

## PART 5 APPLICATION DEVELOPMENT

### CHAPTER 13 PROJECT START TO FINISH

#### IN THIS CHAPTER, YOU WILL LEARN HOW TO:

- Explain and implement program maintenance using the program development lifecycle
- Differentiate between creating a new program and maintaining an existing program.
- Explain and demonstrate the modularization of program code to promote code reusability and simplified program maintenance.
- Describe and differentiate various program design approaches (traditional structured, object oriented and graphical event driven).
- Describe implementation alternatives for new and updated programs
- Distinguish and differentiate between traditional desk top and Internet application development.
- Describe and differentiate various program design approaches (traditional structured , object oriented and graphical event driven).
- Summarize and implement key logic design and program code concepts presented in this wiki.

## THE PROGRAMMER DEFINED

This final chapter will act as a capstone for all the preceded it. In the previous twelve chapters, we have covered a variety of topics related to writing programs. In each chapter, we incrementally took a concept and broke it down into bite size pieces for the new programmer. It is now time to take those pieces and put them together and begin to think like a programmer by performing a task done by most novice programmers. The approach that we will take in this final chapter is not to design a project from scratch but instead to take an existing program and modify it. We will focus on a project that requires program maintenance.

**Program Maintenance** - Program maintenance refers to programming code changes made to existing programs. Long after a new program is designed it is modified, repaired and improved upon. Because programming is a labor intensive activity, most companies will try to preserve their investment in software by extending the life of a program through program maintenance.

There is a good reason why we will start this chapter by maintaining an existing program. The best reason is that most new programmers are assigned maintenance programming tasks before they have the opportunity to design new systems.

Since new programmers are still learning the tools of the trade and I've yet to fully understand and experience the tools of their craft, working with an existing application is a good place to train the new programmer. By seeing another programmer's work, the new programmer gets the added benefit of having a template to follow. They will be able to see logic patterns already implemented in code. This will strengthen the new programmer's knowledge of programming, debugging skills, recognition of programming patterns along with providing the programmer the confidence to take on more complicated programming projects.

Like any good apprenticeship, time spent under the tutelage of a master makes the apprentice's education that more complete. Looking at it another way, by assigning the new programmer maintenance projects, the programmer only has to concentrate on parts of the entire program. Taking on only a small part of the program will make it easier for that new programmer to get the confidence necessary to become successful. The programmer will also be able to see the part of the program they are maintaining working together with its other parts.

**Background Information:** Master - Apprentice - As a new programmer, finding a master or a good mentor is important in your programming education. The advice and experience of a veteran programmer can greatly improve the learning experience. For one, a master programmer will be a good person to go to when

things are not going so well. Maybe a day with a few rookie mistakes has you feeling a little insignificant. Maybe it feels like you are the only one who has ever made a mistake in their program. Especially a mistake that seemed so stupid. An experienced programmer will assure you that you are not the first programmer to make a simple mistake and certainly not the last. They will also tell you that mistakes are okay in programming but you should work hard not to make the same mistake twice. Making a mistake is a learning opportunity and you should always look at it in that light.

## THE SECRET: IT IS IN THE PROGRAMMING PROCESS

This might be a good time to attempt to answer the question, "What is the secret to becoming a successful programmer?" Ask different programmers the answer to that question and you might get some different answers. But if you ask each the importance of following a consistent program development process, they will tell you that consistency is what makes a programmer successful. Most systems are developed using the System Development Lifecycle (SDLC) or some derivative of it like the Program Development Process (PDP). The SDLC process is general enough and at a high enough level so it can be used for everything from programming to installing network hardware. It is a process that is organized, logical and most importantly, repeatable. All of these components are critical if the process will be successful.

I have suggested in this wiki that programmers follow a process which is derived from SDLC. The program development process moves from planning to fact finding, to conceptual design, to coding, to test, to launch, and to program maintenance. As we moved from traditional to object orientated programming and then finally to event driven programming with the graphical interface, we have always returned to the program development process.

It is important that we stay with this programming development process and that we have this process in the back of our minds as we develop software. Even when the program is not being created from scratch as a new program, the program development process remains important in program maintenance. As we move through the program maintenance project for Cactus Books and Tapes, we will stop

and reflect on the process as we make changes to a desktop program used by the Receiving Department to account for book shipments.

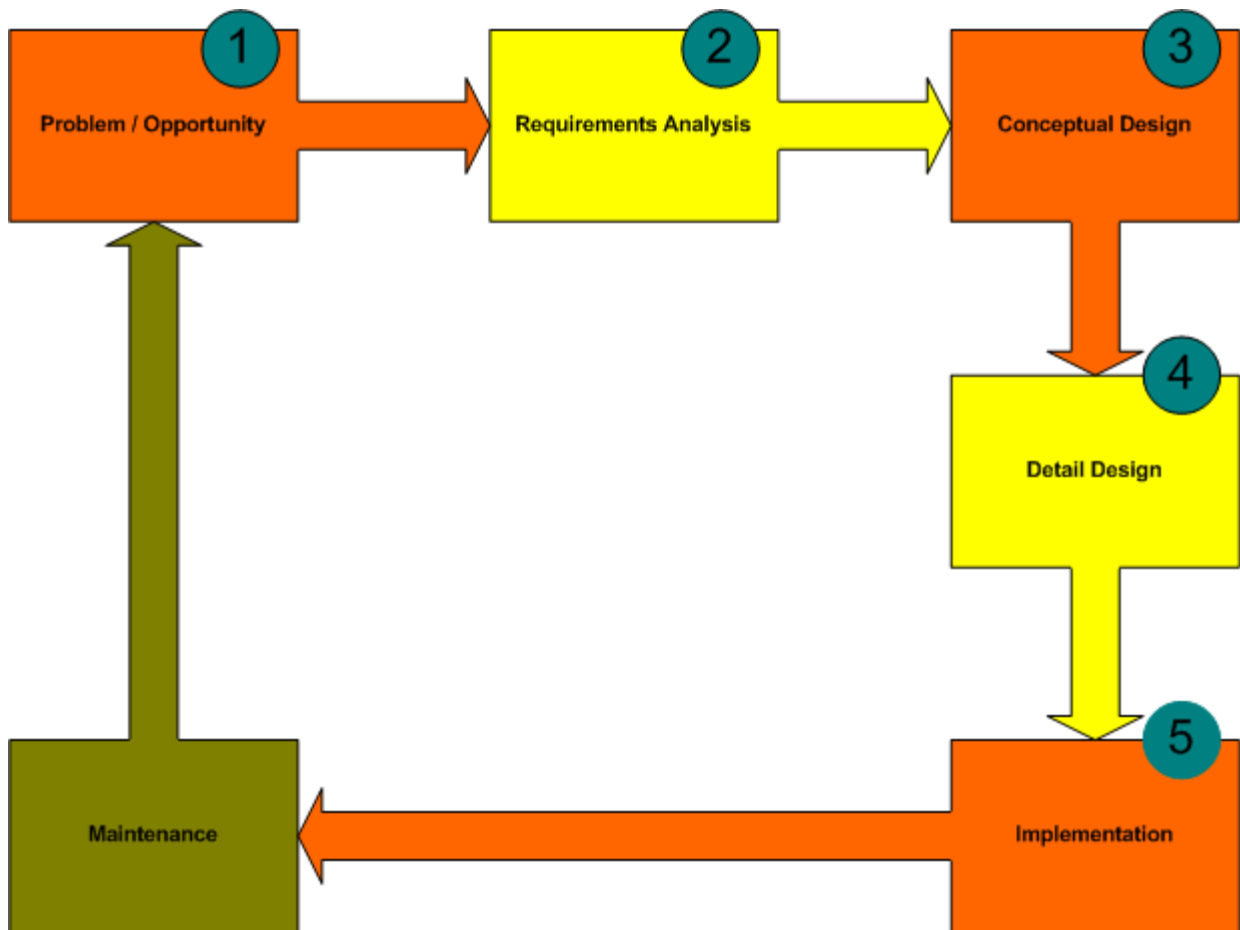
## A PROJECT WALKTHROUGH FOR CACTUS BOOKS AND TAPES

The Receiving Department at Cactus Books and Tapes consists of three employees. Alice has been an employee at the bookstore since its very beginning. Even though Alice does not have manager as part of her title, she has been the team leader of the Receiving Department for the last two years. Bob and Cindy work as Receiving Associates and all share members of the Department have the responsibility of receiving new books into store inventories. The receiving activity includes comparing packing slips against the contents of each package and then recording the books by ISBN number into the stores inventories system. The system in place has been effective and continues to accurately and efficiently log new inventory items into the system. With the increase in business, the receiving department has consulted with a storage space planner and it has been determined that they are running out of inventory storage space and need to better organize their work area. As part of the space planner's recommendations, they have suggested that the mini tower personal computers be replaced by smaller units fastened to the underside of the work table. Above the worktable would be a flat screen touch sensitive display. The keyboards currently take up too much table space. The only problem is that the existing software is text based and runs under the MS-DOS operating system. It is an event driven system that uses a text screen and has no graphical interface with controls to activate program logic. You have met with a salesperson that has a flat screen monitor and requires no special programming other than the ability to use graphical controls to control the program.

For this assignment, you and the other programming intern have been asked to work together to make the necessary program changes.

## THE PROBLEM

The problem could be defined as the existing applications inability to be controlled with a graphical user interface. Fortunately, the program logic is sound and all that needs to be changed is the creation of a graphical interface.



- 1** Create problem statement defining the Receiving System Update. Review business and technical rationale.
- 2** Fact finding activities: Interview users, DBA, original programmer and review existing system. Use data to create logic models.
- 3** Review existing code and create pseudo code and storyboard logic models. Update models that reflect the changes. Create storyboards illustrating the new graphical interface.
- 4** Create PYTHON code from logic models and storyboards. Create text matrix and unit test all business rule logic.
- 5** Perform program and system testing. Update documentation and build on-line help. Work with technical writers to create user documentation. Parallel implementation of program. After 1 month compare new systems reports to existing system to verify new system is working correctly.

Figure 1: Our Plan for Updating the Receiving Programs

## FACT FINDING

Since we have focused on a problem, we now need to do the research and fact finding necessary to understand what needs to be done to solve this problem. The first piece of information that we need to understand is what the existing system looks like. Since only the text interface will need to be changed, screen shots of the existing system will be required to understand what controls will be necessary and what information they will need a capture. We have set up an interview with Alice and have told her in advance that we are interested in getting screen shots of their existing receiving screens. Since we cannot count on Alice to understand the process of creating screen shots from a Microsoft Windows PC, we will visit her in the Receiving Department to look over the system and capture the screens. This is also a good time to meet with Bob and Cindy and begin to establish a rapport with them. They will be invaluable in our fact finding and it is very possible that we will need to continue to work with all three of the employees and the receiving department as we refine our logic and storyboards. We want to get to know them early and gain their trust. If we include the staff in this project, we will promote buy in to the solution that we create. This relationship could come in handy as we perform testing and as we implement the new programs when the maintenance is complete.

We have also set up an interview with Fred who is the Cactus Books and Tapes database administrator and Fran who is also in the programming department. As database administrator, Fred will be of great importance in helping us with setting up program security and perhaps creating test tables for us to use while we test of program changes. We have also asked Fran to sit in on our discussion with Fred because Fran was the original programmer who developed the receiving system. As mentioned earlier in the wiki, as a programmer you will not work as an island and will need and want to include others. Input and advise from other technical staff and employees will only improve the solution.

## FINDINGS

The interviews with bookstore staff went very well and both programmer and user left with a feeling that all understood what was needed to convert the existing application to one that was driven by graphical controls. It was determined with Fred that no database file maintenance changes would be necessary since the program updates really amounted to just the screen interface and the addition to the processing of graphical control events. File maintenance which performs adds, updates and deletes would remain the same.

The interview with Fran was more revealing. Whereas the initial problem had stated that the software in place today was driven by a text interface using event driven processing, it in fact was a traditional program that used nested if statements, while loops). The Receiving system processed valid orders and exception orders. The definition of an exception order was described as an order that was shipped differently than what was documented on the original purchase order. This could be caused from back ordered items or when the quantity of items was less than what was initially ordered. The receiving screen would need to be able to process a valid order but also process an exception. This information constituted a major business rule in the receiving process. In addition, the receiving staff also had a print option available to them which would allow them to print the contents of the screen. This came in handy when there was an interruption and they had to leave the receiving area for short period of time. The new program should also have some provision made for printing out information on the selected order.

Whenever there is fact finding or program testing there will be an opportunity for reflection and further fine tuning of program requirements or program design. These changes should be coordinated with the end user in care should be made not to let slight program modifications to become major changes that change the scope of the project.

**Key Concepts:** The requirements analysis will allow you to discover and isolate the organizations business rules.

**Related Subject:** Scope creep! - A situation that all programmers must be aware of is called scope creep. Scope creep is a term used to define a scenario when an initial project with a set of defined deliverables starts to grow larger and larger with new requirements. This happens as users and programmers find new features to add or problems to solve. Adding to the requirements will add to the complexity and length of time necessary to complete the project. Scope creep has taken many a successful project and derailed it into being late and over budget. It's important that when a modification is requested that all agree it's a necessary modification and part of the original problem. It should be communicated to both management users and other programming staff that program has changed and that deadlines and costs may also change with it. Typically, any changes should be approved (i.e. management and user review) just as new projects are approved to make sure scope creep doesn't side track a good project making it a failure.

**Scope Creep** - The term used to describe a programming project that has additional requirements appended to the original project after the project has started. Scope creep can cause a project to take too much time a go over budget.

## CONCEPTUAL DESIGN

Since the solution requires a graphical user interface, we know that we will have to create both a storyboard and pseudo code that will explain how we will create the interface and build logic. The storyboard will probably be the most important logic model since the fundamental receiving logic is not changing. In fact, we will focus our logic changes on in making sure that events capture input and display output. Since the initial programs made extensive use of class files and concepts of code reuse, we will black box or ignore some of the details concerning the initial logic. We will also assume it is correct since the exiting program has no known problems. Fortunately, this basic receiving logic is not an issue and the fact that it is implemented within class files will allow us to limit its focus in our logic models.

**Black box:** Black box is a term used to describe something which is unknown or undefined but part of a solution. In a program, a part of the programs requirement may yet to be coded but the function is known. A black box acts as a placeholder until that piece of the project can be completed. It can also be used to describe a

piece of reusable code (like a class file) that is part of the program but of which the details are hidden or unknown to the programmer.

We will need to focus on the screen design. We need to select graphical controls which are the most appropriate for the touch screens. An additional requirement over and above normal screen design would be the concern that the interface be ergonomically sound for busy workers who must now use their hands to target controls on a touch screen. The controls must be big enough and visible so that they can work as quickly on the screen as they worked with the keyboard. It is important remember that the old system was text based. The need for a pointing device like a mouse was not something that they had to contend with when receiving products. We have to make sure that the touch sensitive screen, which is really a pointing device, doesn't impact their productivity. The receiving dock is a busy place in the need to quickly get product on the shelves or to customers on special order is critical.

## STORY BOARD

Much of the maintenance required for this program revolves around the introduction of a graphical user interface. Therefore, the storyboard will be critical in ensuring that the user communicates to you a design that will be effective and efficient for receiving products to the store.

I have created storyboards for the receiving screen and the exceptions screen. For these storyboards, I made use of a commercial application popular with programmers for diagramming flowcharts and building storyboards called Microsoft Visio. Although creating professionally detailed graphical designs is not a requirement, a realistic storyboard does let the user know exactly how the screen will look when it is completed.

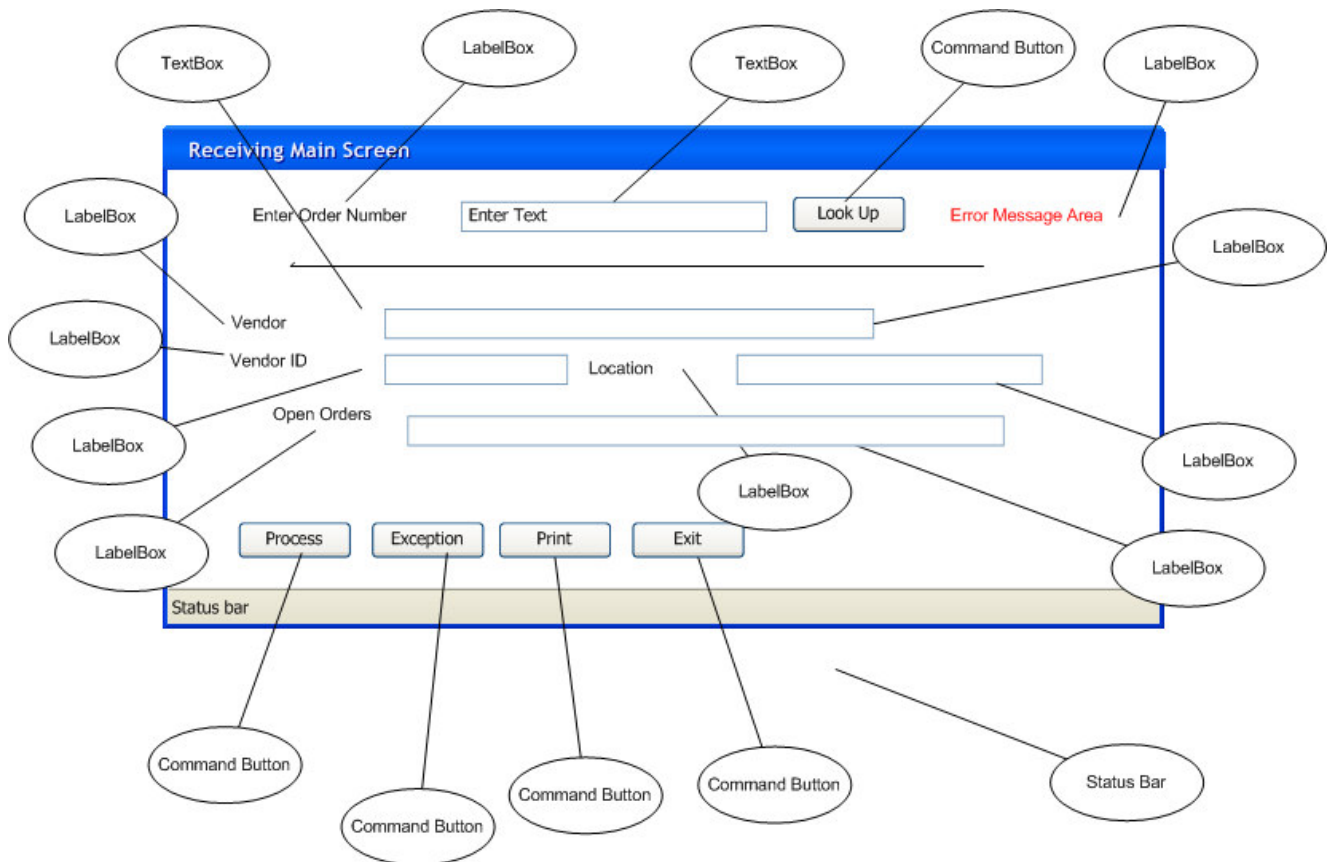


Figure 2: Storyboard Main Receiving Screen

## EXISTING PSEUDO CODE LOGIC

After the session with Fran the programmer of the original receiving screen, we're interested in looking at the actual program logic the currently exists. The original pseudo code was not available so we will have to take the existing PYTHON program code and convert it into pseudo code. Once we have a model the logic for the initial application, we can then model the logic for our modified receiving application.

At this point, we are not concerned with the receiving class which holds receiving data and methods. Since the business rules will remain the same for this new

system, we can assume the class logic will still hold true. We are more interested in the logic which prompts receiving staff for order numbers and then allows the processing of received merchandise into the inventories system. This entire source code listing of the original system is available later the chapter but what I have documented here is the pseudo code of the main logic.

**Business Rule:** A business rule represents a process used by the business to make decisions (i.e. No merchandise return can be made without a sales receipt). Some business rules are common across businesses and some are unique to that business.

## MAIN LOGIC DOCUMENTED IN PSEUDO CODE

The following pseudo code was extracted from the original receiving program. Many times the original pseudo code is not available so you may have to reverse engineer logic from the original program. You will also want to interview users and the programmers who have worked on the logic to get additional information or confirmation that you are on the right track.

**Reverse engineer:** Reverse engineering is the process of taking a program and duplicating either its look and feel, logic or both. The design and coding of the program would be done from scratch without the benefit of viewing the original programs code. The new program would contain original copy but still be a copy in regards to function.

It is also possible that the programmer used their pseudo code inside the program as program comments. If this is the case extracting just the program comments into a new text file might be useful in identifying pseudo code.

```
start
myOrder = ReceivingLogic()
myOrder.setOrderNumber("0")
print "                               Receiving Main Screen"
while (myOrder.getOrderNumber() != "Q"):
    print
    myOrder.setOrderNumber (raw_input("Enter Order Number or Q to
```

```

Quit >> ")
    display = myOrder.lookupOrder()
    if (display not equal "Order Number not Found" and
myOrder.setOrderNumber      not equal "Q") :
        print "ORDER NO. |  VENDOR
|  VEN. ID                    |  QTY  |  AMOUNT |"
        print display
        flag = 0
        while (flag not equal 1)
            dummyDisplay equals raw_input("Enter P to Process
or E to                               process Exception
or Q to Quit  >>  ")
            if (dummyDisplay.upper() not equal "Q") then
                if (dummyDisplay.upper() equals "P")then
                    print "Order Processed"
                    flag equal 1
                else if (dummyDisplay.upper() equals
"E")then
                    print "goto Exception Menu"
                    flag equals 1
                else
                    print "invalid entry"
            else
                flag equals 1
                myOrder.setOrderNumber("Q")
        else
            if myOrder.getOrderNumber() not equal "Q" then the
                print display
end

```

Analysis of this program confirms what Fran had told as an earlier meeting. The existing receiving program was not build on an event driven processing model. The code pattern used is similar to event driven programming but different enough that the new pseudo code will need to have logic created to accommodate event processing. The initial program relied heavily on decision structures (if statements) to guide the user through the receiving program logic.

### Program Updates Pseudo Code Logic

```

start
class MainScreen:
    recvLogic = ReceivingLogic()

    orderNumLabel as Label
    orderNumber as textbox

```

```

calc as Button
errorMessageLabel as Label
  vendorScrLabel as Label
  vendorLabel as Label
  vendorIDScrLabel as Label
  vendorIDLabel sa Label
  locationScrLabel as Label
  locationLabel as Label
  openOrdersLabel as Label
  openOrdersDataLabel as Label

Process as Button
Except as Button
PrintOrder as Button
Exit as Button

lookUpOrder(event)

  orderNum = orderNumber.get()
  recvLogic.setOrderNumber(orderNum)
  returnedOrder.set(recvLogic.lookUpOrder() )

callProcessOrder(event)
  MessageBox ("Window Message", "Order Processed!")
  returnedOrder.set("")
  orderNumber.focus_set()
  orderNumber.delete(0,99)

callExcept(event)
  tkMessageBox.showinfo("Window Message", "Display Exception
Screen")

def callPrintOrder(self, event)
  MessageBox ("Window Message", recvLogic.lookUpOrder())

callExit(event)
  MessageBox ("Window Message", "Exit Program")
end

```

## ANALYSIS OF RECEIVING CLASS

One of the common elements in both the existing program and in the new graphical receiving program is the receiving class file. This receiving class file includes all of the data and methods necessary to enforce the business rules and data receiving

requirements at Cactus Books and Tapes. Even though this class file remains unchanged in the new version of the program, it is still important for us to understand how its logic and how it will be used in conjunction with the graphical interface. Below are exhibits of the PYTHON programming code and the pseudo-code associated with the receiving class.

**Related Subject:** What about Proto-typing? - One option that we have with this project is to utilize the expertise of the receiving a staff and prototype the application with them. This would mean the receiving staff would have taken more active role in the development of the application. If the application was a new one that represented a radical departure from the way they did this in the past this option might be a good one. But with prototype, the costs are always higher. Receiving staff is paid to do a job and receiving and the more time we take them away from the receiving dock the less work gets done. We also run the risk of over designing this application. It is possible to over analyze logic and since we know that the receiving process is not changing, spending time reengineering the process would ultimately lead to scope creep. Given this extra effort and the fact that the application is being modeled from an existing program, we have decided if forgo the expense or proto typing and stick with a more traditional approach of including the users in the requirements and certainly welcome their help during the testing process. Hopefully when all is said and done, we will have a good program design.

## PSEUDO CODE OF RECEIVING CLASS

As with the main code example documented earlier, the receiving class pseudo code was created by analyzing PYTHON programming statements. Our pseudo code will follow a standard class design that has instance variables defined first and methods an event code defined next.

```
start
class ReceivingLogic()
    ReceivingLogic()
        qty equals 0
        cost equals 0
        orderNumber equals "0"
        Vendor equals ""
        VendorID equals ""
```

```

        orderNumber equals 0
setQty(sinVar)
    if (inVar less than 0 ) then
        qty equals inVar
    else
        print "Invalid Quantity"

getQty( )
    return qty

setCost(inVar)
    if (inVar less than 0 )then
        cost equals inVar
    else
        print "Invalid Cost"
getCost()
    return cost

setOrderNumber(inVar)
    if (inVar not equal "" ) then
        orderNumber equals inVar
    else
        print "Invalid Order Number"

getOrderNumber()
    return orderNumber

setVendor(inVar)
    if (inVar not equal "" ) then
        vendor equals inVar
    else
        print "Invalid Vendor Name"

getVendor()
    return vendor

setVendorID(inVar)
    if (inVar not equal "" ) then
        vendorID equals inVar
    else
        print "Invalid VendorID"

getVendorID()
    return vendorID

lookUpOrder()
    searchVar equals 0
    cnt equals 0
    foundValue equals "Order Number not Found"

```

O

```
        for searchVar in self.ordersArray[]
            if searchVar[0].upper() ==
self.orderNumber.upper() :
                foundValue = str(searchVar[0] ) + "
| " +
| " + str(searchVar[2] )+
| " + str(searchVar[3] )+" | " + str(searchVar[4]
                cnt equals cnt plus 1
                return foundValue

end
```

Most languages have a function that supports the conversion of text from lower case to upper case. Pseudo code does not have an established standard for a function like upper so it is acceptable to insert into pseudo code program statements as long as the reader will be able to understand the statement.

**Background Information** - Pseudo Code Revisited - There will always be a hesitation to develop pseudo code or flowcharts before coding. After you have programmed for a period of time, you will get the urge to as we all do to get busy developing the application. As you become a more experienced programmer this may on occasion work out just fine. Many of the newer programming languages, in their sophisticated IDE's, would allow you to shortcut on some of the logic and storyboarding. However, you should resist the temptation since as the program logic becomes more complicated it will become too difficult to code on the fly. Even a hand sketched flowchart with the assistance of notes taken in the form of pseudo code will greatly improve the possibilities of you understanding the problem in creating a program that actually solves it.

Whenever you shortcut the programming development process your increase the chances of something going wrong. Let me put it another way. Let's say you have decided to paint your kitchen. The appropriate planning would suggest that you remove everything from the shelves and put drop cloths down so of the paint spills it will not spill on any of the items in the kitchen. But it is the weekend and you have other things to do. If you have painted before this isn't something which is complicated and it's certainly something that you've done enough times to attempt by just painting and not worry about getting paint on items in the kitchen (you know that this will be the time when the paint DOES spill). Approaching anything without a

plan and process is a random event and therefore the results are not predictable. Use pseudo code, even if informally, whenever putting together a program.

**Related Subject:** I'm stuck... what do I do now? - Sooner or later you will get stuck on a program. You will have a program that either will not compile or have a logic error that does not seem to want to show itself. Many times you will assume that some part of the program is wrong because it is new. If it new it must be wrong right? Wrong, many times the real problem is with code that you have used before. You have just got it stuck in your head that the problem is one thing and you have biased yourself in considering anything else. In any event, sometimes for those more difficult problems, you are better off soliciting another set of eyes so to help find the problem. I have had several students complain about how they have been stuck on a program for 2 hours only to have me find the error in less than two minutes (and this has also happened to me also). If you can, have someone else take a look at your code. Sometimes even walking them through your program code will jar something loose and the solution or problem becomes apparent. A fresh set of eyes with a fresh outlook can be of great help.

## CODING

Once the storyboard and pseudo code has been created, the coding of the program becomes relatively straightforward. One of the concerns that the programmer must have is that the logical design represents a perfect world implementation. A perfect world solution is just that, a solution for perfect world. Since the programming language will have limitations, you may find that what the user and the designer felt was the best design, might not be one that can be implemented easily with the logic the storyboard supplied. For example, maybe the button design is new and not one that can be found within existing set of graphical controls. A search of third party controls yields nothing that approaches what the designer specified as a command button. In this case you have no alternative but to look at other options.

In regards to programs pseudo code, different programming languages have advantages over other programming languages in regards to certain programming patterns. For example, one language may provide a both pre and post test repetition

processing whereas another programming language may only provide pre test repetition processing. A logic design that can't be implemented in the programming language must have changes made. The programmer needs to stick to logic model specifications as closely as they can but to complete the solution may have to deviate from the original spec. This is part of being a programmer.

For this project, PYTHON is the language that the original system was developed with and the solution that we will need to implement our graphical interface. Fortunately, we covered graphical interface programming in the previous chapter (remarkable coincidence isn't it?). You'll be asked to also create the PYTHON code for the graphical interface in this chapters case study.

## EXISTING PROGRAM

The existing program currently makes extensive use of the ReceivingClass module. As a result, one of the first of statements in the program is to import receiving logic from the receiving class. Once the class file is imported it becomes available to be called from within the PYTHON program.

This original version of the program took advantage of a text interface that relied on a fairly extensive set of nested if statements to control the logic into processing or creating exceptions to products processed at the receiving dock.

Due to the importance of the interactive environment, and the current programs inability to process graphical controls and event driven design, if statements had to be used to determine the execution of program logic based on keystrokes.

As a new programmer, the existing program may not be completely understood without executing the program and testing the interface. As you get more experience, you'll be able to look at program language code in visualize the logic as

it executes on the computer. Until then, it would be wise to copy this program and execute with the debugger or trace through the statements, to better understand how it works. For this wiki, this PYTHON example along with the others covered in the wiki are available with a companion CD. Load this PYTHON program and execute it so that you can better understand what the original program looked like.

## NEW PROGRAM

I have broken the complete program up into sections and inserted a number of bullets within the new program code to describe the new logic. The source code has line numbers to tie the program statements together even when I have inserted text to help explain that section of the program. I also have included a screen shot of the final graphical interface to help visualize what the user is going to see when they run this program.

The screenshot shows a graphical user interface titled "Receiving Main Screen". At the top, there is a blue header bar with the title. Below the header, the interface is divided into several sections. The top section contains the text "Enter Order Number" followed by a text input field labeled "Enter Text" and a "Look Up" button. To the right of this section is a red "Error Message Area". A horizontal line separates this section from the next. The middle section contains three rows of input fields: "Vendor" with a single wide field, "Vendor ID" with a smaller field and "Location" with a larger field, and "Open Orders" with a wide field. At the bottom of the main content area, there are four buttons: "Process", "Exception", "Print", and "Exit". At the very bottom, there is a "Status bar" area.

Figure 3: Solution Screen Shot

The original program specifications required that this new receiving system include a graphical interface for the receiving staff. In addition, through the analysis of the logic it has been determined that the ReceivingClass, which represents receiving process business rules, could be reused from the existing receiving applications.

**Related Subject:** Using published code

Many times in this textbook I've spoken of using code that is already been developed in other programs. In the case of PYTHON, the PYTHON web site ([www.python.org](http://www.python.org)) contains references and links to a variety of resources on the Internet where it just such reusable modules exist. The tkMessageBox dialog box module is just such a case. I could have created an output dialog box using the graphical controls that exist within Tkinter. But an output dialog box, because it is so widely used, has already been created by a program author and made available for everyone to use. If you browse the PYTHON web site or use a search engine to locate on PYTHON modules you'll find a wealth of graphical controls or functions that have already been created and can be used freely as part of your solution. This is also true with other programming languages. One search can yield many valuable resources. Just make sure that the source code is freely distributed and not copyrighted. With programming being such a labor intensive activity, "not reinventing the wheel" are truly words to live by.

This next section of the program is by far and away the most lengthy and time consuming to prepare. The main screen class contains all of the graphical controls along with any defined events. If you are developing this screen class with a simple text editor, the programmer can expect multiple iterations in ensuring that this screen will take the same appearance as the storyboard. Lining up controls for a visually appealing GUI is time consuming and a skill all into itself.

This design as with the designed promoted by many programming languages defines and configures graphical controls early within the class and in the order in which they

are presented on the screen. The order of the presentation is controlled by the layout manager. Most graphical interfaces utilize the some form of layout manager. The layout manager dictates where controls are positioned in the window. The default layout manager for most programming languages is the flow layout manager. Flow layout will display controls in the window from left to right until there is no more room. If the control does not fit on one line it will drop to a second line in display underneath the top row of controls.

**Layout Manager:** Many programming languages contain class objects called layout managers that assist the programmer in placing graphical controls on a window. The layout manager is designated before the controls are coded in the program.

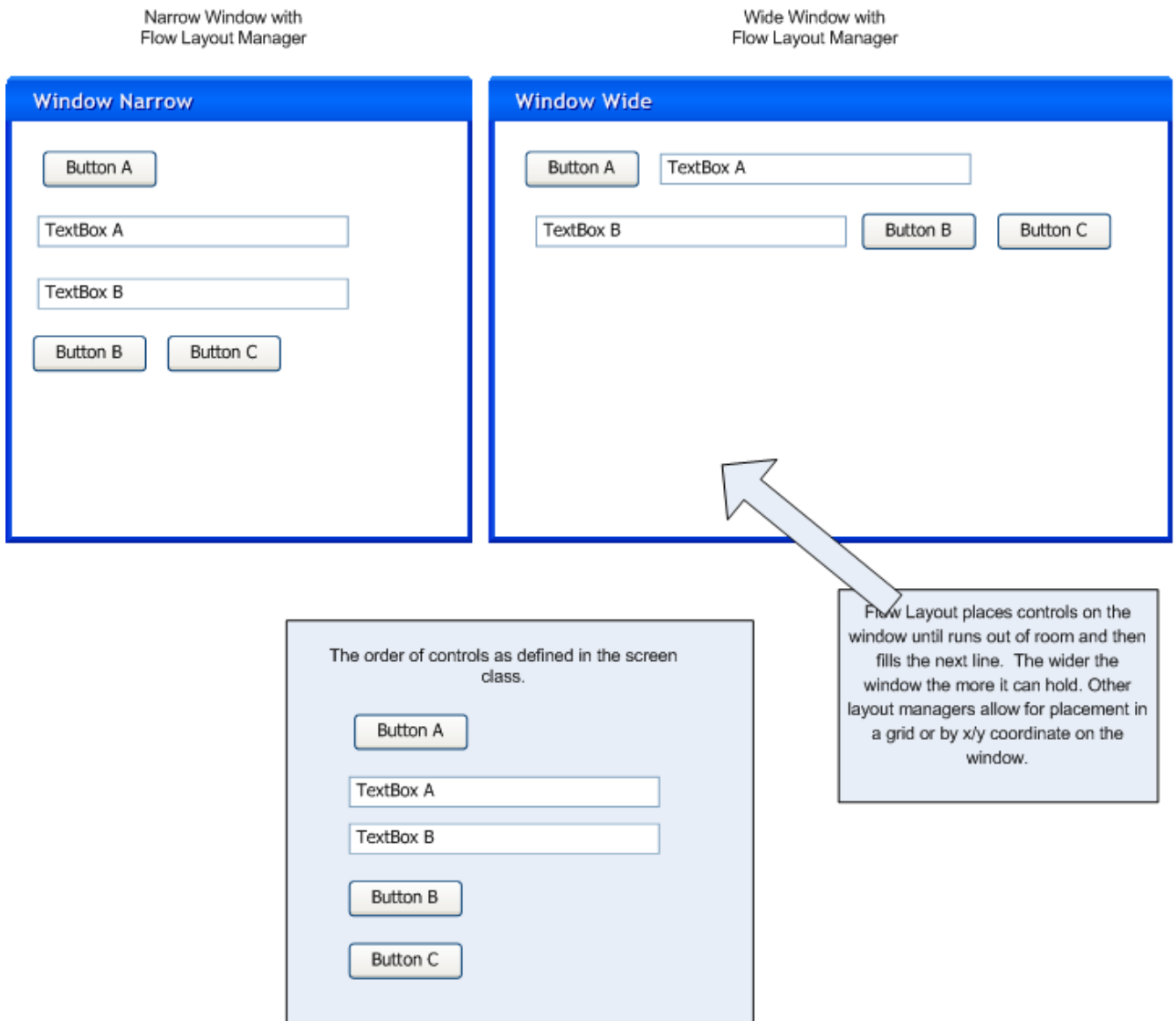


Figure 4: Flow Layout Manager

For this application, I have chosen to use the grid layout manager which allows me to control the surface of the display by designating each controls position by row and column. In previous examples, our PYTHON graphical interface programs typically existed on one line of controls. Since the receiving screens contain far more controls, I have separated the window display into rows and designated horizontal position by a column number. For example, my first row consists of a label box, a textbox (called an entry control in PYTHON), a command button and finally another label box. You'll find this column designation within the grid method available with each

graphical control. In the example below, I am placing the orderNumLabel in the first row, first column of my display. Remember, the value of zero is often used as the first element of an array or collection. The grid is addressed with locations similar to the subscripts of a multidimensional array. Row 0, column 0 would be in the top left corner of the display. Look at the story board or screen shot to locate this label control.

**Programming Tip:** Many languages use layout managers to specify how controls are position in a window.

The remainder of this class file consists of a series of modules that will act as the event handlers for application events. For the lookUpOrder Module, the order number is acquired from the screen's entry control and passes it to the ReceivingClass so that the data associated with that order number can be retrieved and printed on the screen.

The other event handlers identified in this class have yet to be completed. Once the coding for this application is finished, then but the existing code or new code will be called to complete the project.

In some cases, the program coding necessary for the processing of the exception buttons might be the responsibility of another programmer. In this wiki example, the exception button has been assigned as an end-of-chapter task in the process button logic has been conveniently omitted as to not make this example more complex by introducing file or database operations to the program. As you can imagine, the process button would certainly lead to an update of a data base indicating that ordered products are now placed in inventory.

TEST

Testing for this project needs to be done in three stages. The initial testing will occur as the program is being created. As the programmer adds lines of code, they will perform a unit test. A unit test is done on small blocks of code to ensure that that block is functioning correctly. Once the larger program is complete, more testing can be done to simulate what a typical user might do during normal operation (including error situations). This program test is very important but not the end of testing.

In complex environments like that found at Cactus Books and Tapes, it is critical that a System Test is performed. With client server applications especially, system testing must be done to simulate the operation with an adequate load (number of users executing the program) on the system. Adequate load would mean having all three employees from the receiving department working on the program while others in the company were doing different activities on the network. This type of testing can sometimes be difficult to implement. With a small company like Cactus Books and Tapes, it is perhaps a lot more feasible but if we took a larger company that had 50, 100 or thousands of employees, the testing would be more difficult to simulate. If the situation dictates, the company and application is small enough, and then a system test can yield very valuable information as to the success of the program. It is never good to define problems in logic or programming late in the process but always better than after the program has been implemented into production.

**Program Test:** Program test involves testing all the programs associated with an application. It is not unusual for an application to contain many classes or programs. Program test would make sure all of the pieces of an application are working together correctly.

**System Test:** System test would be an application test that involved the operation of the program while other applications were also running. The goal in this test is to make sure the new application behaves well with other programs found on the system or network. Special emphasis is placed on system performance and availability.

**Production:** Production is a term used to describe when an application has passed testing and is ready for everyday use by the user.

**Related Subject:** Pseudo Code and Code Patterns, are they the same? - As you become a more experienced programmer, you might find yourself moving away from a pseudo code model. I think, if asked, most experienced programmers they would tell you that they do not used pseudo code unless it is required or if they are working on a logic problem that they have not experienced before. Most programmers learn that all programs are a series of similar program code patterns strung together to solve the particularities of each program. We are creatures of habit and will naturally learn and use code that has worked for us before. With these code patterns stored in the recesses of our brain, we recall these blocks of code and use them much the same as pseudo code. In reality, the experienced programmer will draw on his inventory of code patterns and supplement them with pseudo code when confronted with a new logic program. I should point out. As programmers, we usually don't remember line by line our code patterns but instead the programs we used them in. We open that program and find the code pattern as we remembered it. This way, the pattern is reused (copy and pasted) into a new program.

## TEST MATRIX

Below, I have put together a first draft test matrix for the new receiving program. As I do more testing, I might decide to add more test cases to this list. As I gain experience with the user, they will help me uncover more texts for my test matrix. I can also use this test matrix for regression testing when I have to make future changes but still be sure that the program still meets all of the original logic processing criteria.

### **Test Passed Failed Notes**

Lookup Button: Order number left blank should return and display "Order Not Found"  
Yes

Lookup Button: Good order number should return and display the rest of record Yes

Lookup Button: Bad order number should return and display "Order Not Found" Yes

Process Order Button should mark order processed in Inventory File Partial \* \* File has not been added to logic processing (currently working with test data in array) but correct module called. Future system testing needed.

Exception Button should allow order to should partial shipment Partial \* \* File has not been added to logic processing (currently working with test data in array) but correct module called. Future system testing needed.

Exception Button should allow order to should description change Partial \* \* File has not been added to logic processing (currently working with test data in array) but correct module called. Future system testing needed.

Print Button should display order in output dialog box Yes

End Program should close graphics interface and end program Yes

**Key Concept:** A test matrix is a living document that changes as the program changes. Test cases will be added, changed and deleted as changes are made to the program.

## IMPLEMENTATION

Once the program has been completely tested, it is ready for implementation. Implementation offers some options. For a direct cutover implementation, where new programs are loaded and old programs are deactivated, you must be sufficiently satisfied that the system test was extensive enough to feel confident the program is ready to run and has no significant errors. This is a "cold turkey" approach where you have committed to a implementation of the new programs with the idea of not returning back to the existing system. This approach is the most dramatic and some would say the most risky.

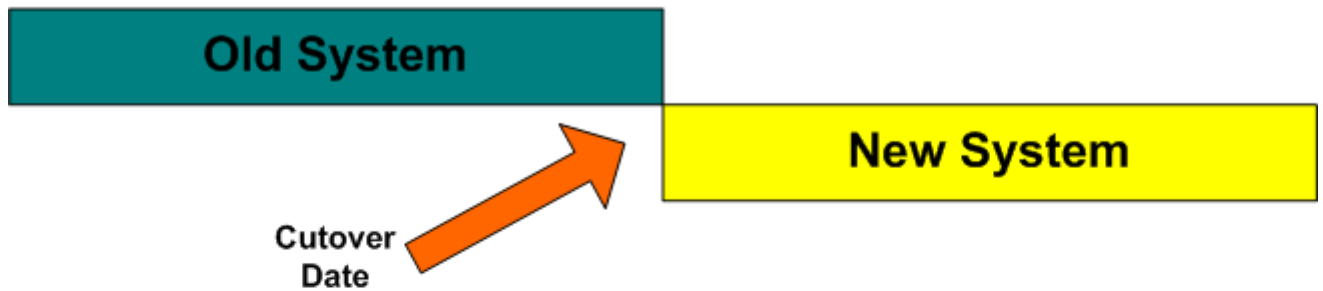
If the program is especially complex or if failure would cause significant damage to the user or organization, a parallel implementation might be in order. When we do a parallel implementation where essentially saying that the new system in the old system will operate side by side for some period of time. At the end of the test period, outputs from both systems are compared to determine if both programs came up with the same results. If the new program returns the same outputs as the old program then we can feel confident that the program is now ready to be turned over to production .

**Direct cutover:** Direct cutover is an implementation strategy where a date is chosen to load the new programs and discontinue use of the existing software. This approach is the most severe and requires extensive pre-test since once the new

software is loaded there is typically no turning back. This approach, if successful, is the least expensive method of implementation.

**Parallel cutover:** A parallel cutover implies that the new and existing systems will operate in tandem for some period of time. At the end of the parallel implementation, outputs from the new system are compared with the old system to make sure the new systems will perform designated requirements. This approach is considered a safer approach but more costly since it requires work to be done twice. Once each for the existing and new system.

## Direct Cutover Implmentation



## Parallel Implementation

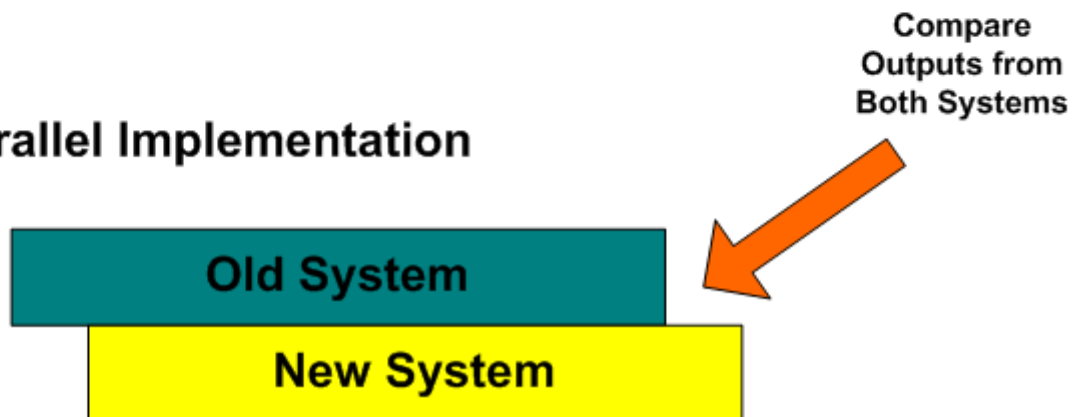


Figure 5: Direct Cutover versus Parallel Implementation

The Cactus Books and Tapes operation is small enough to easily implement a one week parallel test run. This particular receiving area is not high volume and that implication of effort, although it would slow the workers down, would not impact the timing of their job or require overtime or additional expense. The Chavez sisters would also feel more confident if the workers had the opportunity to try the system for a week and then give feedback as to whether the programs are ready to go or if additional rework is required. There is an expense to this type of implementation but the Chavez sisters feel that the added expense is justified if the program has been evaluated under typical work conditions.

## MAINTENANCE

With the extensive use of classes and reusable modules, this program has been well structured for any additional maintenance that may come along. It would be naive unrealistic to believe that this program will not have to be raved visited and changes made to it. As the business changes so will business logic and therefore the logic in the programs that have been created. By creating a reusable class file module, we are reusing programs that reflect the receiving process business rules. If a receiving business rule changes (i.e. maybe the next generation of receiving software will accommodate bar code readers), then the change need only be made in one file (the receiving class file).

We have also developed the program with ample program code comments. These comments should help the next programmer understand the more complicated and difficult pieces of logic that accompany this program. We followed PYTHON coding conventions and used naming standards that were consistent and familiar to other PYTHON programmers. This too will also make maintenance of these programs much more efficient and effective.

**Programming Tip:** Reusable class modules can improve the maintainability of programs by using one class in many programs. Changes need one be made to one class as all implementations of that class share the same logic.

## Status Check

Can you identify the program development phases of the Cactus Books and Tapes maintenance project?

Why was the storyboard so important to this project?

What would be your recommendations be for an implementation strategy?

## INTERNET/INTRANET VERSUS TRADITIONAL PROGRAMS

Perhaps the most significant difference between programming on the Internet versus programming on a standalone computer is the environment. An application written to run over the Internet needs to be designed as a client server application. Since the client and server both share in the processing, program logic written for Internet applications need to be synchronized between client and server to operate correctly. In other words, part of the program will be in the client and part of the program will be on the server.

Synchronization is important first from a division of logic perspective. The client program needs to be run from the browser. Application code is delivered to the browser imbedded within the web page sent from the web server. This client application takes the form of a script or applet. JavaScript is the most popular client side scripting language and Java is the most popular applet program. Even though both of these programming languages include the word Java, they are in fact very different. These client side programs provide some subset of what the entire Internet application will deliver. For example, a Java applet might deliver a program that is run on the client computer and provides the user the ability to calculate a monthly car payment. Working with this client program is a server side program that may take that payment calculation and send it back to the server where a server side program would search a new car inventory for vehicles that could be sold at the price requested.

The server side program might be written with a programming language like Visual Basic.Net or PERL. The program runs on the server and after the database search is complete will transmit back to the client browser a new web page showing the

results of the requested data. In this example, the Internet application consisted of two programs working in concert with each other but running from separate computers.

Related Subject: The Internet Can Be The Great Equalizer!

In my Visual Basic.Net classes, we discuss the limitations of VB programming. One of the limitations is that VB programs are currently only executable on Microsoft Windows Operating systems. With the .Net Framework, this isn't a real limitation. For example, if you have someone that has an Apple MvIntosh, they would normally be out of luck using your VB programs. That is unless you develop the application for the Internet. Since VB Internet programs run on all browsers (running on different operating systems), it is possible to parlay your experience as a VB programmer to create web based programs to run via a browser on a variety of operating systems (Apple, Linux, etc.). Pretty cool huh!

**Internet Programming:** From a Logic and Programming Perspective - From a logic perspective, you have structures and objects available with Internet applications and standalone applications. You can make use of modules and classes to promote code reusability and pseudo code, flowcharts and UML modeling options are still viable in programs created for the Internet.

Logic Tip: Internet programming has all of the same support for structures, modules and classes as with traditional programming. Pseudo code and flowcharts can also be used for logic modeling.

Where you will see a difference is the coding of server side applications. Since data is traveling back in forth from browser to server (directed much like GUI event driven programming), as the program leaves one environment to another, the state of variables is lost. This means the variable loses scope and is wipe clean each time the web page has to go to the server for server side programming. If the program contains counters or accumulators, data for these variables is lost on each trip to the server. To accommodate for this behavior, session variables need to be used to maintain state. This means that before the program leaves the client, variables need to be saved to session variables and reloaded back into variables once the application returns from the server. If you get the opportunity to program in this environment, you will need to make sure you accommodate the stateless nature of variables in Internet programming.

**State:** State describes the ability of variables to store a value throughout the life of the application. Internet applications are stateless in that variables lose their values when the web page needs to visit the server for processing.

**Session Variables:** Special variables that are stored on the server and can be recalled and used in the client program after a trip to the web server.

Programming Tip: Internet programs are said to be stateless since variable data is lost each time the Internet program needs to run code from the server.

## FROM A DESIGN PERSPECTIVE

One of the nice things about traditional programming is that you usually know what hardware (particularly output devices) your users will have. What about the Internet. Since it is such a graphically rich environment, how can you count on knowing the screen size or resolution of your users. Mostly you can't. What you can do is develop to industry standards. In other words, the Internet programmer will know that the standard screen resolution is 800 by 600 pixels. Some screens are less and some are more. Why do you care, especially since you know someone will be disappointed, because if you are at standard you will satisfy the greatest number. It is also important to know what other devices are included with the standard home PC. For example, most home personal computers contain multimedia support for sound and video. Knowing that most computers have this capability will perhaps allow you to consider an alternative that would otherwise be impractical if it did not exist on the average PC.

## WEB SERVICES EXPLAINED

Another Internet based technology which has recently gained increased importance is web services. You can think of web services as modules or class objects deliverable over the Internet. This brings new meaning to reusability. Not only is the code reusable on a single computer but it can be accessed via the Internet at a server located in another country. Some popular web services have been weather

web services that allow programmers to connect to servers that will pass back weather data based on a zip code. Web services can provide support to both standalone applications running on a PC with Internet access or for an Internet program running from a web browser. The impacts of this technology (along with XML) have only just been understood and the travel, healthcare, government and educational sectors should benefit greatly.

## Status Check

What is the significance of program state in an application running from an Internet server?

What is the function of a web service?

## PUTTING IT TO USE

The intent of this section is to introduce the new programmer to standards and techniques frequently used by professional programmers. Topics in this section relate to concepts introduced in the chapter but with a more vocational or occupational focus for students considering a career in programming.

Why I Like Programming And Being A Programmer.

I thought it only appropriate for this last chapter to share one programmer's perceptions on why they like programming and the job of programmer. That person is me. I hope my passion for programming has been demonstrated in this wiki. I was asked once why I liked to program so much (and why I programmed as a hobby after a long day of programming on the job). I came up with several reasons. I am not sure all programmers would agree with my reasons but I suspect they would share some.

- I like solving problems - I am not a puzzle person. Puzzles are interesting but I really enjoy solving problems, real programs. Programming offers me the ability to take business problems and solve them with the help of the computer. I can be thoughtful, creative and a problem solver all at the same time. A perfect combination, I think.

- I like the predictability of programs - With so many things being uncontrollable, I like the fact that computer logic is predictable and consistent. I know if the programming statements are in the correct sequence, I will come up with the same solution every time. Likewise, there is something reassuring that if there is some problem with my program, I can back track and analyze and eventually come up with a reason for the problem. No mysteries here.
- I like to create - I have always been a builder. I like making things from scratch. Sure, it is great buying stuff from the store, but sometimes building something gives you a better perspective. For example, I built my own golf clubs. I did save a little money but mostly I just wanted to learn how it was done. Computers can simulate a lot of things and you can create everything from games to business applications. In all cases, you have the opportunity to build and experiment.
- A dynamic environment - I used to tell people if they didn't like a technology, wait a couple of months and it will reinvent itself. One thing about programming, there is never a dull moment and always something new to learn. If you like change, you will probably like programming.
- Lifetime learning - I have always liked school and loved to learn about new things. Not just topics concerning computers but one thing that is a definite with programming is that there will also be something new to learn. The dynamic environment that I mentioned in the previous bullet drives the need for learning. Sometimes the learning is done by yourself and sometime with others. I like to learn by doing and with programming there is an opportunity to practice hands on learning by creating the program and trying different program classes and functions.

I am just one programmer. Find another programmer and ask them why they like programming and get their advice on the best way to learn more about a very rewarding and challenging career option. Good Luck!

## CHAPTER REVIEW

Chapter summary, highlights, key terms, short answer questions, quizzes and case studies to reinforce topics covered in this chapter.

## CHAPTER SUMMARY

After finishing this chapter you should understand and in some cases, demonstrate the following topics:

- Explain and implement program maintenance using the program development lifecycle.
  - Most of the programming done by the programmer is program maintenance or modifying and updating existing applications.
  - Businesses try to maintain existing software to preserve their investment. The development of new systems is an expensive process and once a set of programs has been developed and working successfully, there is a financial and managerial reluctance to change something that is working well for the organization.
  - New programmers especially benefit from doing programming maintenance. Since new programmers are still learning the tools of the trade, they can develop their programming skills by working with existing applications. The existing application acts like a template for the new programmer to follow.
  - Program maintenance benefits the new programmer by providing the opportunity to see another programmer's work. This will strengthen the new programmer's knowledge of programming, debugging skills, recognition of programming patterns and confidence to take on more complicated programming projects.
- Differentiate between creating a new program and maintaining an existing program.
  - Programming an new program or an existing program remains the same in regards to following the program development process. In other words, the programmer still works with a logic model as input to creating program instructions.
  - Updating or maintaining an existing program may require different analysis to understand the original logic models and programming. This detective work is valuable in understanding difference approaches to solving problems with logic.
- Explain and demonstrate the modularization of program code to promote code reusability and simplified program maintenance.
  - Using class files and reusable class modules allows the programmer to reduce the cost of programming by reducing duplication. Many programming operations are common logic/code patterns that can be repeated in many other programs.
  - Creating common reusable modules reduces the cost of program maintenance by limiting the number of places that program logic will need to be updated.

- Describe implementation alternatives for new and updated programs
  - A direct cut over implementation is used when the programmer is sufficiently satisfied that the system test was extensive enough to feel confident then updated program can replace the existing programs. This approach is more dramatic with increased risk but with lower implementation costs.
  - A parallel cut over implies that the new and existing systems will operate in tandem for some period of time. At the end of the parallel implementation, outputs from the new system are compared with the old system to make sure the new systems will perform designated requirements. This approach is safer but more costly since it requires work to be done twice.
- Distinguish and differentiate between traditional desk top and Internet application development.
  - An application written to run over the Internet is designed as a client / server application. The client and server both share in the processing and therefore both will have program code to execute.
  - The client in Internet programming is a user running an Internet browser and the server is another computer running as an Internet Server. The user, with their browser, sends requests to the Internet server who sends back a web page.
  - Because the entire Internet program exists in two locations, the program logic written for Internet applications need to be synchronized between client and server to operate correctly.
  - Because program control is split between the client and server, values stored in client side variables are lost when doing processing on the server. Client data must be stored in session variables to preserve client values.
  - Values stored in a client application are said to be stateless.

## CHAPTER KEY TERMS

Black box

Business Rules

Direct Cutover Implementation

Flow layout manager

Grid layout manager

Layout Manager

Parallel Implementation

Production

Program Maintenance

Program Test

Prototype

PYTHON Module

Reverse Engineer

Scope creep

Session variables

Variable State

System Test

Unit testing