

DON'T GIVE UP ON COBOL BEFORE YOU AT LEAST TRY WRITING ONE COMPUTER SOFTWARE PROGRAM AS INTENDED AND ENVISIONED BY THE PRESTIGIOUS COBOL COMMITTEE.

Why has the Computer Software Industry been at the bottom of Industry's productivity charts from its very inception?

In order to solve a problem you must first recognize the cause and then seek a solution. "You can't put the cart before the horse".

Before the advent of COBOL all the computer languages were machine oriented and difficult to test, debug and maintain. None of those languages were conducive to clearly understandable coding, "the key to coding productivity ". Therefore Our Software Industry had no easy solution. Then in 1959 the COBOL committee recognizing the cause, decided to create a language whose major attribute would be its conduciveness to clearly understandable coding. Now our Software Industry recognized the cause and found the solution for productivity.

Now let me say what I'm sure will be considered an unbelievable statement in our Software Industry. In 49 years not one software program has been written in COBOL as envisioned by its prestigious creators. What has been written is what can only be termed, "broken COBOL". Broken Cobol, like broken English, ranges from poorly understandable to nearly incomprehensible coding. Yet these broken COBOL programs, whose understandability was no better than the machine oriented languages, were used to measure productivity against all other computer languages. No measurements were ever made with COBOL as envisioned by its creators. Why? There just haven't been any written. This led to the slow demise of COBOL implementation.

The first step toward a dramatic increase in Software Productivity is using COBOL as the implementation language envisioned by the COBOL committee and ban any broken COBOL software. "One small step for COBOL code, one giant leap for Software Productivity." From then on COBOL will, as once before, be the most used computer language it once was and our Software Industry would begin to rise in the Industry Productivity Chart. There are steps that can slowly clarify the broken COBOL already in production but for them there is no silver bullet. To continue, as is, implementing additional software, using machine oriented languages for tasks best suited for basic COBOL will only place our industry further and further down on the Industry Productivity Chart.

The only reason there are millions of broken COBOL programs still being maintained in our industry is, these programs do the task they were assigned. This is proof positive that basic COBOL can effectively handle the greatest variety of software tasks. You know the old cliché, "Don't try to fix what is working." The fact that they were all written in broken COBOL doesn't prevent them from doing the task they were originally assigned. Their negativity is the difficulty in testing, debugging and maintaining them. That is the reason our Software Industry has been at the bottom of Industry's Productivity Chart year after year.

You might rightly ask, "How did all those COBOL programs become broken?" The answer, in one word, is ABBREVIATIONS. From 1959 on, when COBOL was created, so called experts completely disregarded the COBOL envisioned by its prestigious creators. In fact they preached the exact opposite of those creators.

The "experts" erroneous logic and rhetoric was:

"Since COBOL is based in English, clearly understandable coding need no longer be a problem."

“The fact that COBOL is in English assures it's understandability.”

"In addition all efforts should be directed toward reducing software implementation time."

"What better way of reducing implementation time than flooding the coding with abbreviations."

"The more abbreviations the less coding time is needed."

"Standard abbreviations are equally understandable as fully spelled words."

These misconceptions have been devastating for COBOL coding productivity and in turn Software productivity.

What they didn't take into account was that there are a very limited number of acceptable standard abbreviations that will reduce implementation time. Programmers in their misguided zeal to reduce time began flooding their coding with their own abbreviations. Let's face it. If abbreviations are as clearly understandable as fully spelled words how come books, magazines, newspapers, letters etcetera are not flooded with abbreviations as is COBOL coding. (notice I've used the fully spelled word ETCETERA instead of the abbreviated version, etc, even though etc is considered by many as understandable if not more understandable than etcetera). The reason is once management allows one abbreviation, programmers will take that as encouragement to use their own abbreviations. There is no way of controlling their using any abbreviations they alone think of as clearly understandable. They will and have flooded the coding with these abbreviations until it destroys the one major reason for COBOL in the first place, "Understandable coding". It's no abbreviations at all or destruction of understandability.

Understandable coding has always been the key to programming productivity. The following 2 examples of COBOL coding make it evident how abbreviations destroy understandable coding. In addition, as time goes by, remembering the correct spelling of the full word is much more likely than remembering the abbreviation's spelling: Example 1 is coding as envisioned by the creators of COBOL. Example 2 is broken COBOL coding. Can there be any doubt as to which is productive and which is not productive? Which coding is much easier to test, debug and maintain?

```
1) IF PATIENT-PRIMARY-INSURANCE EQUALS "MEDICARE"  
   PERFORM BILL-MEDICARE-PATIENT THRU  
       PRIMARY-MEDICARE-EXIT.
```

```
2) IF PAT-PRIM-INS EQUAL "MR"  
   PERFORM BILL-MR-PAT THRU  
       PRIM-MR-EXIT.
```

Just to see how far off base those so called experts were, here are quotes from the COBOL Committee members.

1) "Majority of the group favored maximum use of simple English language."

2) "The need is for a programming language that is easier to use even if less powerful."

3) "It certainly was intended (and expected) that the language could be used by novice programmers and read by management. We felt the readability could be achieved because of the intended use of English ... most of our concentration was on making it easier to read."

4) "The driving force behind consideration of all the "statements" was the concept of readability."

5) " and there was a definite emphasis on ease of reading not ease of writing."

From day one of COBOL use, none of these quotes were adhered to. The rejection of quote number 5 was the most destructive because it resulted in giving ease of writing the highest priority. When ease of writing is given priority, ease of reading suffers dramatically but when ease of reading (understandability) is given priority, it enhances ease of writing and just as important or even more, ease of maintenance. I challenge any Software manager to find a COBOL program in their library that reflects these quotes. All they will find is "broken COBOL" not the COBOL envisioned by it's creators. Structured coding can be very beneficial but not within "broken COBOL". Structuring poorly understandable code is like paving a dirt road to increase motor speed but leaving the road signs in a foreign language. What benefit is there in going faster if you don't know where you are going. For our Software Industry to have discarded COBOL as it's major implementation language based on an evaluation of broken COBOL was the most costly move it has ever taken. COBOL, as both designed and envisioned by it's creators, is as powerfully productive today as it was when it was first presented to our Industry. If we insist our programmers write COBOL as it was designed it will again become the most used language. Only this time it will be based on it's actual productivity, not on productivity misconceptions. If management has any doubt, there is a test they can run which should prove it.

Jerry Sitner
Clarity Concept Systems
nycnmi@aol.com