Is COBOL-8x cost effective?

by MARCO FIORELLO
Titan Systems
Vienna, Virginia
and
JOHN CUGINI
National Bureau of Standards
Washington, D.C.

ABSTRACT

The purpose of the study is to assess the estimated costs and benefits to the federal government that would result from adoption of the proposed revision of American National Standard COBOL as a Federal Information Processing Standard (FIPS). Potential benefits of $90.2 million have been identified, stemming primarily from improved productivity in both the development and maintenance of COBOL programs. Estimated costs of $17.9 million have been identified, arising principally from the effort needed to convert old COBOL programs to the new specification, which is incompatible in some respects with the current one. In support of the study, we conducted interviews with federal ADP managers and officials, and also analyzed more than one thousand federal COBOL programs for various syntactic characteristics.
STUDY SCOPE AND QUALIFICATIONS

The scope of this study is limited to COBOL-related effects on the federal ADP community. Of course similar effects may be expected in the private sector insofar as the characteristics of its COBOL usage resemble those of the federal government.

In this analysis, we are concerned with effects that may result if the proposed changes to ANSI COBOL-74 are also adopted in the Federal Information Processing Standard (FIPS) for COBOL. Data available on applications software development and maintenance in the federal government are general and approximate in nature, and are particularly limited regarding any one specific programming language such as COBOL (although COBOL is by far the most commonly used language within the government, and therefore can hardly be regarded as atypical). We augmented the available general data with staff interviews at nine federal agencies and with a detailed analysis of a sample of 1068 COBOL programs from eleven federal agencies.

BASE CASE STATISTICS

The base case statistics are derived from various reference materials, cited in this document, and the study survey and program sample.

**Programmer Pool**

For the past 10 years the number of federal agency staff programmers has remained fairly steady—in the range of 118,000–120,000 staff-years. Of those work-years, roughly 60% were primarily for COBOL-related activities in 1980, with a growth to 65% projected for 1985. Depending upon the federal agency, the annual programmer turnover rate will vary from a low of 10% to a high of 30%. A reasonable average appears to be 20%. In most installations, more than half of the staff are devoted to maintenance (corrective, adaptive, and perfective) activities, which reflect the life cycle distribution of application software costs.

**COBOL Program Inventory**

There are roughly 500,000 application software programs in the federal inventory. Of these, 50% to 60% are in some form of COBOL. Very few, 5–10%, of these 250,000–300,000 COBOL programs are in full conformance with the current COBOL FIPS 21-1. The average COBOL program in our sample contains about 1270 lines of source code and was developed about six years ago. This latter figure compares reasonably well with the 5.4-year estimate given in Reference 4.

In our sample of 1068 COBOL programs, with more than 1.3 million lines of code from 11 federal agencies, we learned that 80% use one or more of the 50 proposed incompatible changes analyzed in this study. If we discount the somewhat special case of the incompatibility concerning the DISPLAY verb (see below), this figure drops to about 40%.

An important point about interpretation of the statistics is that the detection of incompatibilities was done by a syntactic scan of the source code. Where the incompatibility involves a syntactic change (e.g., the deletion of ENTER), this is a reliable procedure. In those cases where the semantics are being changed or clarified (EXIT PROGRAM closing out PERFORMs, for example), however, the best that can be done is to look for source code where such a change might make a difference. This analysis represents, therefore, only a worst-case estimate. The DISPLAY incompatibility is an especially striking example of this. Syntactically, we counted every occurrence of DISPLAY as an incompatibility, even though the great majority of vendors currently implement this verb as described in the revision. The age of programs was determined simply by the contents (if any) of the DATE-WRITTEN paragraph. This is, of course, not a foolproof metric. Nonetheless, we feel the data are worth presenting, and they do agree with a previous General Accounting Office estimate. We were able to find a DATE-WRITTEN entry in 58% of the sample programs.

**Application Program Conversion and Maintenance**

In the current setting, the source code for application programs is updated for a variety of reasons:

1. Conversion to a new or modified host system (hardware or software)
2. Accommodation of modified functional requirements
3. Correction of errors detected in the code
4. Reprogramming to reduce the number of compilers used or to improve processing efficiency

The interviews with federal ADP managers revealed that COBOL programs are recompiled at least once a year because of maintenance activities, and sometimes as often as six times annually. A reasonable average is two or three times per year.
COSTS AND BENEFITS

Program Development

The proposed revised standard COBOL features that have the potential to enhance programmer productivity include the following:

1. Nested programs provide a facility for segmenting large programs into smaller logical units
2. Scope delimiters assist in the generation of structured code
3. Reference modification allows the programmer to access any part (substring) of a character field without having to redefine the item
4. EVALUATE statements incorporate a well-known construct from structured programming practices, the multi-way conditional
5. Other constructs that should prove useful in clearing up previously awkward aspects of COBOL are the ability to PERFORM routines in-line, set up tables with more than three dimensions, accept as well as generate numbers in edited form, and INITIALIZE the values in tables.

Of the above, we were able to search the sample programs for programming practices in which features 3 and 4 could have been used and would have saved time for the programmer. For feature 3, we searched for data items defined as PIC X (one character only) with an OCCURS clause. For feature 4, we searched for GO TO ... DEPENDING ON. In our sample, roughly 22% of the programs could have employed feature 3, and 5% could have used feature 4.

Feature 1 will be especially useful for organizing large programs. In our sample, programs with more than 1500 lines of source code account for approximately 65% of all the lines of code (even though they constitute only 25% of all programs). We note that all COBOL programs can make use of feature 2. Moreover, in the interviews conducted with representatives of various federal agencies, this enhancement was the one most often cited as potentially improving programming practice. Thus, we anticipate that the enhancements to COBOL will apply to some degree to virtually all programs in the federal inventory. For a considerable percentage of the code, the effect will be quite significant.

We make the following conservative assumptions: First that COBOL-8x will be adopted by federal agencies at the rate of approximately 10% per year, and second, that the use of the advantageous features will result in a 5% increase in productivity during the coding phase of development. These assumptions generate a savings of $36.1 million over the next ten years.

Program Maintenance

Program maintenance concerns those activities involving correcting, perfecting, and adapting existing application software, and currently represents 50–70% of the program life cycle costs.\textsuperscript{3,5}

The principal ways in which the proposed changes to standard COBOL would affect the maintenance function are by increasing the understandability of COBOL programs and by reducing the error-prone features of COBOL-74. The enhancements to the language cited above under program development apply strongly to program maintenance as well, since they make it easier to read and write code. Many of the proposed 50 incompatibility changes are intended to eliminate or clarify certain error-prone or ambiguous features of the current COBOL standard.

Again, assuming that federal agencies adopt COBOL-8x at the rate of 10% per year, and that the advantages of COBOL-8x generate a 1% savings in maintenance activities, the resulting savings will be $54.1 million over the next 10 years.

Program Conversion

Software conversion is the transformation, without functional change, of computer programs and data elements to new hardware or software processing environments. The greater the degree of incompatibility between the source and target systems and the setting, the more difficult the conversion.

Clearly, there will be an extra cost associated with moving programs from a COBOL-74 compiler to a COBOL-8x (this is the name sometimes used to refer to the proposed new standard) compiler insofar as there are incompatibilities between the two. This cost is the object of the quantitative analysis. It is also true, however, that in those cases involving the definition by the proposed revision of features that had been ambiguous or implementation-defined, there will be an associated benefit. This is because future conversions within the COBOL-8x standard will not be vulnerable to different implementation of these features.

Programs may be brought into conformance with COBOL-8x in the following ways:

1. Recoding for the sole purpose of conforming to the new standard
2. Recoding in conjunction with a system conversion to a new host system
3. Recoding in conjunction with normal software maintenance requiring recompilation
4. Reprogramming to meet new functional requirements of the application

In assessing the effect of the incompatibilities, it is useful to consider the federal COBOL inventory as a whole, and to ask how many of these programs will eventually be converted to COBOL-8x (as opposed simply to being left as-is until no longer needed), and in which of the four ways listed above this will occur. The list is ordered from greatest to least effect per program. At one extreme, if a program is converted purely for the sake of conformance, then the entire cost of conversion is attributable to the adoption of the new standard. At the other extreme, if a program is completely redesigned anyway, there is no measurable additional cost in seeing that it conforms to the standard. Midway between these cases would be bringing a program into conformance in conjunction with some other
form of updating, be that conversion or maintenance. While there is some extra effort involved, much of the conversion overhead (e.g., recompilation, retesting) is "free," in that it would be done even if the two versions of the standard were completely compatible. It is worth recalling that programs are recompiled rather frequently (at least once a year) for routine maintenance, and so there is plenty of opportunity for re-coding in category 3.

The cost effect is the additional effort expended in each of the above categories. Based on interviews with federal agencies, and also on a review of the transition process from COBOL-68 to COBOL-74, we conclude that very few, if any, conversions will be done merely for the sake of conformance.

Also, the previous experience in making the transition from COBOL-68 to COBOL-74 indicates that installations will continue to maintain the compiler for the previous version of the standard for a considerable time after introduction of the new version. We conclude, then, that the cost of achieving conformance in categories 1 and 4 is negligible, because virtually no conversion will be done in category 1 and there is no effect on conversion in category 4.

Measurable costs, then, are confined to categories 2 and 3, which we will treat together. The key questions are how many conversions will be done this way (as opposed to category 4 or not being done at all), and how much extra effort will be introduced by the incompatibilities.

The first question, about the percentage of programs to be converted, may be approached by noting some of the characteristics of the age of programs. The statistics on age allow us to formulate only a rough idea about the pattern of longevity for the current federal inventory. Note that the statistics are for the age of existing programs. This age distribution would directly reflect longevity only if we assumed that COBOL programs were being created at a constant rate over the past 15 years or so—clearly not the case. Nonetheless, almost any reasonable model one can develop that assumes an average age of six years for federal COBOL programs will yield a result no greater than 70–75% for the share of programs that will be converted to COBOL-8x over the next 10 years.

Next, we must consider the degree of extra effort entailed by the incompatibilities. For this analysis, we decided to use various parts of the Federal Conversion Software Center model. Its formulation is exclusively oriented to and based on federal ADP systems. Also it provides reasonable definitions of the conversion complexity classes and of average conversion cost per line of code by class. Through the use of this model, we can express in a precise way the intuitively natural notion that the costliness of a given incompatibility will depend strongly on how often the incompatibility is used (as measured by the sample) and how complex is the conversion that it entails. Based on this model, the cost of converting to COBOL-8x over the next 10 years is $17.9 million.

Sensitivity Analysis

The principal objective of a sensitivity analysis is to assess the degree of variation in the cost–benefit effect estimates generated by changes in the study assumptions, and to provide insight about the validity of the study findings (I for a summary). Therefore, we will discuss in greater depth those assumptions that are most subject to doubt and that affect the outcome most strongly.

Benefits

The benefits, as is typically the case for standards, are broad but shallow. Estimating the breadth (i.e., scope) of the benefit is relatively simple: Clearly, the effect extends throughout the use of COBOL in the federal government. The difficulty is in arriving at a reasonable estimate for the depth: How much good will the new standard do in an "average" federal agency? We have tried to be cautious in our estimates of the programming savings factor (PSF = 5%) and maintenance savings factor (MSF = 1%). The less precise of these is probably MSF. If we assume that MSF is 2%, instead of 1%, the maintenance benefit increases by $54 million. Such value is well within reason, but cannot be demonstrated with the available data.

Cost

We now examine those assumptions upon which depend the most likely cost estimate of $17.9 million. Clearly, the bulk of the cost stems from those incompatibilities that both occur frequently and force a more severe modification. There are four of these that deserve some individual comment:

1. Deleting MEMORY SIZE from the standard
2. Deleting ENTER from the standard
3. Defining the effect of EXIT PROGRAM on PERFORMs
4. Defining the order of evaluation of subscripts within PERFORMs

<table>
<thead>
<tr>
<th>Assumptions</th>
<th>Assume</th>
<th>Assume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSP = 1%</td>
<td>MSP = 2%</td>
</tr>
<tr>
<td>most likely</td>
<td>Benefit: 90.2</td>
<td>144.3</td>
</tr>
<tr>
<td>unchanged, 10%</td>
<td>Cost: -17.9</td>
<td>-17.9</td>
</tr>
<tr>
<td></td>
<td>Net: 72.3</td>
<td>126.4</td>
</tr>
<tr>
<td>benefit loss</td>
<td>Benefit: 81.2</td>
<td>129.9</td>
</tr>
<tr>
<td></td>
<td>Cost: -11.3</td>
<td>-11.3</td>
</tr>
<tr>
<td></td>
<td>Net: 69.9</td>
<td>118.6</td>
</tr>
<tr>
<td>Conversion of</td>
<td>Benefit: 90.2</td>
<td>144.3</td>
</tr>
<tr>
<td>50% of Program</td>
<td>Cost: -12.8</td>
<td>-12.8</td>
</tr>
<tr>
<td>Inventory</td>
<td>Net: 77.4</td>
<td>131.5</td>
</tr>
</tbody>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
Items 3 and 4 cannot reasonably be changed back to the original specification of COBOL-74. They simply define the semantics of two cases that were not described in COBOL-74.

For item number 1, the effect was completely dependent on the implementation in any event; almost all modern systems accept such information as part of their system control language. For item 2, it is technically feasible to keep the specifications of COBOL-74. If this were done, the cost estimate would shrink to $11.3 million. There would also be, however, an adverse effect on the benefit side. ENTER was deleted precisely because it encourages the development of the code that is error-prone and difficult to maintain. It would take only a 7% reduction of the benefits to cancel out the $6.6 million cost savings.

It is worth noting that in all four cases above, programs depending on the COBOL-74 specification were not guaranteed to be portable by that specification; all four changes are examples of taking aspects of the COBOL-74 standard that were ill-defined (purposely or not) to begin with, and either deleting the feature outright, or simply defining its effect. In none of these cases is a truly well-defined portable feature being affected.

The final issue is which policy federal agencies will adopt governing coding practices in the years leading up to the actual transition to a COBOL-8x implementation. We have somewhat pessimistically assumed that as new code replaces discarded programs, it will have the same degree of incompatibility. If, on the other hand, new code under development were monitored for conformance to COBOL-8x, then the effective percentage of code actually needing to undergo conversion would shrink from 70% to 50% within a few years. A figure of 50% implies conversion costs of $12.8 million.

FINDINGS AND RECOMMENDATIONS

This study shows that the effect of revising the COBOL standard as proposed should not be dramatic, either for good or ill. There is a real opportunity to improve certain features of the language, which should not be ignored, but the changes will hardly revolutionize COBOL programming in the federal sector. At the same time, there will be some problems created by incompatibility. These are not unusual, either in kind or in degree. Nor should it be surprising that the effect is relatively small; the proposed revision is just that: a revision of an existing standard—and not that markedly different from it.

It is important to put the projected costs and benefits into perspective. An effect of $100 million, spread out over 10 years, represents 0.3% of the salaries (unadjusted) of federal programmers over that same period. Concerning incompatibility, there was virtual consensus among the ADP personnel we interviewed that modifying source code was among the easier aspects of conversion. They had experienced far more difficulty with conversion of data and of job control code. Some agencies actually had to write their own input-output routines, rather than use those of the new system, because of data incompatibility. When asked what their biggest problem was, most answered, “the lack of documentation.” One interviewee characterized this as the problem of “portability of programs between programmers.”

There is no need to improve compatibility between the current and proposed versions of COBOL. While there are theoretical problems, the way in which COBOL is actually used in the federal government renders them relatively minor. The introduction of any further incompatibilities, however, should be subject to careful evaluation to ensure that their effects are no more adverse than those considered in this study.

The benefits of revising the COBOL standard are largely associated with the COBOL programs yet to be written. The costs are associated with those that already exist and depend on features unique to COBOL-74. Therefore, the sooner the standard becomes known and adopted, the better. The problems of incompatibility, real as they are, do not justify delaying the ongoing maintenance and improvement of the COBOL language.

REFERENCES


SUGGESTED READINGS