University of North Georgia
Nighthawks Open Institutional Repository

Honors Theses | Honors Program

Spring 2017

COBOL as a Modern Language

Charles Kiefer
University of North Georgia, charleskiefer01@gmail.com

Follow this and additional works at: https://digitalcommons.northgeorgia.edu/honors_theses

Part of the Programming Languages and Compilers Commons

Recommended Citation

Kiefer, Charles, "COBOL as a Modern Language" (2017). *Honors Theses*. 17.
https://digitalcommons.northgeorgia.edu/honors_theses/17

This Honors Thesis is brought to you for free and open access by the Honors Program at Nighthawks Open Institutional Repository. It has been accepted for inclusion in Honors Theses by an authorized administrator of Nighthawks Open Institutional Repository.

COBOL as a Modern Language

A Thesis Submitted to
the Faculty of the University of North Georgia
in Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in Computer Science
With Honors

Charles Kiefer
Spring 2017

Accepted by the Honors Faculty

of the University of North Georgia

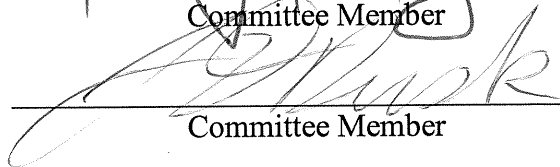
in partial fulfillment of the requirements for the title of

Honors Program Graduate

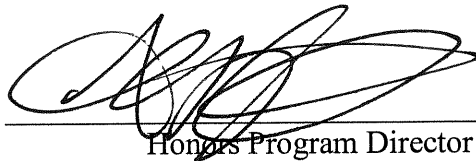
Thesis Committee:

Thesis Chair

Committee Member



Committee Member



Honors Program Director

ACKNOWLEDGEMENTS

My thanks go to my committee, Dr. Markus Hitz, Dr. Bryson Payne, and John-David Rusk for their valuable insight on what can be a difficult language to research, as well as to Dr. Stephen Smith for his assistance in the creation of this thesis.

Introduction

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. – Edsger Dijkstra [1]

This statement may be hyperbole, but Dijkstra's view on the language reflects underlying feelings about COBOL throughout the programming world. The language was created in 1959 to allow for interactivity between computation machines. [2] More than half a century later, COBOL is still used extensively in mainframes, computers designed for large-scale calculation and record processing. Numerous factors have contributed to the longevity of COBOL, including ease of use compared to its contemporaries and an upgrade to object orientation in the 1990s. [3]

This longevity has also contributed to problems with COBOL. The chief criticism is that it has become difficult to learn as other programming languages become more user-friendly. [4] COBOL software tends to be verbose, even for simple tasks. It's said that the average size of a COBOL program is 600 lines of code, whereas a Java program performing the same operation should be 30 lines or

---

1 Dijkstra, Edsger W. Selected Writings on Computing: A Personal Perspective. 1st ed. New York, NY: Springer New York, 1982. Print.

2 Wexelblat, Richard L. *History of Programming Languages*. 1st ed. New York, New York: Academic Press, 1981. 210.

3 Arranga, Edmund C., and Frank P. Coyle. *Object-Oriented COBOL*. New York, New York: SIGS Books & Multimedia, 1996. 15.

4 Volpano, D., & Dunsmore, H. (1981). Problems with COBOL--Some Empirical Evidence. Computer Science Technical Reports, 81(371). Retrieved from http://docs.lib.purdue.edu/cstech/300/

fewer. [5] Difficulties with the language will only increase as the workforce knowledgeable in COBOL's use retire.

History of COBOL

COBOL (Common Business Oriented Language) was commissioned in 1959 as a joint project between the United States Navy and several computing corporations such as IBM and RCA. [6] According to Jean Sammet, a member of the original COBOL design group, COBOL suffered from being in an intermediate period where companies had high expectations for language features and the technology required had not yet caught up. COBOL's scope had to be scaled down a more concise application for it to be completed. [7] Due to the many investors in its development, COBOL had a lengthy process of design by committee, which was cited as being both useful and dangerous, [8] since it meant more resources for production, but also could have led to an overdesigned final product. The committee established properties critical for the language:

1. Creation of four divisions in a program: PROCEDURE, DATA, ENVIRONMENT, and IDENTIFICATION.
2. Use of the English language throughout for commands, and data names, including allowance of 30 characters for data names.
3. Data could be organized into files that contained records, then subrecords, and fields within (sub)records.[9]

[5] Du Preez, Derek. *Banks will stick with COBOL because Java has performance issues*. Computerworld UK, June 13, 2013.

6 Wexelblat, Richard L. History of Programming Languages. 1st ed. New York, New York: Academic Press, 1981. 210.

7 Ibid, 212.

8 Beyer, Kurt. Grace Hopper and the Invention of the Information Age. Cambridge, Massachusetts: MIT Press, 2009. 285.

[9] Reilly, Edwin D. "COBOL." In *Concise Encyclopedia of Computer Science*, 104. Chichester, West Sussex, England: Wiley, 2004.

Compared to its predecessors which were made for processing mathematical formulae, COBOL had to be easy to use and understand. It also needed to be able to run on machinery that didn't have purpose-built hardware, which is circuitry with logic circuits created for a specific application. Betty Holberton, a member of the design committee, claims that this is what lead to COBOL's unexpected longevity when its lifecycle is compared against other languages of the time, even though she initially thought COBOL would be a temporary solution to the military's needs. [10]

As with many other languages of the time, COBOL began as a functional language, with extensive use of GO TO statements for logical control. [11] Only later, in COBOL-74 and COBOL-85, were structured programming paradigms available in COBOL. These include functions such as loops and function blocks that could be used repeatedly.

During COBOL's peak as business's most used programming language, [12] the concept of object orientation became the new favored programming style. Object orientation allows for logic units of data, or objects, to be reused throughout a program's structure, with the majority of the code performing operations on these objects. Languages like C and Java began to take market share away from COBOL with their simpler data manipulation controls.

---

[10] Wexelblat, Richard L. *History of Programming Languages*. 1st ed. New York, New York: Academic Press, 1981. 288.

[11] Sneed, H.m. "Extracting business logic from existing COBOL programs as a basis for redevelopment." *Proceedings of the 9th International Workshop on Program Comprehension*, 2001, 2. Accessed April 9, 2017. doi:10.1109/wpc.2001.921728.

[12] Philippakis, Andreas S., and Leonard J. Kazmier. *COBOL for Business Applications*. New York: McGraw-Hill, 1973.

COBOL's rapid obsolescence in the face of object-oriented languages caused it to lose support not only in business, but also in academia.[13] The loss of academic support for the language led to fewer programmers for the language being available, a dearth that is still visible today. [14] This programmer shortage lead to businesses growing even more eager to replace their COBOL frameworks.

The continued lifespan of COBOL was called into question at least by 2003 in a paper examining if businesses and academia should continue to support COBOL. [15] The paper notes that businesses are slowly halting development of new COBOL programs, and the most likely reason is that newer languages have essential features that COBOL is lacking; in particular, structured programming only became popular after COBOL's creation. Structured programming is the practice of keeping code well organized through the use of classes and objects. This paper was written in the transitional period to object-orientation for COBOL, and the authors admit that this could be a temporary phase until COBOL's features catch up to its peers. [16]

A trend in COBOL usage has become visible over the last decade. While COBOL has been on the decline in both academia and business since the 1970s, since 2010 it has experienced a resurgence. There are two categories of COBOL

---

[13] Dunn, Deborah L., and Dennis Lingerfelt. "Can visual basic replace COBOL? ...and should it?" *Journal of Computing Sciences in Colleges* 20, no. 4 (April 1, 2005): 214-20. Accessed April 10, 2017.

[14] Mitchell, Robert L. "Rebuilding the Legacy." Computerworld. April 24, 2006. Accessed April 10, 2017. http://www.computerworld.com/article/2554624/enterprise-applications/rebuilding-the-legacy.html.

[15] Carr, Donald, and Ronald J. Kizior. "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study." June 13, 2003. Accessed April 10, 2017. https://dl.microfocus.com/000/WP-20030613_tcm21-2774.pdf.

[16] Ibid, 16

development to be considered in this trend: Percentage of businesses that are utilizing COBOL in any capacity, and the total percentage of programming effort being expended on COBOL. The former indicates a stronger tendency toward maintenance of legacy software, whereas the latter shows potentially new software and systems being developed in the language.

Data on overall businesses usage of COBOL is available starting from 1999 in a Micro Focus survey. In 1999, 87% of businesses were using COBOL in any capacity. [17] Micro Focus performed a similar survey in 2003, showing that the number had dropped to 56%. [18] In 2006 and 2012 Mitchell conducted similar surveys showing a slight uptick to 62% [19] and then 64%, [20] respectively. These increased values could be a result of different surveying techniques or samples, but the increase in usage correlates with other data.

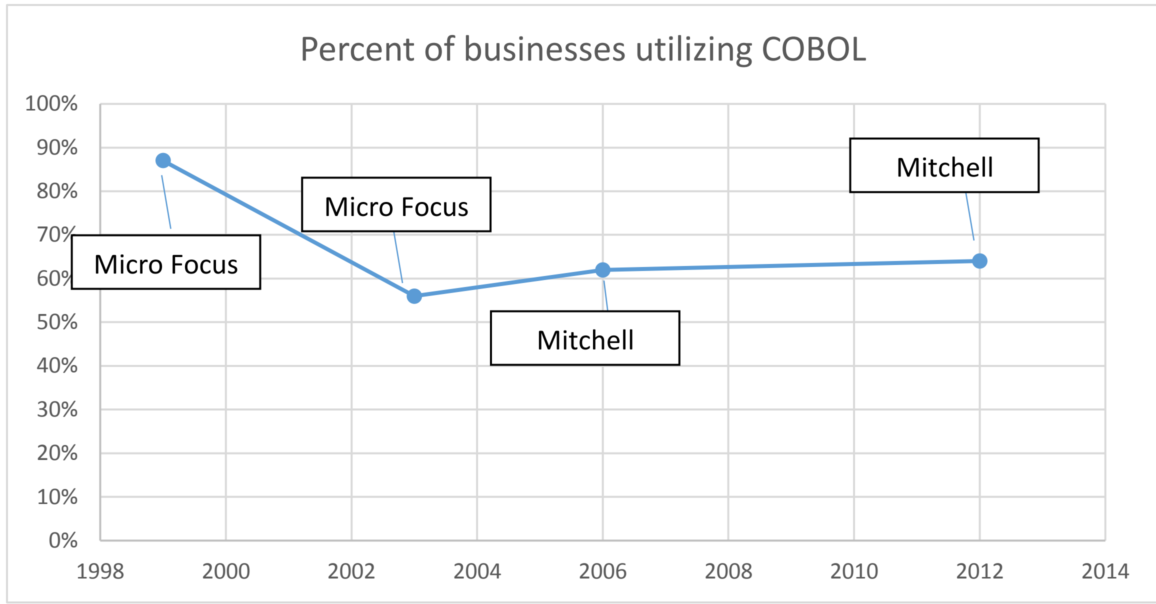
---

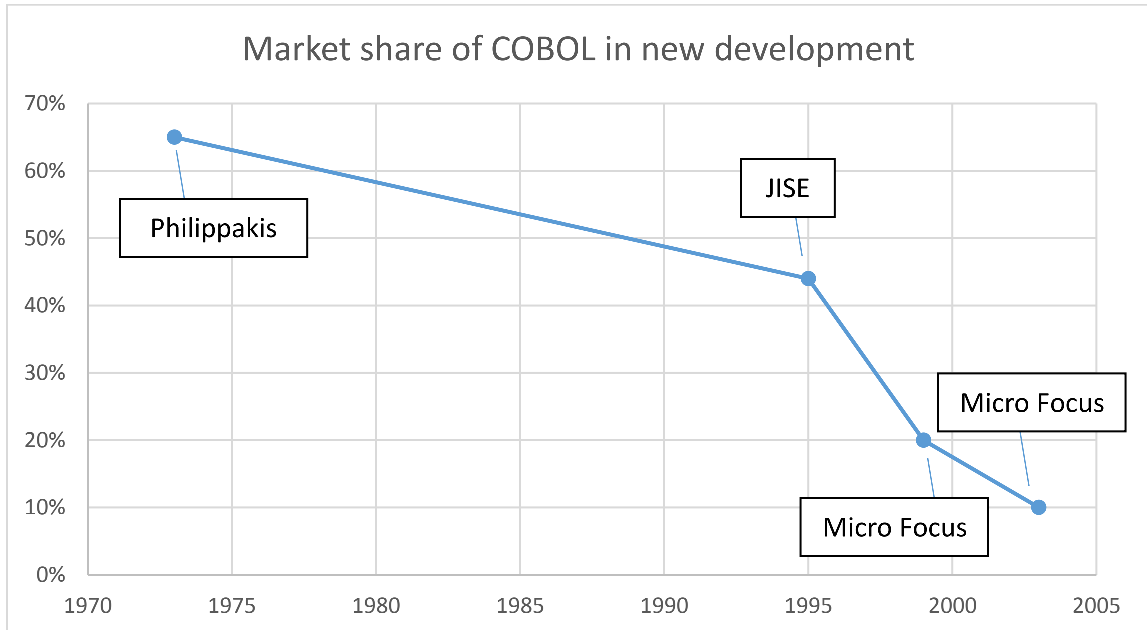
[17] Carr, Donald, and Ronald J. Kizior. "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study." June 13, 2003. Accessed April 10, 2017. https://dl.microfocus.com/000/WP-20030613_tcm21-2774.pdf.

[18] Ibid.

[19] Mitchell, Robert L. "COBOL: Not Dead Yet." Computerworld. October 04, 2006. Accessed April 10, 2017. http://www.computerworld.com/article/2554103/app-development/cobol--not-dead-yet.html.

[20] Mitchell, Robert L. "Rebuilding the Legacy." Computerworld. April 24, 2006. Accessed April 10, 2017. http://www.computerworld.com/article/2554624/enterprise-applications/rebuilding-the-legacy.html.





While the number of businesses using COBOL has remained relatively stable since the 1990s, new development in the language has dropped dramatically. Philippakis found in 1973 that an estimated 60 to 70% of new software development was being

done in COBOL. [21] That value fell to 44% in 1995, [22] then 20% in 1999, [23] and finally 10% in 2003. [24] No further studies have been conducted for this particular data point, possibly because any remaining new COBOL development is statistically insignificant. From this trend, it can be concluded that while legacy COBOL systems have remained intact, very few businesses find it worthwhile to do any new development in COBOL.

Possibly the most complete usage history of COBOL can be seen in the TIOBE index, which tracks programming languages through search engine hit counts. This indicates only search-based popularity of a given language, not usage percent or new development efforts. Data on COBOL is available beginning in 2001, giving it a 1.6% market share when compared to all other relevant programming languages. This market share follows the same trend that surveys have indicated, dropping steadily to a nadir of 0.3% in 2011. However, it began to climb in rank again after 2012, peaking at 1.3%. [25] This trend may continue.

---

21 Philippakis, Andreas S., and Leonard J. Kazmier. COBOL for Business Applications. New York: McGraw-Hill, 1973.

22 Gotwals, John, and Carlin Smith. "Restructuring Programming Instruction in The Computer Information Systems Curriculum: One Department's Approach". Journal of Information Systems Education 7.2 (1995): 68. Print.

23 Carr, Donald, and Ronald J. Kizior. "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study." June 13, 2003. Accessed April 10, 2017. https://dl.microfocus.com/000/WP-20030613_tcm21-2774.pdf.

24 Ibid.

25 "COBOL | TIOBE - The Software Quality Company". Tiobe.com. 2017. Web. 10 Apr. 2017. https://www.tiobe.com/tiobe-index/cobol/.

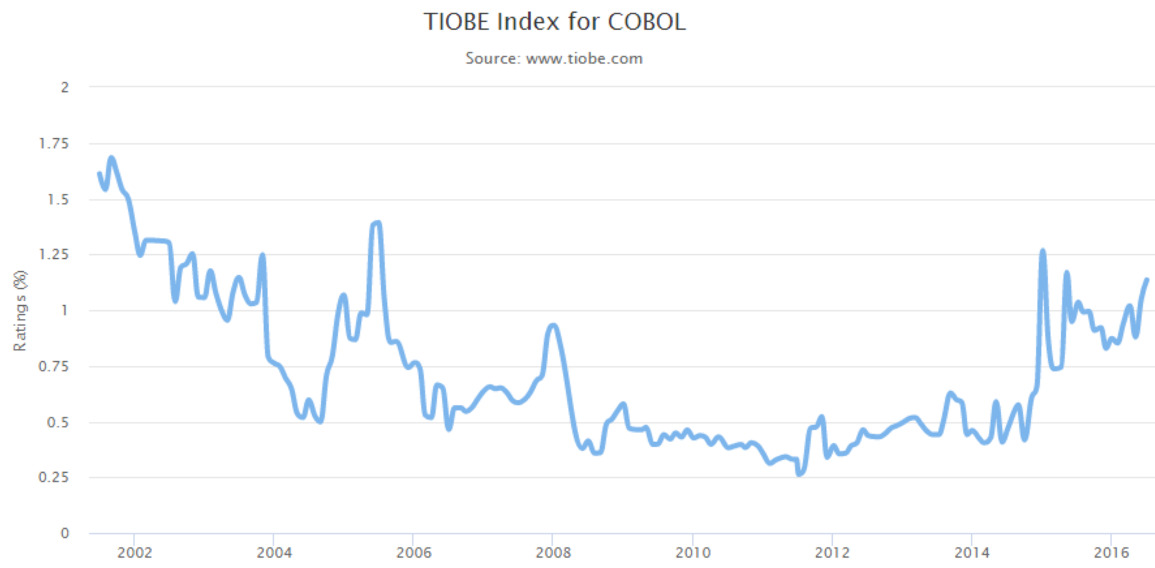


Figure 1 "COBOL | TIOBE - The Software Quality Company". Tiobe.com. 2017. Web. 10 Apr. 2017. https://www.tiobe.com/tiobe-index/cobol/.

The underlying issue is a lack of people skilled in COBOL. The source of this may be that COBOL is no longer being taught in academia as a beginner's language. The curriculum change started around 1995, when doubt arose as to COBOL's longevity, and at that time C was the preferred alternative. [26] By 2003 COBOL's demise was considered by educators to be a certainty, and those teaching had to identify a new introductory language. [27] COBOL was used first because it was expected to be a permanently useful skill for a programmer, and the new language had to have that same trait. In 2003 it was suggested that while COBOL could be retained for teaching in upper-level courses, Java should become the new

[26] Gotwals, John, and Carlin Smith. "Restructuring Programming Instruction in the Computer Information Systems Curriculum: One Department's Approach". Journal of Information Systems Education 7.2 (1995): 68. Print.

[27] Haney, John. "Something Lost - Something Gained: From COBOL to Java to C# in Intermediate Programming Courses". *Journal of Computing Sciences in Colleges* 19.1 (2003): 227-234. Print.

introductory language. [28] A 2005 paper came to the same conclusion, although it recommends Visual Basic .NET, and states that their choice could prove to be lacking in versatility as an introductory programming language. [29]

It is possible that academia is following this same usage trend as business in which COBOL becomes more widespread since the early 2010s. Numerous times, institutions with a computer science department have been surveyed. In 1999, COBOL was offered at 90% of these institutions. [30] A slight drop to 83% followed in 2003, [31] followed by a falloff to 27% by 2013. [32] Micro Focus's 2013 survey of academic institutions also revealed the relative rate at which COBOL developers are graduating compared to other specialties, finding that the number of COBOL courses in American college is on the rise as companies such as IBM seek to replace retiring COBOL engineers. [33]

A new study has now been conducted of 413 higher education institutions in the United States. Universities were first filtered to those with an existing computer science program. Availability of COBOL courses was determined through searches of each university's publicly available course catalog for COBOL

---

[28] Haney, John. "Something Lost - Something Gained: From COBOL To Java To C# In Intermediate Programming Courses". *Journal of Computing Sciences in Colleges* 19.1 (2003): 227-234. Print.

[29] Dunn, Deborah L., and Dennis Lingerfelt. "Can visual basic replace COBOL? ...and should it?" *Journal of Computing Sciences in Colleges* 20, no. 4 (April 1, 2005): 214-20. Accessed April 10, 2017.

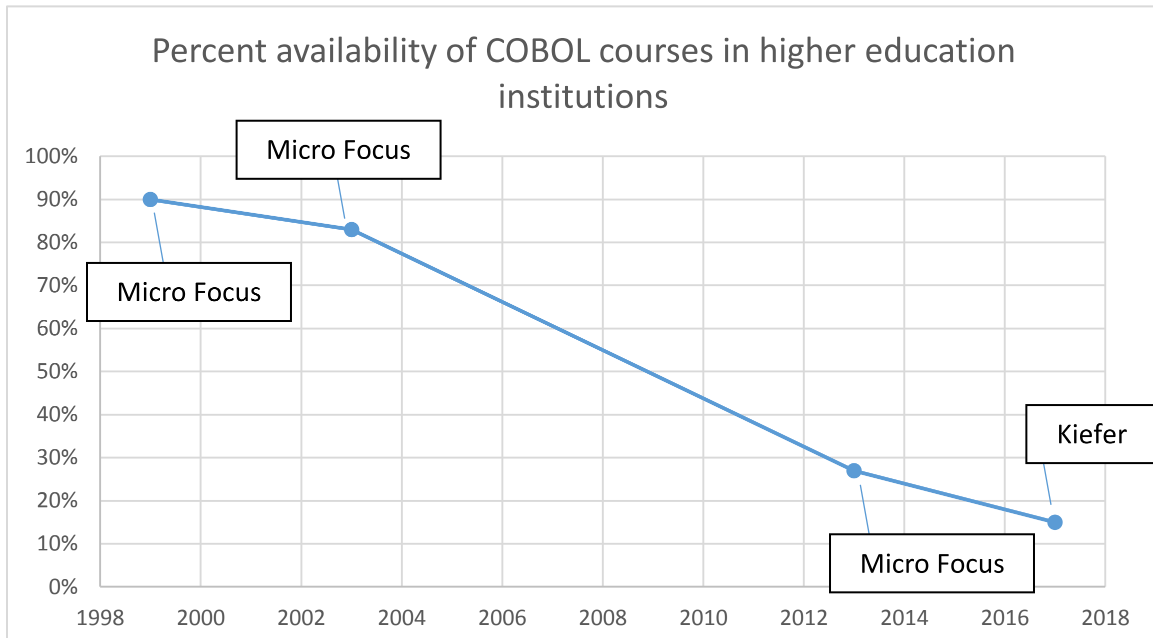
[30] Carr, Donald, and Ronald J. Kizior. "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study." June 13, 2003. Accessed April 10, 2017. https://dl.microfocus.com/000/WP-20030613_tcm21-2774.pdf.

[31] Ibid.

[32] "Academia Needs More Support To Tackle The IT Skills Gap | Micro Focus". *Microfocus.com*. N.p., 2013. Web. 10 Apr. 2017.

[33] Ibid.

courses or courses that use COBOL as the primary language. Of the 413, 61 were found to have an active COBOL course available. This represents a further decrease in 2017 to COBOL being available in approximately 15% of higher education institutions.



IBM is taking its own initiative to solve the workforce issue through sponsorships of institutions to grow existing COBOL programs and begin new ones. [34] The IBM Academic Initiative program includes a drive to teach mainframes technology in universities. The website lists eighty-two universities taking part in the program, though not all are teaching COBOL-related material. [35] Other corporations are making similar efforts, resulting in programs such as the Tennessee University COBOL training program. [36]

---

[34] "IBM Academic Initiative - Enterprise Systems". *Enterprise.waltoncollege.uark.edu*. Web. 10 Apr. 2017.

[35] "Mainframe Schools". *Mainframes.com*. Web. 10 Apr. 2017.

[36] McGee, Jamie. "Tennessee State University Offers COBOL Bootcamp". *The Tennessean*. N.p., 2016. Web. 10 Apr. 2017.

Benchmarking

One common method of comparing programming languages is by direct benchmarking. A representative program is written in multiple languages and run using each language's compiler. The running times for each language make for an easy and visibly distinct comparison.

No standardized benchmarking tests of COBOL could be found, so a simple set of benchmarking tests were conducted. Java, C#, and Python were selected to benchmark against COBOL based on their similar real-world application in locations where COBOL may be used.[37] An n-body simulation program, rewritten in COBOL with the same logical flow, was used to test each language.[38] N-body simulations conduct a large number of mathematical simulations based on the desired number of iterations. Generally, more iterations of the program will result in a linear increase in running time. The full source of the COBOL program can be found in Appendix A.

The tests were conducted on a system with an i7 4970k CPU with no overclocking at 4.00 GHz and 26 GB DDR3 RAM. The operating system was Windows 7 Home Edition. The COBOL and C# programs were compiled to executables using the GnuCOBOL compiler and the Visual C# Express compiler, respectively. Java and Python used just-in-time (JIT) compilation through the Java Eclipse Neon IDE and Thonny IDE, respectively. Times for the programs

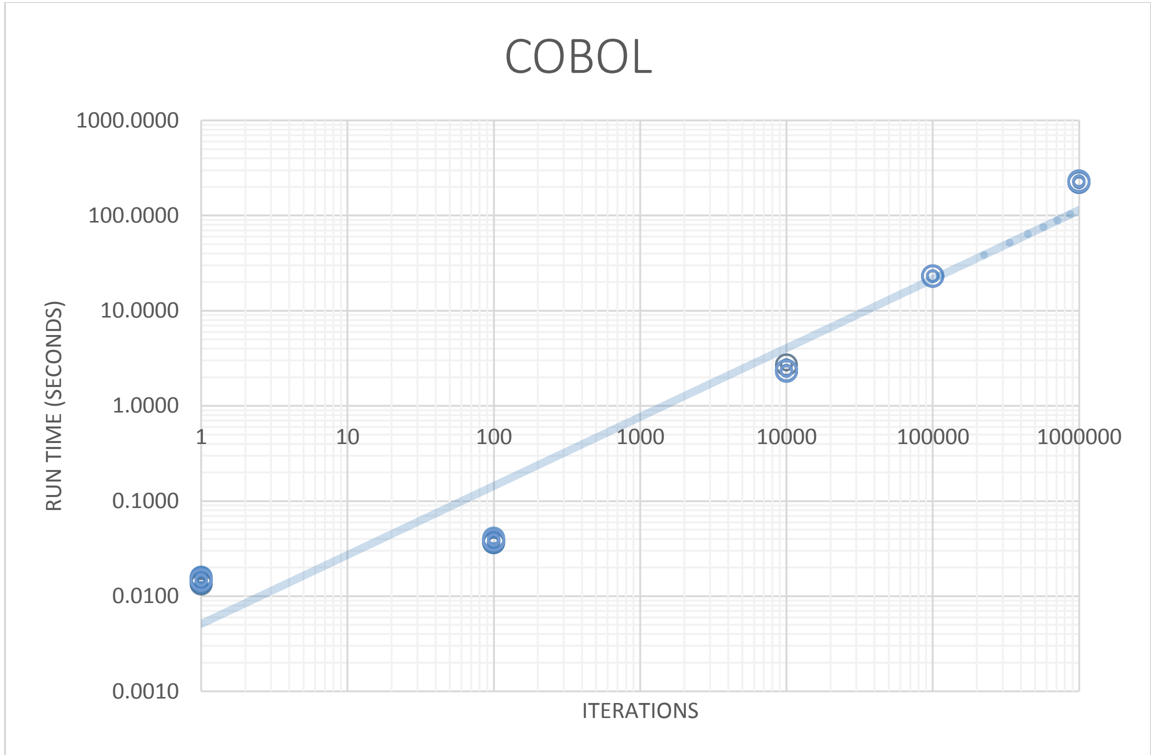
---

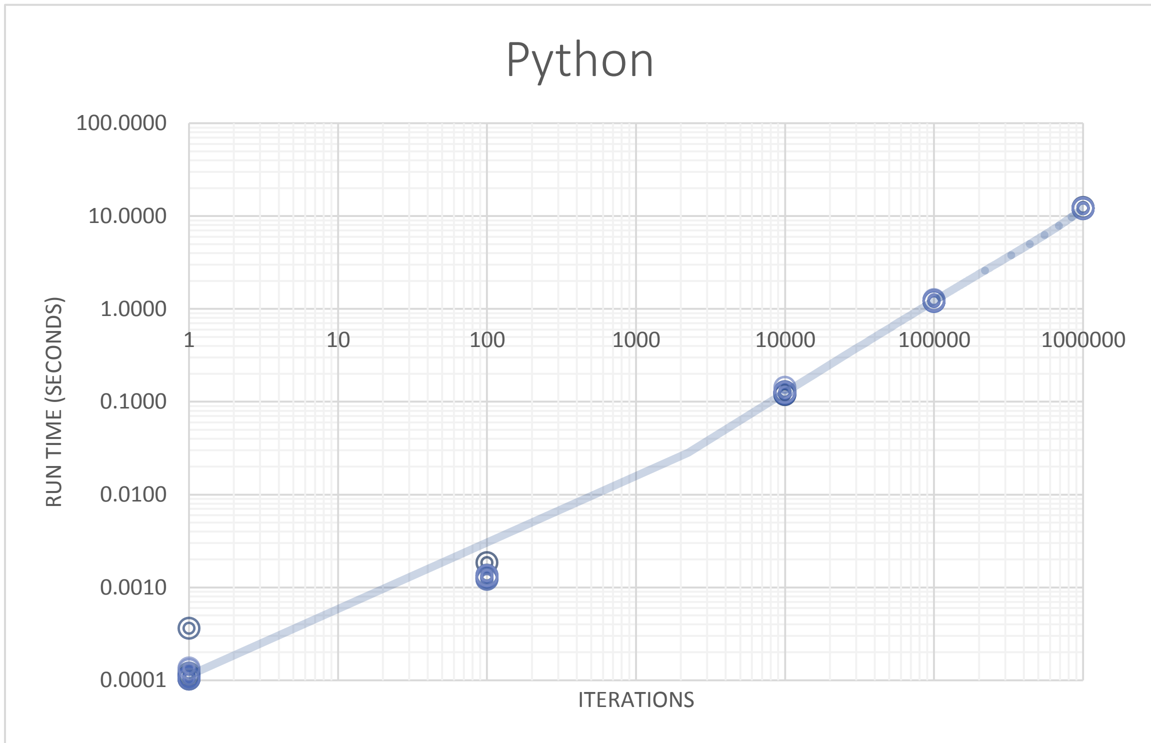
[37] Al-Qahtani, Sultan S., Rafik Arif, Luis F. Guzman, Adrien Tevoedjre, and Pawel Pietrzynski. "Comparing Selected Criteria of Programming Languages Java, PHP, C, Perl, Haskell, AspectJ, Ruby, COBOL, Bash Scripts and Scheme." *Concordia University*, August 20, 2010. Accessed February 13, 2017.

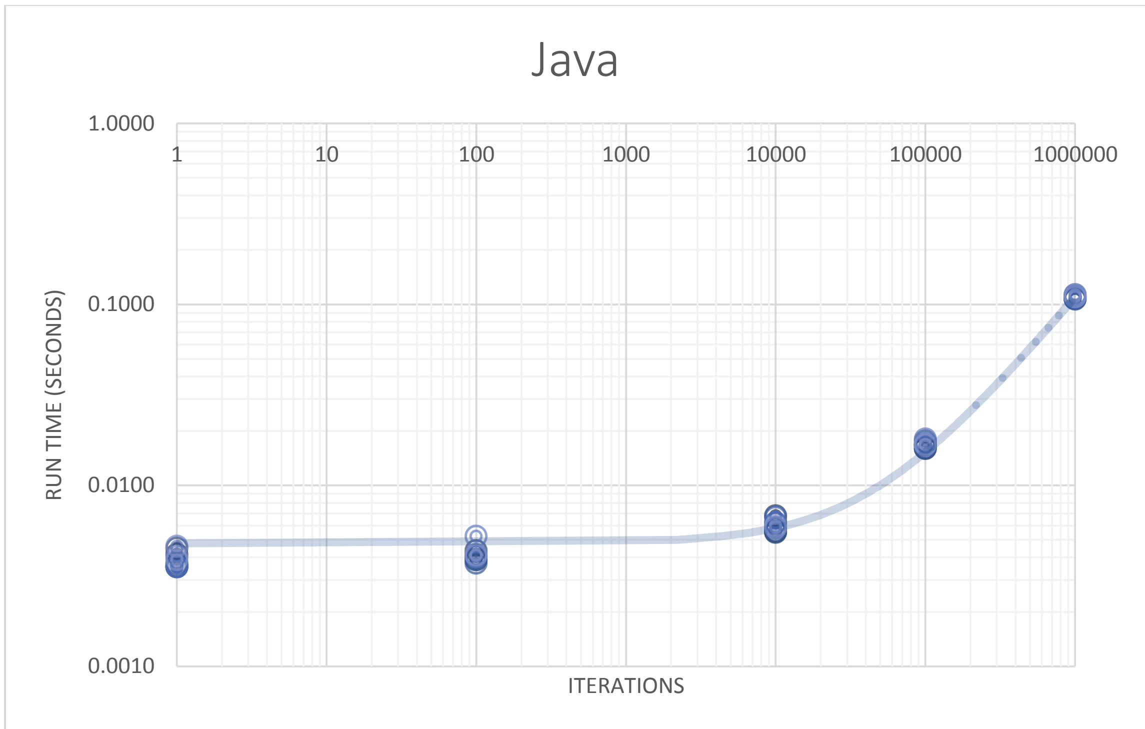
[38] Bagley, Doug, Brent Fulgham, and Isaac Gouy. "The Computer Language Benchmarks Game." The Computer Language Benchmarks Game. Accessed February 13, 2017. https://benchmarksgame.alioth.debian.org/.

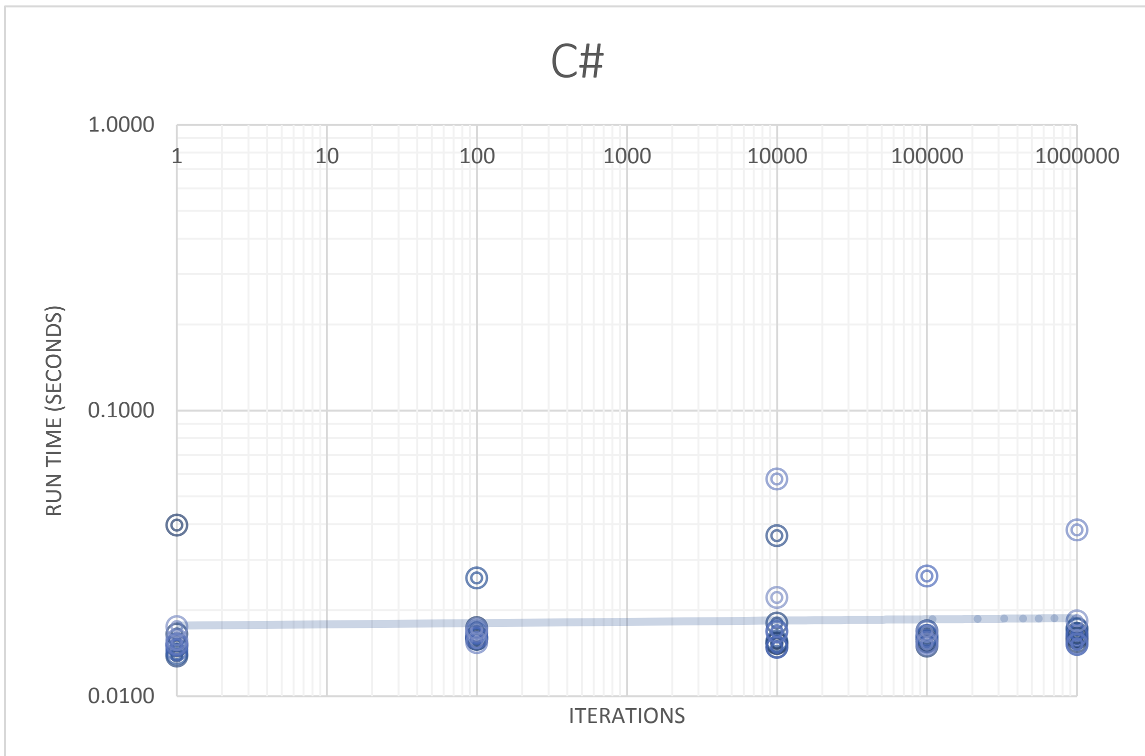
compiled to executables (C# and COBOL) were collected using Windows PowerShell's measure-command function. For Java and Python, basic timing commands were added to the logic to output run time as well as existing outputs.

Data was collected for 1, 100, 10,000, 100,000, and 1,000,000 iterations. Each program was run ten times for each iteration count, and an average time was calculated from those runs. Both time and iterations are shown in logarithmic form on the charts.


COBOL
1000.0000
100.0000
10.0000
1.0000
0.1000
0.0100
0.0010
RUN TIME (SECONDS)
1
10
100
1000
10000
100000
1000000
ITERATIONS


Python
100.0000
10.0000
1.0000
0.1000
0.0100
0.0010
0.0001
RUN TIME (SECONDS)
1
10
100
1000
10000
100000
1000000
ITERATIONS

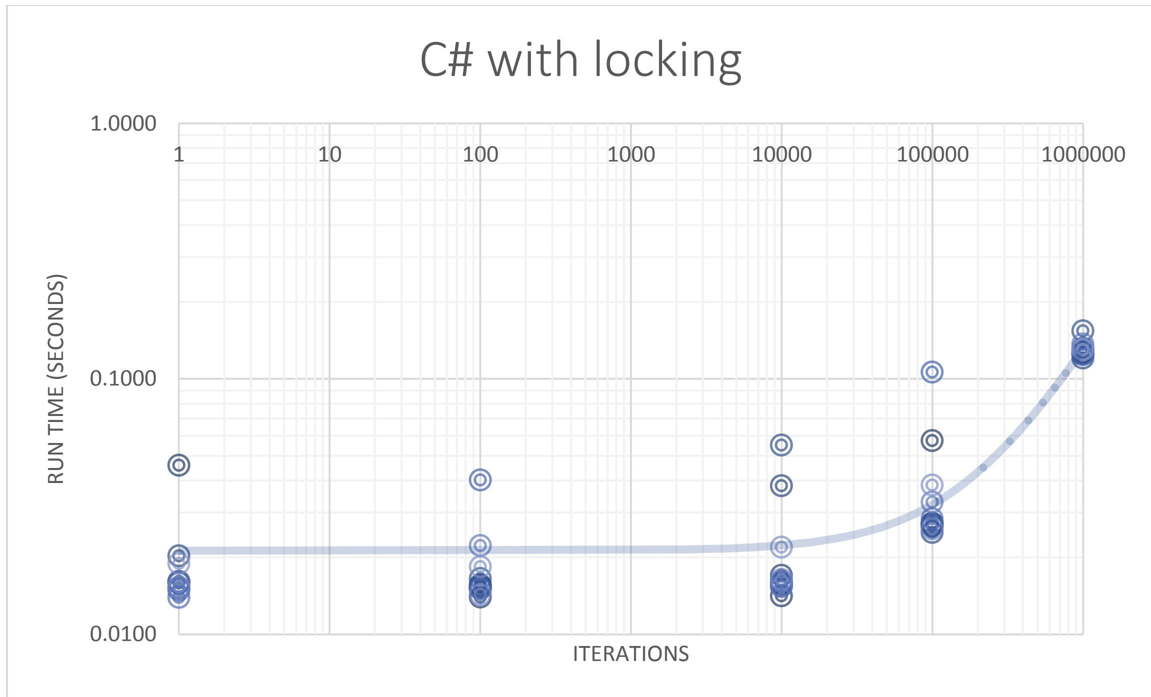

Java
1.0000
0.1000
0.0100
0.0010
RUN TIME (SECONDS)
1
10
100
1000
10000
100000
1000000
ITERATIONS


C#
1.0000
0.1000
0.0100
RUN TIME (SECONDS)
1
10
100
1000
10000
100000
1000000
ITERATIONS

The most obvious result of the tests is that COBOL shows a much longer running time than any of the other tested languages, for any given iteration count. It is possible that this time is due to overhead induced by the compiler (GnuCOBOL). The pattern of increasing runtime is most comparable to Java, in that both languages show a constant increase with the number of iterations at the same interval. It was theorized that COBOL's longer running times are partly due to the lack of multithreading, which is not directly supported by the language. [39] More modern languages frequently have multithreading support as part of the compiling process or JIT compilation.

To simulate single threaded execution on one of the other languages, a mutex lock was applied to the C# program. A mutex lock allows only one thread to work on a section of code at a time. When the lock is placed over the entire program, the compiled result is effectively single-threaded.

[39] "IBM Knowledge Center." IBM Knowledge Center. October 24, 2014. Accessed February 07, 2017. https://www.ibm.com/support/knowledgecenter/en/SS6SG3_4.2.0/com.ibm.entcobol.doc_4.2/PGandLR/tasks/tpthr02.htm.



After adding locking, the C# program begins to show an increasing run time with more iterations in a similar manner to the other languages. The runs in C# also show a higher average deviation, likely due to the executable's assigned thread being already occupied with another task. COBOL's runs also show this pattern, although it is not as noticeable due to the longer overall run times.

A further pattern to notice in the graphs is that some languages exhibit a time floor, beyond which fewer iterations do not decrease run time further. Java and especially Python show a lower running time floor on execution than the others, most notably when run with a single iteration, in which case running time becomes lower than that of C#, which otherwise displays the best performance. This can be interpreted as a benefit of the JIT compilation technique. To finalize this data, more sample programs will need to be written in COBOL to match the existing programs for others. This data set can be extended by comparing the results here against previous benchmarks, including benchmarking COBOL

against FORTRAN,[40] FORTRAN against Java,[41] and Java against C++.[42] The combination of these tests gives a broad overview of the performance landscape.

Other Comparisons

A more practical manner by which to compare languages is a feature by feature review. Implementing a new system in a language that come with an exhaustive library of pre-built modules can save programmers time that would need to be spent implementing them. COBOL shows itself to be lacking in many of these regards, particularly concerning recent advanced in concepts such as encapsulation and proper exception handling. Much of this comparison process has been performed by Al-Qahtani et al. in 2015. [43] The investigation includes comparisons of functionality, support, and other critical aspects of a programming language.

Replacing COBOL

A gradient of options are available for handling legacy COBOL software, ranging from low impact and low cost to high impact and high cost. The first few options allow for preservation of the existing software without completely changing languages. This is the preferred method for avoiding downtime and

---

[40] Paul, Lois. "CDC's, DEC's Time-Sharing Called Most Cost-Effective by RDC Study." *Computerworld*, May 31, 1982, 31-32.

[41] Bull, J.M., L.A. Smith, L. Pottage, and R. Freeman. "Benchmarking Java against C and Fortran for Scientific Application." Proceedings of the 2001 Joint ACM-ISCOPE Conference on Java Grande, 2001, 97-105.

[42] Sangappa, Sudhir, K. Palaniappan, and Richard Tollerton. "Benchmarking Java against C/C++ for Interactive Scientific Visualization." Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande, 2002, 236.

[43] Al-Qahtani, Sultan S. et al. "Comparing Selected Criteria of Programming Languages Java, PHP, C++, Perl, Haskell, Aspectj, Ruby, COBOL, Bash Scripts and Scheme". *Arxiv.org*. N.p., 2010. Web. 10 Apr. 2017. https://arxiv.org/abs/1008.3434

simplifying the modernization process. A total rewrite could result in loss of service for customers, the avoidance of which is crucial for certain businesses. While upgrading the existing system will inevitably require downtime handling, unexpected issues should be relatively fewer compared to changing technology.

	Keep original code	**Replace code**
Less in-depth	Refactor existing code	Managed COBOL
More in-depth	COBOL-2002	Change languages

Refactor existing code

Sellink et al. have made recommendations on restructuring COBOL code without a version upgrade. [44] The primary focus involves the removal of GO TO commands in favor of PERFORM-based looping. While this does not directly impact performance or functionality, it makes the code far easier to maintain, which will result in fewer unforeseen bugs from changes. To speed up the process of conversion, they have written an automatic process for adapting the most important instances of the GOTO command (those which are called often during execution).

COBOL-2002

Refactoring alone may not be enough of an improvement to justify the investment. To compete with languages like Java and C++ that took the majority

[44] Sellink, Alex, Harry Sneed, and Chris Verhoef. "Restructuring Of COBOL/CICS Legacy Systems". *Science of Computer Programming* 45.2-3 (2002): 193-243. Web.

of COBOL's market share, the COBOL 2002 standard implemented concepts from both languages, the most important being object orientation and encapsulation. [45]

Being able to use object-based logic will alleviate a long-standing issue with COBOL, its verbosity. As seen in the n-body simulation benchmark program, the object-oriented languages can be more succinct in their data declarations, with simpler member access and cleaner logical patterns. Without object orientation, the COBOL program needed to declare arrays of each data member. Having variables as a series of arrays makes it difficult to delete a logical structure (e.g. a planet in the benchmark program) without severely impacting performance.

---

[45] "COBOL 2002 – The Good, the Bad, and the UGLY". 2005. Presentation.

N-body declaration in C#

```
class Body { public double x,
y, z, vx, vy, vz, mass; }

class NBodySystem
{
    Body Jupiter = new Body()
    {
        x =
4.84143144246472090e+00,
        y = -
1.16032004402742839e+00,
        z = -
1.03622044471123109e-01,
        vx =
1.66007664274403694e-03 *
DaysPeryear,
        vy =
7.69901118419740425e-03 *
DaysPeryear,
        vz = -
6.90460016972063023e-05 *
DaysPeryear,
        mass =
9.54791938424326609e-04 *
Solarmass,
    };
} [46]
```

N-body declaration in COBOL-85

```
DATA DIVISION.
WORKING-STORAGE SECTION.
  01 SYSTEM.
    05 BODIES OCCURS 5 TIMES.
       10 X COMP-1.
       10 Y COMP-1.
       10 Z COMP-1.
       10 VX COMP-1.
       10 VY COMP-1.
       10 VZ COMP-1.
       10 MASS COMP-2.

PROCEDURE DIVISION.
SETUP-PROCEDURE.
*> Array position (2) aligns
with Jupiter

  COMPUTE X(2) =
4.84143144246472090.
  COMPUTE Y(2) = -
1.16032004402742839.
  COMPUTE Z(2) = -
0.103622044471123109.
  COMPUTE VX(2) =
0.00166007664274403694 * DAYS-
PER-YEAR.
  COMPUTE VY(2) =
0.00769901118419740425 * DAYS-
PER-YEAR.
  COMPUTE VZ(2) = -
0.0000690460016972063023 *
DAYS-PER-YEAR.
  COMPUTE MASS(2) =
0.000954791938424326609 *
SOLAR-MASS.
```

[46] Bagley, Doug, Brent Fulgham, and Isaac Gouy. "The Computer Language Benchmarks Game." The Computer Language Benchmarks Game. Accessed February 13, 2017. https://benchmarksgame.alioth.debian.org/.

N-body declaration in COBOL-2002

```
IDENTIFICATION DIVISION.
CLASS-ID. BODY
   DATA IS PROTECTED
   INHERITS FROM BASE.
DATA DIVISION.
WORKING STORAGE SECTION.
   CLASS-OBJECT.
      *> Functions to instantiate new & set variables, used later
   END CLASS-OBJECT.
   OBJECT.
       OBJECT-STORAGE SECTION.
       5 X COMP-1.
       5 Y COMP-1.
       5 Z COMP-1.
       5 VX COMP-1.
       5 VY COMP-1.
       5 VZ COMP-1.
       5 MASS COMP-2.
   END OBJECT.
   END CLASS BODY.
   01 JUPITER OBJECT REFERENCE OF BODY.

PROCEDURE DIVISION.
SETUP-PROCEDURE.
INVOKE BODY "NEW" RETURNING JUPITER.
```

While more boilerplate code is required to scaffold an object class, the procedure section of the code becomes cleaner when handling objects instead of variable arrays.

Encapsulation is the concept of preventing one portion of code from accessing, manipulating, or modifying other code segments. COBOL lacks this entirely. Instead, all variables are publicly accessible and instantiated in the data declaration section at the start of the program. This can allow for variables to be altered before their intended usage point later in the logic, causing logical errors and incorrect output. This issue is relieved somewhat by the addition of a local storage section, available in the COBOL 2002 release. However, compilers for

COBOL 2002 are not easily available for operating systems outside of the UNIX family, requiring more changes than to just the core COBOL framework.

Managed COBOL

Managed COBOL is a branch of COBOL created by Micro Focus to bridge the gap between COBOL, .NET, and Java. [47] It features extensions to connect to a JVM or .NET framework, solving the issue of COBOL's lack of library support. Most existing COBOL code can be imported directly to Managed COBOL without issue, unless certain incompatible features have been used. .NET COBOL works by combining the framework's COBOL, C#, and Visual Basic code into an intermediate language, then, at runtime, that language is adapted into a single native code language. The same applies for JVM COBOL, with just-in-time compilation to Java bytecode.

[47] "Micro Focus Documentation". *Documentation.microfocus.com*. Web. 10 Apr. 2017.

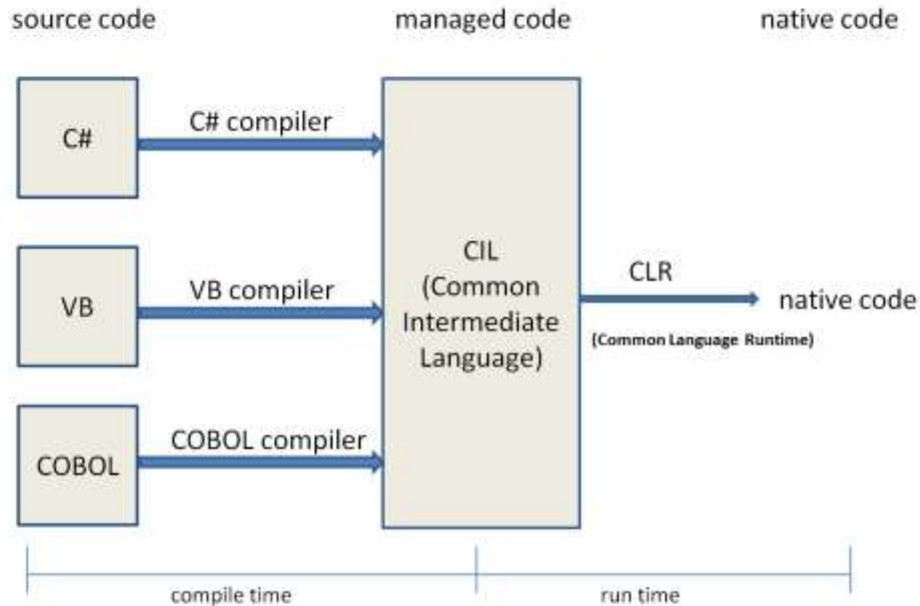


Figure 2 Willis, Paula. "Managed COBOL - An Overview". Micro Focus Community. 2012. Web. 10 Apr. 2017.

Managed COBOL gets the framework access to the vast libraries of .NET and Java, and allows the existing COBOL framework to easily interact with utilities written in either language. Because these utilities can now be interpreted through the COBOL syntax, vital features such as exception handling, dependency injection, and native SQL calls also become available. These should then be integrated with the existing framework for reliability and performance.

Change Languages

The most extreme method for handling a legacy COBOL framework is to entirely uproot and replace the language with something else. The potential for downtime and software issues is the highest here, but completely replacing COBOL will thoroughly solve the legacy software issues COBOL presents. The process can

be done entirely by hand, through an automated workflow, or a combination of both.

To rewrite the code by hand, it is best to first discern the business logic patterns used by the COBOL program so it can be recreated in a different language. Sneed has outlined the process. [48] Sneed's focus project was to reengineer the COBOL framework of a banking system, and he chose to completely rewrite the system in one written with object-orientation in mind. Sneed's method of repeatedly breaking down the code into smaller logical pieces is applicable to a large variety of COBOL systems.

An automated process can also be applied to convert the code to a different language. Tinetti et. al have attempted this on Fortran legacy code with the intent of adding multithreading and parallelization support. [49] They place extra weight on ensuring minimal downtime during the upgrade, and outline a five step cycle to do this:

[48] Sneed, H.m. "Extracting business logic from existing COBOL programs as a basis for redevelopment." *Proceedings of the 9th International Workshop on Program Comprehension*, 2001, 2. Accessed April 9, 2017. doi:10.1109/wpc.2001.921728.

[49] Tinetti, Fernando G., Mariano Méndez, and Armando De Giusti. "Restructuring Fortran Legacy Applications for Parallel Computing In Multiprocessors". *The Journal of Supercomputing* 64.2 (2013): 638-659. Web.

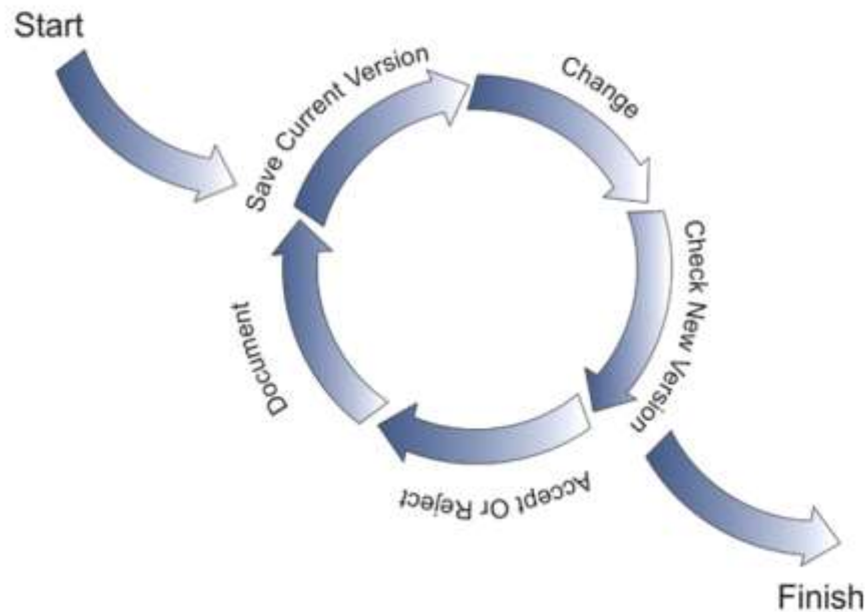


Figure 3 Tinetti, Fernando G., Mariano Méndez, and Armando De Giusti. "Restructuring Fortran Legacy Applications for Parallel Computing In Multiprocessors". The Journal of Supercomputing 64.2 (2013): 638-659. Web.

Their recommendations for programs capable of converting FORTRAN to other languages are inapplicable for COBOL, but numerous applications for this purpose can be found. [50]

Conclusions and Further Work

The decision for which of these approaches should depend on the business use of the COBOL application. If the codebase is relatively stable and free of issues, simply refactoring for performance may be sufficient. Other cases can require as much as a full replacement of all COBOL code. As new programs are not being written in COBOL compared to decades prior, the use rate of COBOL is expected to drop even further. Therefore, it will make the most business sense to handle

[50] "Convert COBOL To C++, CPP With COB2CPP Translator Converter." *Mpsinc.com*. Web. 10 Apr. 2017. http://www.mpsinc.com/cob2cpp.html

COBOL issues immediately rather than delaying until no programmers are available.

Further research must be performed into the exact circumstances of a business facing this dilemma with COBOL. Individual interviews would give much more detail into why a business would choose to either replace or keep legacy COBOL software. Additionally, more up-to-date research could be useful on the percentage of businesses reliant on COBOL since the 2012 survey by Mitchell. While a survey was attempted to find this data, the number of responses were insufficient to make a conclusion.

Appendix A: N-body simulation COBOL source code

```
IDENTIFICATION DIVISION.
PROGRAM-ID. NBODY-COBOL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
01 CONSTANTS.
05 RUN-TIMES        PIC 9(7)        VALUE 100000.
05 PI               COMP-1     VALUE 3.141592653589793.
05 DAYS-PER-YEAR    PIC 999V99   VALUE 365.24.
05 DT               PIC 9V99        VALUE 0.01.
05 SOLAR-MASS       COMP-1.
01 SYSTEM.
*> 1: Sun, 2: Jupiter, 3: Saturn, 4: Uranus, 5: Neptune
05 BODIES OCCURS 5 TIMES.
10 X     COMP-1.
10 Y     COMP-1.
10 Z     COMP-1.
10 VX    COMP-1.
10 VY    COMP-1.
10 VZ    COMP-1.
10 MASS  COMP-2.
01 TEMP-VARS.
05 I      PIC 9(7) VALUE 1.
05 J      PIC 9(7) VALUE 1.
05 PX     COMP-1.
05 PY     COMP-1.
05 PZ     COMP-1.
05 ENERGY COMP-1.
05 MAG                       PIC 9V9(30).
05 DISTANCE     COMP-1.
05 DSQUARED     COMP-1.
05 AX           COMP-1.
05 AY           COMP-1.
05 AZ           COMP-1.

PROCEDURE DIVISION.
SETUP-PROCEDURE.
*> Globals
COMPUTE SOLAR-MASS = PI * PI * 4.

*> Sun
COMPUTE X(1) = 0.
COMPUTE Y(1) = 0.
COMPUTE Z(1) = 0.
COMPUTE VX(1) = 0.
COMPUTE VY(1) = 0.
COMPUTE VZ(1) = 0.
COMPUTE MASS(1) = SOLAR-MASS.

*> Jupiter
COMPUTE X(2) = 4.84143144246472090.
COMPUTE Y(2) = -1.16032004402742839.
COMPUTE Z(2) = -0.103622044471123109.
COMPUTE VX(2) = 0.00166007664274403694 * DAYS-PER-YEAR.
COMPUTE VY(2) = 0.00769901118419740425 * DAYS-PER-YEAR.
COMPUTE VZ(2) = -0.0000690460016972063023 * DAYS-PER-YEAR.
COMPUTE MASS(2) = 0.000954791938424326609 * SOLAR-MASS.

*> Saturn
COMPUTE X(3) = 8.34336671824457987.
```

```
COMPUTE Y(3) = 4.12479856412430479.
COMPUTE Z(3) = -0.403523417114321381.
COMPUTE VX(3) = -0.00276742510726862411 * DAYS-PER-YEAR.
COMPUTE VY(3) = 0.00499852801234917238 * DAYS-PER-YEAR.
COMPUTE VZ(3) = 0.0000230417297573763929 * DAYS-PER-YEAR.
COMPUTE MASS(3) = 0.000285885980666130812 * SOLAR-MASS.

*> Uranus
COMPUTE X(4) = 12.8943695621391310.
COMPUTE Y(4) = -15.1111514016986312.
COMPUTE Z(4) = -0.223307578892655734.
COMPUTE VX(4) = 0.00296460137564761618 * DAYS-PER-YEAR.
COMPUTE VY(4) = 0.00237847173959480950 * DAYS-PER-YEAR.
COMPUTE VZ(4) = -0.0000296589568540237556 * DAYS-PER-YEAR.
COMPUTE MASS(4) = 0.0000436624404335156298 * SOLAR-MASS.

*> Neptune
COMPUTE X(5) = 15.3796971148509165.
COMPUTE Y(5) = -25.9193146099879641.
COMPUTE Z(5) = 0.179258772950371181.
COMPUTE VX(5) = 0.00268067772490389322 * DAYS-PER-YEAR.
COMPUTE VY(5) = 0.00162824170038242295 * DAYS-PER-YEAR.
COMPUTE VZ(5) = -0.0000951592254519715870 * DAYS-PER-YEAR.
COMPUTE MASS(5) = 0.0000515138902046611451 * SOLAR-MASS.

MAIN-PROCEDURE.
PERFORM OFFSET-MOMENTUM-PROCEDURE.

PERFORM CALCULATE-ENERGY-PROCEDURE.

DISPLAY ENERGY.

PERFORM ADVANCE-SYSTEM-PROCEDURE RUN-TIMES TIMES.

PERFORM CALCULATE-ENERGY-PROCEDURE.

DISPLAY ENERGY.

STOP RUN.

OFFSET-MOMENTUM-PROCEDURE.
MOVE 1 TO I.
PERFORM UNTIL I > 5
COMPUTE PX = PX + (VX(I) * MASS(I))
COMPUTE PY = PY + (VY(I) * MASS(I))
COMPUTE PZ = PZ + (VZ(I) * MASS(I))
ADD 1 TO I
END-PERFORM.

COMPUTE VX(1) = -1 * PX / SOLAR-MASS.
COMPUTE VY(1) = -1 * PY / SOLAR-MASS.
COMPUTE VZ(1) = -1 * PZ / SOLAR-MASS.

CALCULATE-ENERGY-PROCEDURE.
MOVE 0 TO ENERGY.
MOVE 1 TO I.
PERFORM UNTIL I > 5
COMPUTE ENERGY = ENERGY +
(0.5 * MASS(I)
* ((VX(I) ** 2)
+ (VY(I) ** 2)
+ (VZ(I) ** 2)))
```

```
COMPUTE J = I + 1
PERFORM UNTIL J > 5
COMPUTE AX = X(I) - X(J)
COMPUTE AY = Y(I) - Y(J)
COMPUTE AZ = Z(I) - Z(J)
COMPUTE DISTANCE =
(AX ** 2 + AY ** 2 + AZ ** 2) ** 0.5
COMPUTE ENERGY = ENERGY -
(MASS(I) * MASS(J)) / DISTANCE
ADD 1 TO J
END-PERFORM
ADD 1 TO I
END-PERFORM.

ADVANCE-SYSTEM-PROCEDURE.
MOVE 1 TO I.
PERFORM UNTIL I > 5
COMPUTE J = I + 1
PERFORM UNTIL J > 5
COMPUTE AX = X(I) - X(J)
COMPUTE AY = Y(I) - Y(J)
COMPUTE AZ = Z(I) - Z(J)

COMPUTE DSQUARED = AX**2 + AY**2 + AZ**2
COMPUTE DISTANCE = (DSQUARED) ** 0.5
COMPUTE MAG = DT / (DSQUARED * DISTANCE)

COMPUTE VX(I) = VX(I) - (AX * MASS(J) * MAG)
COMPUTE VY(I) = VY(I) - (AY * MASS(J) * MAG)
COMPUTE VZ(I) = VZ(I) - (AZ * MASS(J) * MAG)

COMPUTE VX(J) = VX(J) + (AX * MASS(I) * MAG)
COMPUTE VY(J) = VY(J) + (AY * MASS(I) * MAG)
COMPUTE VZ(J) = VZ(J) + (AZ * MASS(I) * MAG)
ADD 1 TO J
END-PERFORM
ADD 1 TO I
END-PERFORM.

MOVE 1 TO I.
PERFORM UNTIL I > 5
COMPUTE X(I) = X(I) + (DT * VX(I))
COMPUTE Y(I) = Y(I) + (DT * VY(I))
COMPUTE Z(I) = Z(I) + (DT * VZ(I))
ADD 1 TO I
END-PERFORM.

END PROGRAM NBODY-COBOL.
```

Bibliography

"Academia Needs More Support To Tackle The IT Skills Gap | Micro Focus". *Microfocus.com*. 2013. Web. 10 Apr. 2017.

"COBOL | TIOBE - The Software Quality Company". Tiobe.com. 2017. Web. 10 Apr. 2017. https://www.tiobe.com/tiobe-index/cobol/.

"COBOL 2002 – The Good, the Bad, and the UGLY". 2005. Presentation.

"Convert COBOL to C++, CPP with COB2CPP Translator Converter." *Mpsinc.com*. Web. 10 Apr. 2017. http://www.mpsinc.com/cob2cpp.html

"IBM Academic Initiative - Enterprise Systems". *Enterprise.waltoncollege.uark.edu*. Web. 10 Apr. 2017.

"IBM Knowledge Center." IBM Knowledge Center. October 24, 2014. Accessed February 07, 2017. https://www.ibm.com/support/knowledgecenter/en/SS6SG3_4.2.0/com.ibm.entcobol.doc_4.2/PGandLR/tasks/tpthr02.htm.

"Mainframe Schools". *Mainframes.com*. Web. 10 Apr. 2017.

"Micro Focus Documentation". *Documentation.microfocus.com*. Web. 10 Apr. 2017.

Al-Qahtani, Sultan S., Rafik Arif, Luis F. Guzman, Adrien Tevoedjre, and Pawel Pietrzynski. "Comparing Selected Criteria of Programming Languages Java, PHP, C, Perl, Haskell, AspectJ, Ruby, COBOL, Bash Scripts and Scheme." *Concordia University*, August 20, 2010. Accessed February 13, 2017.

Arranga, Edmund C., and Frank P. Coyle. *Object-Oriented COBOL*. New York, New York: SIGS Books & Multimedia, 1996. 15.

Bagley, Doug, Brent Fulgham, and Isaac Gouy. "The Computer Language Benchmarks Game." The Computer Language Benchmarks Game. Accessed February 13, 2017. https://benchmarksgame.alioth.debian.org/.

Beyer, Kurt. Grace Hopper and the Invention of the Information Age. Cambridge, Massachusetts: MIT Press, 2009. 285.

Bull, J.M., L.A. Smith, L. Pottage, and R. Freeman. "Benchmarking Java against C and Fortran for Scientific Application." Proceedings of the 2001 Joint ACM-ISCOPE Conference on Java Grande, 2001, 97-105.

Carr, Donald, and Ronald J. Kizior. "Continued Relevance of COBOL in Business and Academia: Current Situation and Comparison to the Year 2000 Study." June 13, 2003. Accessed April 10, 2017. https://dl.microfocus.com/000/WP-20030613_tcm21-2774.pdf.

Dijkstra, Edsger W. Selected Writings on Computing: A Personal Perspective. 1st ed. New York, NY: Springer New York, 1982. Print.

Du Preez, Derek. *Banks will stick with COBOL because Java has performance issues*. Computerworld UK, June 13, 2013.

Dunn, Deborah L., and Dennis Lingerfelt. "Can visual basic replace COBOL? ...and should it?" *Journal of Computing Sciences in Colleges* 20, no. 4 (April 1, 2005): 214-20. Accessed April 10, 2017.

Gotwals, John, and Carlin Smith. "Restructuring Programming Instruction in the Computer Information Systems Curriculum: One Department's Approach". Journal of Information Systems Education 7.2 (1995): 68. Print.

Haney, John. "Something Lost - Something Gained: From COBOL To Java To C# In Intermediate Programming Courses". *Journal of Computing Sciences in Colleges* 19.1 (2003): 227-234. Print.

McGee, Jamie. "Tennessee State University Offers COBOL Bootcamp". *The Tennessean*. 2016. Web. 10 Apr. 2017.

Mitchell, Robert L. "COBOL: Not Dead Yet." Computerworld. October 04, 2006. Accessed April 10, 2017. http://www.computerworld.com/article/2554103/app-development/cobol--not-dead-yet.html.

Mitchell, Robert L. "Rebuilding the Legacy." Computerworld. April 24, 2006. Accessed April 10, 2017. http://www.computerworld.com/article/2554624/enterprise-applications/rebuilding-the-legacy.html.

Paul, Lois. "CDC's, DEC's Time-Sharing Called Most Cost-Effective by RDC Study." *Computerworld*, May 31, 1982, 31-32.

Philippakis, Andreas S., and Leonard J. Kazmier. *COBOL for Business Applications*. New York: McGraw-Hill, 1973.

Reilly, Edwin D. "COBOL." In *Concise Encyclopedia of Computer Science*, 104. Chichester, West Sussex, England: Wiley, 2004.

Sangappa, Sudhir, K. Palaniappan, and Richard Tollerton. "Benchmarking Java against C/C++ for Interactive Scientific Visualization." Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande, 2002, 236.

Sellink, Alex, Harry Sneed, and Chris Verhoef. "Restructuring Of COBOL/CICS Legacy Systems". *Science of Computer Programming* 45.2-3 (2002): 193-243. Web.

Sneed, H.M. "Extracting business logic from existing COBOL programs as a basis for redevelopment." *Proceedings of the 9th International Workshop on Program Comprehension*, 2001, 2. Accessed April 9, 2017. doi:10.1109/wpc.2001.921728.

Tinetti, Fernando G., Mariano Méndez, and Armando De Giusti. "Restructuring Fortran Legacy Applications for Parallel Computing In Multiprocessors". *The Journal of Supercomputing* 64.2 (2013): 638-659. Web.

Volpano, D., & Dunsmore, H. (1981). Problems with COBOL--Some Empirical Evidence. Computer Science Technical Reports, 81(371). Retrieved from http://docs.lib.purdue.edu/cstech/300/

Wexelblat, Richard L. *History of Programming Languages*. 1st ed. New York, New York: Academic Press, 1981.