Astadia™ THE MAINFRAME TO CLOUD EXPERTS

WHITE PAPER

TRANSFORMING COBOL INTO C#

Notice

Astadia makes no warranty that the content of this document is timely or complete; or is free of omissions, inaccuracies, typographical errors, or other errors. All contents of this document, including but not limited to the text and images contained therein, are made available on an "as is" basis without any warranty, express or implied, of any kind, including the implied warranties of merchantability, title, non-infringement, quality, or fitness for any particular purpose.

Certain sections of this document may contain forward-looking statements that are based on product management's expectations, estimates, projections and assumptions. Words like "plans," "intends," "expects," "believes," "future," "estimates" and variations of these words and similar expressions are intended to identify forward-looking statements. These statements are not guarantees of future performance and involve certain risks and uncertainties, which are difficult to predict. Therefore, actual future results and trends may differ materially from what is forecast in forward-looking statements due to a variety of factors.

Trademarks

TestMatch, DataMatch, CodeTurn, DataTurn, and CobolBridge are trademarks of Astadia. These trademarks may not be used without the permission of Astadia. The absence of a product, company, or service name or logo from this list does not constitute a waiver of the trademark or other intellectual property rights of Astadia concerning that name or logo.

Other trademarks that appear in this document are used for identification purposes only and are the property of their respective owners. These marks may not be used without permission from these owners.

CONTENTS

TRANSFORMING COBOL INTO C#

How can businesses move safely and cost-effectively from COBOL to C#? This document explains the migration through automated code transformation, a mature approach being perfected by Astadia as an answer to this question.

The white paper explores key business drivers for making the step from COBOL to C# and how Astadia's tools make this transition possible. It will explain the added value that the Astadia migration offers over other approaches and demystify the migration process with a step-by-step look at the way COBOL can be transformed into working, maintainable C# code.

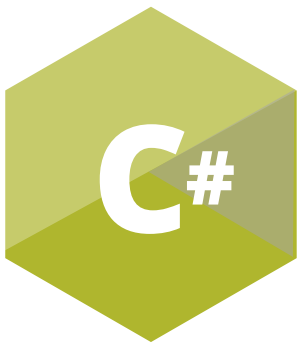
Automated Application Modernization

1. WHY TRANSFORM COBOL INTO C#?

1.1. Reasons to go to C#

There are many good reasons to make the move to C# from COBOL, but the following are the largest concerns for businesses:

- High (and continuously increasing) maintenance and runtime fees for the existing COBOL products
- Shrinking availability of COBOL developers and lack of interest in COBOL from young developers
- End-of-life scenarios for certain COBOL technologies
- Lack of application extensibility and interoperability with other, non-COBOL applications



C# offers answers to all of the above concerns:

- Support fees are negligible (or even non-existing depending on the choices made) when compared to COBOL.
- C# is one of the most widely-used programming languages today[1] and it is the language of choice for instruction in the IT programs of many schools.
- Documentation of the language, software libraries, and development tools are also available free of charge.
- C# and .NET are actively being developed by Microsoft and the product and its documentation are publicly available on the Internet (earlier versions were also official ISO/IEC and ECMA[2] standards).
- Interoperability with all Microsoft platforms and technologies is a given and portability to other platforms is being actively developed by Microsoft in the form of .NET Core[3]; the Mono[4] project also provides cross-platform capabilities.

Next to that, moving to C#/.NET also means:

- Enabling the use of a state-of-the-art IDE in the form of Visual Studio, with extensive debugging, refactoring, profiling and (unit)testing support
- Enabling the use of thousands of third-party libraries, covering almost all imaginable computing needs: UI-development, database interaction, mail/ftp/http/... communication, parsing, xml processing ,...
- Enabling the use of modern application architectures (multi-tier, SOA or micro-service based) and deployment techniques (cloud, docker, ...)

[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
[2] http://www.ecma-international.org/publications/standards/Ecma-334.htm
[3] https://docs.microsoft.com/en-us/dotnet/articles/core/index
[4] http://www.mono-project.com/

1.2. Approaches: Replacement, Rewrite or Automated Migration to C#

Once the need to move away from COBOL has been established, the next big question is: how to turn a large, mature COBOL code base into its C# equivalent?

Most organizations take one of three approaches:

Replacement by COTS software is the most cost-effective solution, but only practical if one can actually find 3rd party software that offers the same functionality as the existing COBOL application.

Rewriting on the other hand too often has led to huge costs and ultimately to project failures. According to a research note from Gartner[1] the cost for rewriting is between $6 and $26 per Line of Code (LOC), and accomplished at a rate of 160 LOC per day and per developer.

Extrapolated even to a moderate code base of 1M LOC, it is obvious that these will become very expensive, lengthy and risky projects.

Astadia advocates automated migration as the only realistic approach for large legacy COBOL applications:

It offers consistency
Since all code is translated by software, there can be no differences in quality and all existing functionality is kept as-is.

It offers speed and continuous improvement possibilities
A complete code base can be converted in a couple of hours, meaning that this process can be repeated as often as needed or wanted.

It offers a very large degree of testability
When keeping the converted application's functionality unchanged, the original application's behavior can effectively be used as a regression test (and running this regression test, too, can be automated).

It offers minimal interruption
Existing developers and IT departments can keep working on their daily tasks, including the ongoing maintenance of the COBOL code.

All the above, combined with a vast experience in executing migration projects, gives Astadia the confidence to offer projects with fixed duration and fixed price, equivalent functional behavior and equivalent performance.

Like most IT projects, a migration project is a complex undertaking, one that deserves the right amount of expertise and dedication. Astadia has developed project and product methodologies over the past decades to deal successfully with these challenges: we are happy to provide you with more information and case studies.

[1] Gartner Research Note "Forecasting the Worldwide IT Services Industry: 1999,1"

2. COBOL TO C#: ARCHITECTING A MIGRATION

This section focuses on the underlying design principles that any migration from COBOL to C# should take into account, and how Astadia has incorporated these in its toolset.

2.1. Keys to a Successful Migration

What are the keys to a successful migration project? A primary concern in any migration is how to validate the functional correctness of the migrated programs. In COBOL to C# migrations, this can be hard, especially if the COBOL programs are driving business-critical processes and they are being adapted to evolving user requirements while a migration is running at the same time.

Under these circumstances, migrations should target:

Equivalent program behavior and performance

For programs to be functionally correct, business users have to accept them. For large COBOL applications that are being actively maintained, users and developers have an existing and well-tried process to specify changes, develop, test and bring new releases into production. This familiar process should not be disturbed by migration efforts.

An important best practice in migrations is targeting functional and performance equivalence with the production release, and avoid whenever possible the introduction of functional extensions that don't exist in the COBOL application.

This approach makes it possible to leverage automated testing tools to reduce project costs, improve accuracy and move the project forward faster.

Automated iterative processes

To manage risk, the generally accepted process in migrations is no different from that of conventional software engineering: using agile, iterative processes that can periodically align the migration project to the latest versions of the COBOL programs undergoing the migration. Consistency and speed are also critical to a parallel iterative process. Using tools to automate the migration imposes rigorous consistency of transformation and brings the end goal within reach of stakeholders.

Developer confidence

There are many differences between COBOL (a structured, compiled, business-oriented language) and C# (an object-oriented, managed, multi-purpose language). COBOL developers that need to maintain the new C# programs will need to be confident in their ability to recognize the business rules and correctly implement and test the changes they make after the migration is completed. Meanwhile, new C# developers also need to feel comfortable with the migrated code to maintain it like any other regular C# program.

2.2. Key Traits of a Professional Software Conversion Tool

The key hallmarks of a professional language translation tool are the way it can strike a balance between three spectra of interest. Astadia's COBOL-to-C# conversion toolset offers a solution to each of these areas:

Customization and Consistency

Anyone who has written software in a team before knows any program can be written in a variety of ways. A professional software conversion tool will provide the means to apply customization options that suit the requirements of the customer. These options can be simple things, like naming conventions and the formatting of comments, or they can be more sophisticated, like the extent to which the tool will optimize structural patterns in the COBOL code.

At the same time, the tool should make it possible to manage such a configuration of customization options for a consistent application in an iterative process. These management facilities should also include the configuration of other aspects of the migration like the translation of scripts, screens, or databases.

Maintainability by COBOL and C# Developers

Consistent translation improves the maintainability of the generated C#, but for the C# code to be easy to work with for the original COBOL developers, it also needs to be based around simple design principles.

This means the tool should generate code that allows as much as possible a 1:1 relationship between the number of lines of COBOL code and the number of lines of C#, and reuse (wherever it is syntactically allowed) the names of identifiers appearing in code.

This will facilitate the recognition of business entities and rules in C# by the COBOL developer. At the same time, C# developers with limited exposure to COBOL should be able to pick up the programs and be optimally productive in the shortest possible timeframe. Simple design principles improve the understandability of the converted programs also by C# developers. Typical COBOL constructs that are unknown in C# (like DECLARATIVES or PERFORM) are transformed into their closest C#-style equivalents.

Functional Equivalence with COBOL and Full Support for the Target Platform

The basic requirement of a conversion tool is that the code it produces is 100% functionally equivalent (including side effects encountered at runtime) with the original COBOL code. At the same time, the code that is produced should enable full use of the richness of the target .NET platform.

This means some COBOL language syntax will be replaced with calls to .NET APIs. It also means that the code should be fully usable in Visual Studio and support execution in debug mode.

Last but not least, the converted programs must easily be integrated with newly written C# (and to a wider extend: .NET) programs and vice versa, hereby ensuring a solid basis for continuous improvement and further modernization.

2.3. Flexible Tools

2.3.1. Rationale

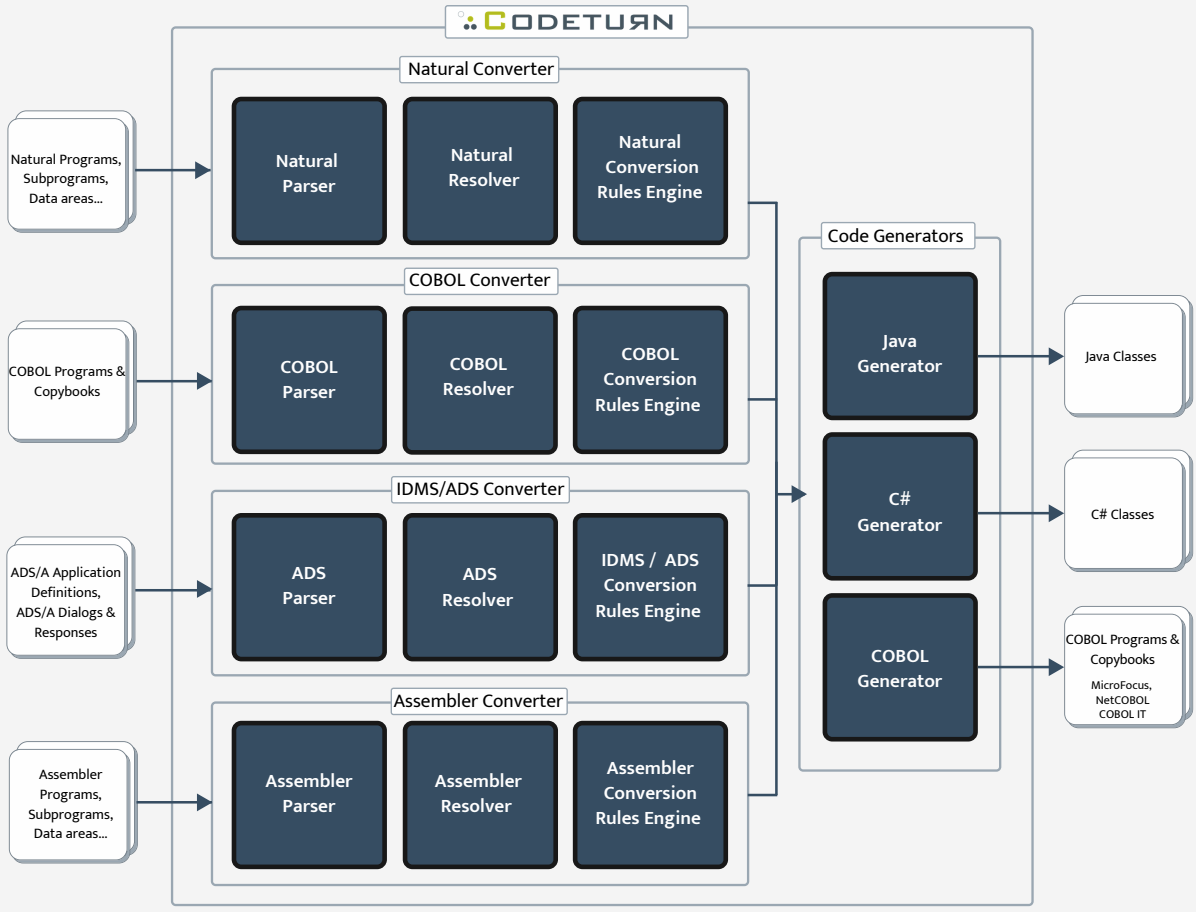
For transformations from one programming language to another, Astadia has built a set of tools collectively called CodeTurn. These tools share one important consideration: each customer is different, and each migration is different. Every organization has its own development and design standards, patterns and frameworks. Some prefer all data access in a separate layer, others choose for embedded data access.

When migrating from COBOL to C#, some organizations prefer to keep the resulting code relatively close to the original COBOL, for reasons of readability and maintainability by the existing developers. Other organizations will choose a more radical approach and prefer code that uses more Object-Oriented design patterns. In order to be able to accommodate these considerations, standards and frameworks the Astadia tools can be parameterized and customized for each project.

To address the stringent requirements listed above, Astadia has chosen a modular approach to building a migration toolset for all of its supported source technologies (COBOL, IDMS, Natural, ...).

For language to language transformations, such as from COBOL to OO, the architectural overview of the internals of CodeTurn looks like this:

- Parsers that support all COBOL dialect syntax, and also embedded languages (e.g. EXEC CICS or EXEC SQL) Resolvers that link together the AST[1] that is produced by the parsers with control-flow and data-usage information
- Conversion rules for all COBOL syntax, from single statements to complex patterns of code, as part of the COBOL to OO Converter
- Code generators for Java, C#, and various COBOL dialects enabling any desired coding style



2.3.2. Intelligent Transformation

To keep up with the design principle of a fully automated migration, transformation rules have been implemented in the COBOL to OO Tranformation module to cover all possible edge cases. From time to time this could lead to relatively complicated OO code. Therefore, the conversion tools also recognize coding patterns (by static code analysis) that indicate where simpler, more elegant code can be generated while still staying 100% functionally equivalent to the COBOL original.

A good example of this is the conversion of the COBOL GO TO statement. In its most general use a GO TO statement can implement a complex state-machine and translating this to C# leads again to a state-machine: the conversion engine has a baseline rule that produces this state-machine code. Oftentimes however, a GO TO is simply used as an early exit point (e.g. GO TO exit-paragraph), or as a way to avoid a certain block of code, or even to emulate a loop. In these circumstances, much more elegant code is generated as can be seen further down this document.

In a typical project, more than 95% of all GO TO statements are automatically converted to a more structured equivalent.

Static code analysis limits the optimization of the transformed code that can be done by CodeTurn. Application architects can however help to understand additional restrictions and coding patterns that exist, and based on this understanding, new conversion rules can be implemented. Good examples of where this is likely to help are again the use of GO TO, but also the use of code copybooks and ENTRY statements are typical candidates for such optimization.

2.3.2. Procedural vs. Object-Oriented Paradigm

Another example of the importance of flexible tools lies in the preference of some developers for Procedural-Style code, while others prefer to take maximum advantage of OO-concepts.

Astadia offers both variations, as can be seen in the following short COBOL code snippet:

COBOL	`SELECT DATFILE ASSIGN TO "test.dat"`
	
	`* ...`
	`OPEN OUTPUT DATFILE`
	`ACCEPT W-TIME FROM TIME`
	`MOVE SS OF W-TIME`
	`TO BREAK-VAL`
	`WRITE DAT-REC`

Procedural-Style C#	`Cobol.Open(datfile, "test.dat", FileOpenMode.Output);`
	`Cobol.acceptTime(wTime);`
	`Cobol.move(wTime.ss, datfile.datrec.breakVal);`
	`Cobol.write(datfile);`

OO-Style C# 1	`datfile.Open("test.dat", FileOpenMode.Output);`
	`wTime.SetValue(Cobol.GetTimeAsString());`
	`datfile.Datrec.BreakVal.SetValue(wTime.ss);`
	`datfile.Write();`

OO-Style C# 2	`datfile.Open("test.dat", FileOpenMode.Output);`
	`wTime.Value =`
	`(AlphanumericLiteral)Cobol.GetTimeAsString();`
	`datfile.Datrec.BreakVal.Value = wTime.ss;`
	`datfile.Write();`

Notes:

- datfile, wTime and breakVal are true C# objects in the OO-Style: they represent a specific portion of data and they offer the methods to set, get and manipulate this data.
- In the Procedural-Style, the order of the operands is more like in the original COBOL code, leading to an even stronger visual link between original COBOL code and migrated C# code.
- Both the Procedural and the OO style sometimes require different operands: in the last lines the file object datfile is used to as argument, respectively target of the write method, while in COBOL the WRITEstatement works on the record DAT-REC.
- The OO-Style can take advantage of C# properties for setting the value of a field.

For more information

Astadia
info@astadia.com
www.astadia.com
+1 877-727-8234