Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 1998 Proceedings

Americas Conference on Information Systems (AMCIS)

December 1998

# Tools to Facilitate Event-Driven Program Design in Introductory Courses

Alka Harriger
*Purdue University*

S. Lisack
*Purdue University*

# Tools to Facilitate Event-Driven Program Design in Introductory Courses

**Alka R. Harriger**
**Susan Lisack**
Computer (Information Systems) Technology Department
Purdue University

## Abstract

*Widespread acceptance of the Windows environment has increased the popularity of application development tools that facilitate creation of Windows programs. In response, many universities are starting to teach introductory programming courses using these new software development tools. However, the event-driven nature of these new tools requires a design change to the traditional methods of teaching introductory programming. Unfortunately, most programming textbooks that employ the new tools neglect to provide a suitable framework for designing programs for this new event-driven software paradigm. This paper will present the key differences between event-driven and conventional programming, particularly as it affects teaching programming development concepts to beginning students. It will also describe how a new design tool (Object-Event Diagram) can be used to promote student understanding of event-driven programs.*

## Introduction

With the widespread acceptance of the Windows environment, event-driven programming has become more commonplace for today's new software development projects. Products like Microsoft's Visual Basic and Borland's Delphi provide an integrated program development environment that make it easy to create elaborate user interfaces for a Windows program, write event procedures, and test and debug programs. Visual Basic exceeded all initial expectations and has grown to be a very popular product within the software development industry. Its increasing popularity has created a need for application developers who can program with objects and use event-driven programming techniques. Therefore an increasing number of schools are starting to teach introductory programming courses using this new software development tool.

The textbook industry has responded with a number of Visual Basic programming textbooks that show how to use the tool's programming environment to both create the user interface and to write program code through event procedures. While these books do a good job of teaching Visual Basic coding using a wide variety of controls provided by Visual Basic, they do not totally address the program design issue of relating the code modules to their controls and events.

Programs created by Visual Basic and other similar products, are not only event-driven, but also very object-based. By re-using predefined objects available within the tool, the programmer is able to quickly create a customized interface that is both attractive as well as functional. In planning for the distinct processing needs of the user, the programmer must identify the most appropriate triggers that are associated with specific screen objects on the interface. By writing code for only these predefined events associated with select screen objects, the programmer is able to transfer the majority of the control of the program's operation to the user and/or system. Because the majority of the input and output functions have been separated from the processing functions, the traditional software development methodology no longer applies to the event-driven environment. Nonetheless, many of the traditional methods can be adapted for the new event-driven environment.

## Event-Driven vs. Traditional

There are notable differences between event-driven programming and traditional programming. In traditional programming, the program is in control of the processing sequence that takes place. If the program needs input from the user, a question is asked. The program then performs the necessary calculations and sends the results to the appropriate output device. Once an event-driven program is loaded into memory, the user (or system) controls the processing that will take place. Each screen object or control on the form can have zero, one or many events that can be triggered. Most of the events that trigger program activity are a result of an action taken by the user or a change to the system (such as the ongoing tick of the clock). Examples of user actions include moving the mouse, pressing the left or right mouse button, clicking the mouse while its cursor is over a particular control on the form and typing characters in a text box. The user can choose to perform these actions in various orders. While it is true that the program can limit the user's choices by disabling certain controls or options, or even making them invisible, the program should only use this to indicate options that are inappropriate, and not to control the user.

Once an event is triggered, the code that comprises the event must be processed in its entirety before control returns back to the form (so the user can initiate the next event). Sometimes, the code within an event can actually trigger another event resulting in cascading events that must be completed before control reverts back to the form. From this perspective, an event-driven program is a collection of multiple event procedures, each of which are directly related to specific controls on a form, and any of which could trigger additional events.

The nature of the event concept forces the structure of an event-driven program to be more modular than the traditional program has to be. Each event that the program responds to has its own event handling procedure. These event modules can still call on other program modules to accomplish their task. However, the initial functional decomposition in an event-driven environment is based on the events for which the programmer has written code.

## Event-Driven Program Development

Regardless of whether a conventional programming language or event-driven programming language is used, most educators will agree that a well-defined, systematic process should be employed for application development. The recommended program development process for the new event-driven environment is depicted in Figure 1.

STEP 1: Analyze the Problem
(Identify IPO: Input, Processing, Output)
STEP 2: Design the User Interface
(Add controls for each IPO need)
STEP 3: Design the Logic
(Object-Event Diagram (OED) identifies controls and events they trigger)
(Flowcharts or pseudocode depict detailed logic for all events and modules)
STEP 4: Translate Logic into Code
STEP 5: Test and Debug
STEP 6: Package the Documentation
STEP 7: Maintain the Program
(start over with Step 1)

**Figure 1. Event-Driven Program Development**

Since the early days of programming, introductory programming was taught using a top-down, functional decomposition approach that employed various program design tools (Bohl and Rynn 1998; Topping with Gibbons 1985). Hierarchy charts or structure charts were used to represent the functional decomposition of the problem. The detailed logic for each module was depicted using tools such as flowcharts or pseudocode. In the case of event-driven programs, functional decomposition is strongly influenced by the events related to the controls on the user interface.

A confusing concept for most beginning students is the fact that any event can be triggered in any sequence via a user or system action. The action is directly related to a screen object. Having a tool to visualize these events and their relationships to various screen objects will enable students to better understand event-driven programming.

## Event-Driven Program Design Tools

There are a few textbooks that employ design tools to provide a framework for programming in an event-driven environment. Zak (1998) introduces a TOE (Task, Object, Event) chart as a tool that is used during problem analysis to document in a tabular fashion a list of major tasks, the objects associated with those tasks, and the events where the tasks should be processed. Once the TOE chart has been completed, the programmer is instructed to create a rough design of the user interface. Unfortunately, because tasks are organized chronologically in the order that they would be most likely processed, the TOE chart forces a structured approach on an event-driven environment.

Schneider (1998) employs the hierarchy chart as a tool to document the various high-level tasks and their associated subtasks in a manner much like functional decomposition for traditional program development. Unfortunately, the tool does not relate any of this processing to specific objects. Furthermore, like Zak's TOE chart, it forces a structured approach on an event-driven environment.

Bradley and Millspaugh (1998) use a two-column table to list the events and help plan the actions associated with these events. The first column lists procedures and the second column lists actions. The procedures column describes events, while the actions column includes pseudocode for the associated event. Again, this is fine for an initial planning tool, but it is limited in its ability to show the relationship of the events to the components and the overall program structure.

The remaining texts minimally use similar tools, but do not address the need of the beginning student to understand the role of events and their relationships with screen objects. In all of these cases, there is some merit to showing the need for an initial planning and design tool. Unfortunately, all of the preceding approaches appear to force a traditional, structured approach on the new event-driven environment.

## An Object-Event Diagram

In an event-driven environment, the user controls the events that are processed by initiating appropriate triggers such as clicking a button, pressing a key, and moving the mouse. Because the events may be triggered in any order, not at all, or multiple times, an event-driven program is a collection of several mini-programs that act upon the same user interface. Recognizing the relationships between the program's events and the interface's screen objects is critical to understanding how event-driven programs work. A design tool for this environment should therefore show not just the decomposition of each event, but also the relationships between the events and their associated screen objects.

The traditional hierarchy chart is not applicable to an event-driven environment because it only shows the names and relationships of the modules in a program. Also, the initial decomposition is determined by the major functional tasks of the program. In contrast, the nature of event-driven programs pre-defines the initial decomposition through the program's controls and associated events.

The authors have developed a tool called the Object-Event Diagram (OED) that helps students visualize the relationships of screen objects to their events. It looks like a hierarchy chart, but it is based on the controls and events of the user interface.

(Lisack and Harriger 1998) Once the interface has been created, designing the OED becomes a natural transition from the interface design stage. From a user's perspective, everything starts from the form. Thus, the single node at the top of the OED represents the form. At the second level are nodes for form-related events, as well as for each control on the form. The third level contains nodes for events associated with the controls of level two. The functional decomposition approach of traditional hierarchy charts can then be applied to any events that are too complex. The modules resulting from this decomposition are added to the fourth levels of the OED and beyond, as needed, to achieve manageable module sizes. Figure 2 illustrates how to construct the OED for today's event-driven, visual programming environments.

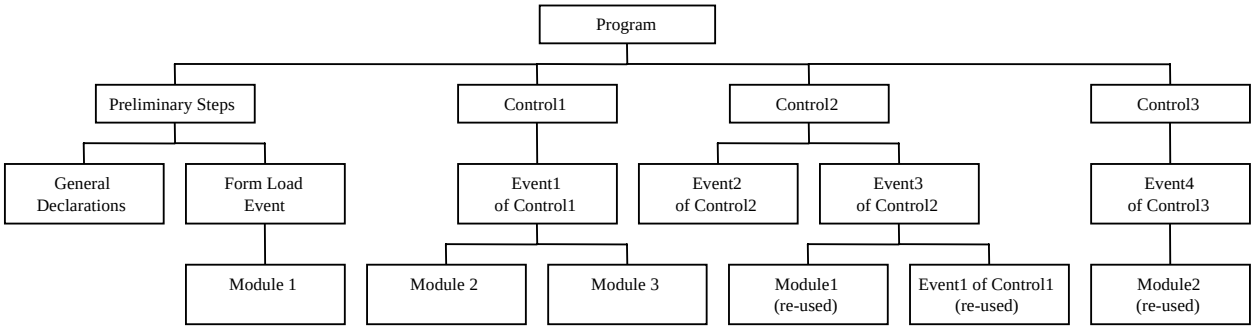


**Figure 2. Object Event Diagram**

For programs that employ multiple forms, each form would have its own OED, and an additional OED would be created to document preliminary steps (global processing) and project forms. This OED would serve as a virtual table of contents for the separate OEDs that document each of the separate forms.

## Conclusions

Developing programs for an event-driven environment is very different from the process used for a traditional environment. Available resources still force a structured approach in a manner that serves to limit the beginning student's understanding of the nature of event-driven programming. The use of the OED can help the student visualize that the form is the main focus of the event-driven program, as well as recognize that the user can control which events are processed by triggering the desired event through an appropriate action. This visualization should abet the student's understanding of event-driven programs. Future projects may test the feasibility of using Object-Oriented tools such as state diagrams or activity diagrams to teach program design in introductory programming courses.

### *References*


Bohl, M. and Rynn, M. *Tools for Structured Design: An Introduction to Programming Logic,* Prentice-Hall Inc., Upper Saddle River, New Jersey, 1998.

Bradley, J.C., and Millspaugh, A.C. *Programming in Visual Basic 5.0,* Irwin McGraw-Hill, Boston, MA, 1998.

Burrows, W. and Langford, J. *Programming Business Applications with Visual Basic 5.0,* Irwin McGraw-Hill, Boston, MA, 1998.

Lisack, S. and Harriger, A. "A New Use for an Old Program Design Tool," in *Proceedings of the Informing Sciences Mini-Track of the World Multiconference on Systematics, Cybernetics and Informatics,* Orlando, Florida, in-press.

Schneider, D. *An Introduction to Programming Using Visual Basic 5.0, Third Edition,* Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1998.

Shelly, G.B., Cashman, T.J., and Repede, J.F. *Microsoft Visual Basic 5 Complete Concepts and Techniques,* Course Technology, Cambridge, MA, 1998.

Topping, A. with Gibbons, I. *Programming Logic: Structured Design,* Science Research Associates, Inc., 1985, pp. 128-140.

Zak, D. Programming with Microsoft Visual Basic 5.0 for Windows, Course Technology, Cambridge, MA, 1998.