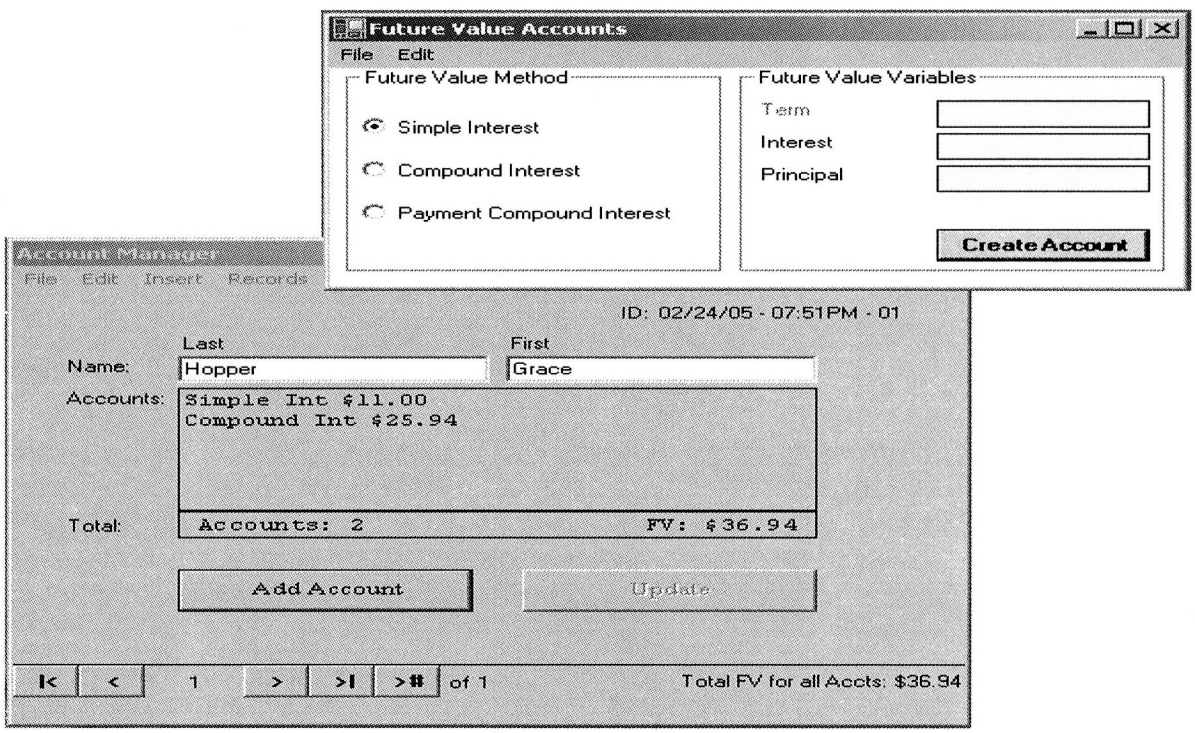


Figure 4 - Example of System User Interface

- Customer Customer Identification – String
- Customer Last Name – String
- Customer Accounts – Account[]
- Number of Account – Integer
- Shared Number of Clients – Integer
- Shared Total Investments – Double

The customer can have a maximum of five accounts. The customer class should have accessors and mutators for the customer name and identification number. The Customer should have the following operations:

- Constructor: Customer(Customer First Name, Customer Last Name) - to provide the data to stantiate an object of the Customer class.
- addSimpleAccount(interest, principal) As Boolean – instantiates a SimpleAccount object to add to the Customer, returns a Boolean indicating whether it was successful.
- addCompoundAccount(interest, principal, term) As Boolean – instantiates a CompoundAccount object to add to the Customer, returns a Boolean indicating whether it was successful.
- addCompoundPaymentAccount(interest, payment, term) – instantiates a PaymentCompoundAccount object to add to the Customer, returns a Boolean indicating whether it was successful.
- getNumberAccounts() As Integer – the number of Accounts the customer owns.
- getTotalFutureValue() As Double – the total future value of all of the customer's accounts
- Shareable: getTotalInvestmentValue() As Double – the total future value for all the customers and all their accounts.
- Shareable: getNumberCustomers() As Integer – the number of customers.

Students are told the user interface should be designed to accommodate all of the functionality of the system, which includes managing customers and accounts. The system is designed for banking employees, so the employees will need to be able to create new customers. The customer identification is the current date and time appended with a hyphen to the current total number of customers created. This is a variable created within the constructor of the Customer class. For example, assume the current date and time is 07/04/04 – 11:22AM. The customer identification then is 07/04/04 – 11:22AM – 01, assuming this is the first customer created. The user interface should prompt the banking employee for the appropriate information to create a customer.

The interface should also allow the banking employee to traverse through the different customers so that they can view or update their information. The only field that should be editable is the customer name. The interface should also show the number of customers currently in the system.

The system also enables the banking employee to view a summary of the customers account information (total number of accounts, and the total future value). Accounts cannot be changed or removed. There should be an option to add an account. However, if the number of accounts is already at the maximum of five, a message box should appear indicating that no other accounts can be added. When iterating through the different customers, the program should reflect the future values and number of accounts for each customer. The interface should also show the total future value for all the customers in the system. An example is shown in Figure 5.

This phase is where the student should realize the object-orientation as the underlying theme. Students come to understand how class methods and data work together with instance methods and data. The concept of composition is expanded by emphasizing the relationship between the user interface and Customer, and the Customer and Account ('has a'). This programming project also emphasizes arrays, Collections, and repetition logic. The total value of a specific customer instance is a derived variable from the array of Accounts. It is suggested to students in this phase to reuse the user interface from the previous phase to capture the account information for a customer – multiple forms.

2.5.5 Programming Project 5: This phase will require making four modifications to the previous phase to provide functionality for the final tier in the multi-tier model: sorting capabilities, report generation and printing, file save capabilities using Simple Object Access Protocol serialization, and search capabilities. The third tier is treated lighter than the other two because the intent of the next course is to elaborate on the ADO .NET.

The Customer class implements the IComparable interface and provides the implementation of the required compareTo function (Microsoft, 2005). The Customer class sorts based on last name and first name, if necessary ignoring the case. The user should have the option to sort the customers at any given instance of time.

The software system also allows the user to save the current listing of customers at any given point in time. The system should also enable the user to load the current listing saved. Therefore, SOAP serialization is used to save the ArrayList and all of its contents out to a file. The FileDialogs are used to save and load the files to specific locations. A caveat for this feature is that private and shared (class) data will not serialize. Therefore a mechanism for reloading the collection into primary memory is necessary. This can easily be accomplished by traversing the list and recreating each of the objects to be inserted into a new list.

The software also generates reports and prints the reports. Students are instructed to use the PrintDocument and PrintPreviewDialog to accomplish this. It is required that the report be sorted by last name. The report reflects the total value of each of the customers, the total value of all the customers, the total number of customers, and the average value of each customer. Finally, students build searching capabilities into the form. The user should be able to search for a Customer by their last name. Students are instructed to accommodate for multiple instances of the same name.

3. CLOSING REMARKS

This paper has documented some of the problems surrounding VB.NET instruction in information system curricula, which includes the textbooks not outgrowing their origins in the earlier versions of the language. The VB.NET student must develop skills in many areas: procedural, event-driven, and object-oriented paradigms; MSDN navigation and use skills; and a proficiency in the IDE – Visual Studio.NET.

The approach presented in this paper is intended to be at minimum a second programming course for information system students. The approach embraces an object-first methodology to lead to advanced topics in software development. We believe that a strong foundation in object-orientation is necessary to for students to be able to design robust and maintainable source code in VB.NET.

This course was offered twice at the community college level. The course retained 21 students out of 27 enrolled (78%) in the first semester and 17 students out of 24 enrolled (70%) the second semester. Overall, the retention rates for an intermediate programming class, we felt, are reasonable for the demands of such a rigorous curriculum.

Student response to this approach has been positive. Feedback from students includes beliefs that the course provided a strong foundation in object-oriented programming for future study and use in industry, good exposure and use of the MSDN, and that the UML class diagrams help them understand object-oriented programming better. "The projects assigned to us were excellent - gave me a full sense of object oriented programming (Annamalai, S., interview, October 21, 2005)."

One student mentioned that the approach to presenting the material was critical to the effectiveness of the course. "I found the content valuable, but it may not have been so had it not been for the presentation of the material (Martin, G., interview, October 21, 2005)." The only negative feedback received from the course is that the workload is too intensive for an intermediate level programming course. "For a person with no prior experience in object-oriented programming, I would say it was a really challenging course, but I learned a lot (Ornelas, Y., interview, October 24, 2005)."

We feel that the positive feedback compensates for the negative, and IS faculty should consider adopting such an approach in their curriculum.

4. ACKNOWLEDGEMENTS

We are grateful to Florida Community College at Jacksonville (FCCJ) and the University of North Florida for their support in offering this course, and to Derrol Andre Poole, an instructor at FCCJ for his insights and constructive criticism in the writing of this article.

5. REFERENCES

Bradley, Julia C. and Millspaugh, Anita C. (2003), "Programming in Visual Basic .NET," Fourth Edition, McGraw Hill, 0072938706.

Chaytor, Louise and Soleda Leung, 2003, "How to Creatively Communicate Microsoft.NET Technologies in the IT Curriculum." Proceeding of the 4th conference on Information Technology Education, pp. 168-173.

Haiduk, H. Paul. (2003), "Object-Oriented Classic Data Structures for CS 2 in Visual Basic .NET," Journal Computing of Sciences in Colleges, Vol. 18, No.1, October 2002, pp. 185 - 198.

Gorgone, John T., Davis, B. D., Valacich, J. S., Topi, H. Feinstein, D. L., Longenecker. (2002), "IS 2002: Guidelines for Undergraduate Degree Programs in Information Systems," Association of Information Systems.

Koneman, Philip A., "Visual Basic.Net Programming for Business," Prentice Hall, 2003, 0-13-047368-5.

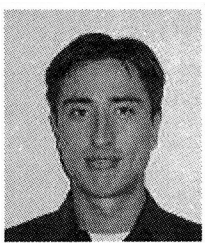
Microsoft, Microsoft Developers Network Retrieved On February 24, 2005 from: http://msdn.microsoft.com/library/.

Murray, M. (2003), "Move to Component Base Architectures: Introducing Microsoft's .NET Platform into the College Classroom," Journal of Computing Sciences in Colleges, Vol. 19, No. 3, January 2004, pp. 301 – 310.

Object Management Group, Unified Modeling Language 2.0, Retrieved on February 24, 2005 from: http://www.uml.com.

Turk, Michael. (1997), "Introducing Object-Orientation to Experienced Procedural Programmers," Proceedings of the 2nd Australasian conference on Computer science education, Vol. 2, pp. 135 - 140.

AUTHOR BIOGRAPHIES



Albert D. Ritzhaupt is an adjunct instructor at the University of North Florida. He has a B.S. in Computer and Information Sciences and an M.B.A. from the University of North Florida. He is currently a research assistant conducting research in instructional courseware development for information systems curriculum and is completing his Ph.D. at the University of South Florida. Albert has taught in the areas of UNIX and Linux, data processing mathematics, Internet programming, Visual Basic .NET, microcomputer applications, Java programming, and COBOL Programming.

Ron J. Zucker is an instructor at the University of North Florida. He has over 35 years of information technology experience including over twenty years in industry and seventeen years teaching at the university level. He has a Master's Degree in Computer and Information Science from Troy State University in Montgomery and is currently a PhD Candidate in Computer Science and Engineering at the University of South Florida. His current research is in Human Computer Interaction.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096