

## **STRUCTURED IS, AS STRUCTURED DOES**

**System.out.println(“OBJECTIVE”);**

This paper is designed to provide the reader with a basic knowledge of my findings as I have set out to uncover the “basics” of Visual Basic (no pun intended). Though it began as an opportunity to analyze the language itself, the objective slowly meandered away from the complex analysis of Visual Basic, and steered towards the structure of structured languages. So now, the purpose of this paper is to emphasize the importance of structured programming, using Visual Basic as a tool of reference.

**System.out.println(“WHAT IS STRUCTURED PROGRAMMING?”);**

### ***The Dilemma—***

I compare the assignment of this paper to running a race. As the starting gun sounded, I found myself leaping from the block, sprinting towards a glorious victory. However, as I turned from reflex to reason, I realized that there were no lines guiding my path of destination. Not only that, but there was also no determined destination; no finish line awaiting my arrival; no shout of excitement from a crowd; just me, my computer, and the library. Some might call this “freedom”—I called it “frustration.” With no definite path to follow, I found myself spending countless hours in the Harold B. Lee library, just trying to orient myself as to what direction I was headed in.

It did not take long to notice that the computer science section of the library was scarcely trafficked at. Putting this to my advantage, I planted myself on a very uninviting piece of carpet, hoping that the tender sprouts of my mind would quickly spring into a vast forest of knowledge. My hope of this soon withered, and died (probably due in part to poor lighting, and equally pitiable ventilation). As I reached for book after book, I finally came up with one great conclusion: I had no idea of what the term “structured programming” really meant.

There were so many different books, with a plethora of opinions on this controversial subject, and with an equally immense amount of views on what was required in structured programming. I just couldn't seem to get a clear answer on the 'configuration' of structured programming. Slowly I uprooted myself from my spot on the ground, and slothed my way out of the library.

*The Answer—*

Some people claim that time is the great healer of wounded emotions; I personally feel that healing can often be found in a tall glass of skim milk and a couple of frozen cupcakes. Therefore, deciding this to be the next logical step, I invested a few cents of my fiancé's Dining Plus bucks, and engaged in some serious meditation on the great dilemma I was faced with. Like a Mack-truck with no brakes, the answer seemed to hit me all at once: if I was having trouble deciphering what structured programming is, then maybe that is what I should declare as one of the focal points of my paper! So here it is, the great fulcrum of my studies...

Although the difference of opinion spanned quite an array of ideas, there were a few points that seemed very consistent with every notable author. I have selected the

three that seemed the most widely accepted, and have chosen to word them exactly as cited in Dan Robinson's "Fundamentals of Structured Program Design." They are as follows:

- 1) Each procedure performs one task and one task only.
- 2) Each procedure has only one exit and one entrance.
- 3) Each procedure does what the name implies.

***The Analysis—***

Because my research was developed around the model of Visual Basic, I would now like to discuss how these three principles relate to Visual Basic, and how they function as principles in "good" programming.

The first fundamental concept of having structure to programming is the idea that each procedure performs only one given task. This principle can be very easy to interpret within the implied meaning; however, there is also room for valid arguments as to the legitimacy of such a statement. I say this with the thought in mind that one could easily propose the question, "what is a procedure?" For the purpose of simplifying the discussion, a procedure will hereby be defined as the smallest section of code, which is intended to perform some type of duty. Meaning, if the program were written out in flow-chart form, then a procedure would be a method, or function, which takes the program from one state into another. In Visual Basic, there are two types of procedures; namely, subroutines (called 'subs') and functions. The only difference between the two of these being that a function returns some value, whereas a sub does not. In either case, each procedure has the capability of having only one given task. Of course, this task may be intricately detailed, and contain a vast amount of coding, but, in the end, it will only

perform one task. If one were to write a program containing some sort of “mouse-clicked” event, then the program could only contain one such event. In order for a program to be classified as structured, there can only be one “mouse-clicked” event for that particular object. In this respect, Virtual Basic is very structured. The program editor and compiler are very strict in not allowing a sub or function to be declared twice, as doing separate tasks.

The second principle is perhaps the most controversial in today’s programming world. Each defined procedure can have only one exit and one entrance. This has been the topic of great debate among programmers for quite some time. Spawned largely by Edsger W. Dijkstra’s historical article *Go To Statement Considered Harmful*, a great debate over the use of statements such as “GoTo,” has been a hot topic among programmers of all levels. The major argument against this type of coding is that it creates a somewhat unpredictable, and highly unreadable program base. Being able to jump from within one function to another, then to another, and so on, makes it very difficult for somebody to be able to look at the code of a program and see exactly what is going on. In the book “Hacker’s Guide to Visual Basic,” it describes, very briefly, the ability to use a GoTo statement within the capacities of Visual Basic. However, the book is also very clear in stating that it is not good practice to use such a statement, and avoids giving any types of examples for how to use it. In any of the other books I researched about this topic, I found that none of them mentioned ever using such a statement while programming in Visual Basic (see the Bibliography at the end of the text). I tried experimenting on my own with this infamous statement, but quickly relinquished those efforts as I determined that it was more trouble to learn how to use it than it was

convenient. One of the features of Visual Basic is that it makes it easy to write subs and functions, so that it is hardly even a temptation to use something even remotely detrimental as the GoTo. Thus it may easily be said that Visual Basic is a language with unstructured capabilities, but highly structured preferences. As I was researching this idea, I often wondered if Visual Basic could be compared to a 'wolf in sheep's clothing.' Perhaps this is quite an extreme to conclude, but then again, maybe it isn't.

The third principle is one that might seem at first to be somewhat nebulous in its direct relation to structured programming. Titling a procedure after its primary function seems to be something more of a technique, rather than a requirement. However, with even the slightest consideration, it becomes quite clear that it is an absolute necessity in structured programming. The naming of variables, and of procedures, is something that is far too often neglected in importance. Many programmers become lazy in documenting their programs, and often claim that their code is self-documenting. However, when viewed by another programmer, it is often discovered that such a program is really not self-documenting at all. I refer to this as a technique because the naming of variables, and so forth, is left solely to the writer of the code. Although it does not affect the running of the program, poor use of naming technique can create quite a hassle in the debugging, or revising process. That which sustains the foundation of structured programming is the concept of organization, and general flow of the program. A program laced with poorly named variables is just as unreadable as a program loaded with GoTo's. Visual Basic is very compatible with many different variable names. There are virtually no limits on names, so that the user may be descriptive when choosing titles for procedures and names for variables.

Another concept that is not within the three principles of structured programming is that of re-usable code. Professor Helps of the EIT program of Brigham Young University brought this topic to my attention, during a discussion we had on the subject of Visual Basic. The question came up as to whether or not Visual Basic was capable of reusing large portions of code. I found the answer to this question while writing a program that was similar to a program I had previously written. In this program, there was a particular form that I desired to implement in my current project. It did not take a lot of searching before I found the option to import forms from another project. After importing a form, it was quite simple to be able to make slight corrections to the form, and then simply save it as a different form. This is a very useful concept to be able to implement in programming. This is something that is very difficult in an unstructured program, due to the fact that if a portion of the program were copied, it might refer to another piece of the program that doesn't exist, or that is incompatible with the current project. Visual Basic has made this task practically effortless. Also, Visual basic includes several libraries of modules that can be imported into any program. The user is also capable of creating their own module library, but this involves a deeper understanding of programming, and how Visual basic works.

**System.out.println("CONCLUSION");**

It is very difficult to try and touch lightly on a subject so vast and intricate as structured programming. There are numberless topics of discussion, and literally thousands of references concerning this matter. I found myself overwhelmed with a treasure chest of knowledge so extensive that I could never dream of utilizing all of the

information available. My determination is that structured programming is vital in the field of computer science and any related software field. Having true structure to a language provides the writer with great training and experience in forming habits that will be very useful in the field. From my limited experience with Visual Basic, I feel that it is a language that is as structured as its programmer. This returns me to my previous analogy of a wolf in sheep's clothing. Although Visual Basic may be very encouraging of structuring the programs written in Visual Basic, it does extremely little to enforce this idea. On the 'outside,' it seems to be designed towards structure; on the 'inside,' it can become very messy, and may also be designed in a way that breaks the rules of good structure. As much as it may appear to be structured, Visual Basic simply does not comply with all of the characteristics of a structured programming language. Perhaps I wouldn't go so far as to call it a 'ravens wolf,' but in respect to the analogy, it is definitely not 100% wool.

## **BIBLIOGRAPHY**

- 1) Michael Halvorson. *Step by Step*. Microsoft Press, 1997.
- 2) Vincent Chen and John Montgomery. *Hacker's Guide to Visual Basic*.  
Vincent Chen and John Montgomery, 1996.
- 3) Bob Reselman and Richard Peasley. *Using Visual Basic 6*.
- 4) Len Dorfman. *Structured Assembly Language*. Windcrest Books, 1990.
- 5) Scot Hillier. *Inside Visual Basic*. Microsoft Press, 1996.
- 6) Dan Robinson. *Fundamentals of Structured Program Design*. Prentice-Hall Inc,  
2000.
- 7) Jeanette Allen. University Lecture. 12/4/1998.