Ilyas et al LGURJCSIT 2020

ISSN: 2521-0122(Online)
ISSN: 2519-7991(Print)

LGU Research Journal for Computer Science and IT

Vol.4 Issue4, Oct - Dec 2020

Feature-Based Comparison of Language Transformation Tools

[1]Muhammad Ilyas, [2]Saad Razzaq, [3]Fahad Maqbool, [4]Qaiser Abbas, [5]Wakeel Ahmad, [6]Syed M Adnan

[1,2,3,4]Department of Computer Science and IT, University of Sargodha, Sargodha, Pakistan

[5,6]Department of Computer Science, University of Wah, Wah Cantt, Pakistan

[1]Muhammad.Ilyas@uos.edu.pk,

Abstract

Code transformation is the best option while switching from farmer to next technology. Our paper presents a comparative analysis of code transformation tools based on 18 different factors. These factors are Classes, pointers, Access Specifiers, Functions and Exceptions, etc. For this purpose, we have selected varyCode, Telerik, Multi-online converter, and InstantVB. Source Language considered for this purpose is C sharp (C#) and the target language is Visual Basics (VB). Results show that VaryCode is best among the four tools as its converted programs throw fewer errors and require minor changes while running the program.

Key words: Code transformation, Multiconverter, Stylistic Feature, Code Smell, Author style.

1. INTRODUCTION

Source to source compilation is a process of converting source code written in one high-level language to itself or other high-level languages [1]. It is a refactoring process that is helpful when programs that have to refactor are outside the control of the original implementer. The purpose of transpilation is to convert legacy code to a newer version of a particular language. Such as converting a program from one dialect to another in the same programming language [2, 25]. Translation of code from one to another language is also necessary for understanding code according to expertise in a particular language. This makes code much more readable for the developer. Due to the rapid improvement in programming languages, companies want to migrate their software to new updated/upgraded code. Developing code from scratch each time is difficult because it is a complex and cumbersome activity. The automatic transformation of the code is always very favorable and speeds up the work.

Several techniques have been used for this purpose such that automatic source to source error compensation of floating-point programs [17], generating database access code from domain models [18], and generating pseudo-code from source code [19], etc.

The objectives of feature-based comparison are to help in making newer versions of tools efficient and more accurate as well as to highlight the areas where a particular tool is lagging. For this purpose, we have selected four source to source compilers: Telerik [20], VaryCode [21], Multi-online Converter [22], and Instant VB [23]. We have chosen C# as the source language and Visual Basic (VB) has the target language. We have focused on eighteen different features for comparison, some of these features are Classes, Functions, Access Specifiers and Exceptions, etc. The paper is organized as follows: related work is described in Section 2, feature-based comparison of language transformation tools are presented in Section 3, and the conclusion is described in Section 4.

2. RELATED WORK

A source to source compilation is a process of translating a high-level programming language to itself or another high-level programming language [1]. One of the purposes of the source-to-source compilation is translating legacy code to use the upcoming version of the underlying programming language. It is a refactoring process that is helpful when the programs to refactor are outside the control of the original implementer such as to convert a program from legacy API to the new API, or when the program size makes it impossible to refactor it by hand.

Malton [2] defines three conversion tasks: First is Dialect Conversion which converts a program from one dialect to another in the same programming language. This is used when a new version of the compiler is used. The second is API Migration used to convert the program into a new set of APIs. The third is Language Conversion that converts from one programming language into another.

There are many purposes for code transformation such as performance improvements [11], memory optimization [12], parallelization, and vectorization. Due to the rapid growth of the Internet, one of the main reasons for code transformation is the migration of a legacy system into a web-enabled environment [3]. Code transformation [13] also helps advisory tools that guide the developers to parallelize the real-world problems. These tools have faced problems due to the large size of programs and high code complexity. That's why these tools are unable to provide meaningful parallelization hint to the developers. Then code transformation overcomes this problem by simplifying the code.

Cfir Aguston et al. Proposed the approach to overcome the complexity of code issue called Skeletonization which automatically transforms the complex code into a much simpler structure. The proposed algorithm transforms the constructs such as covers pointers, nested conditional statements, nested loops, etc. This algorithm transforms pointers into integer indexes and replaces C struct references with references to arrays. Source level compiler performs analysis on skeletonized code for parallelizing the code. The generated code is not equivalent to the original code, it suggests possible parallelization patterns to the developers.

Another tool that is concerned with performance improvements is GPU S2S which automatically transforms the C sequential code into CUDA (Compute Unified Device Architecture) code [11]. This contains three modules: Directive recognition, Parsing module, and CUDA

code generation. GPU S2S takes C code with directives as input and translates it into the CUDA code. Experiments showed that generated code has significant improvements as compared to C original code, which leads to performance enhancement.

G. Dimitroulakos and C. Lezos et al. presented a source to source compiler MEMSCOPT for Dynamic code analysis and loop transformations and assisting the optimization of the memory hierarchy of a digital hardware system. MEMSCOPT is a command-line application and is extended by providing a Graphical User Interface (GUI) in C#. It takes a C code file as input and performs an analysis such that monitoring the loop. It also performs a dynamic code translation. MEMSCOPT applies 7 types of transformations such that loop extends, loop shift, loop reversal, loop interchange, loop fusion, loop fission, loop normalization, loop reorder, loop switching, loop scope move forward and loop scope move backward [12].

J. Cronsioe et al proposed another source to source compiler that performs automatic transformations such that optimization of loop structures on multi-core platforms. The proposed research is concerned with a scientific application written in FORTRAN named BigDFT (Density Functional Theory). BigDFT is defined by its heavy use of convolution operators on large arrays. PIPs and BOAST the two S2S applications have been used with MagicFilter to optimize BigDFT. It is found that PIP's generic transformation logic is not always suitable for adapted optimizations there is another technique BOAST for more suitable transformations over multi-core platforms [15].

The concept of source-to-source Translator can be viewed as a generation of Database Access code from Domain Models. N. Y. Khelifi et al. presented his Idea on Database Access code generation from Domain Models. As a source, they used RSL (Requirements Specification Language). The process of code generation consists of two main steps. The first step is related to transformation rules having translational semantics for domain vocabulary constructs of RSL. The second step constitutes of MOLA (Model Transformation Language) algorithms for implementing transformations. The validity of results is assured by using the framework of the ReDSeeDS (Requirements-Driven Software Development System) tool suite. The resultant transformations produced consistent and quality code that can be utilized directly for the implementation of the data access layer [18].

A Pseudogene is a tool for converting pseudo code from Source code using SMT (Statistical Machine Translation), worked on by H. Fudaba et al. The tool performs the transformation of Python code to English or Japanese. The proposed tool uses T2SMT (Tree-to-String machine translation) method. Input to this method is a parse tree through which Tokenization and parsing are performed. The tree is broken up for translation using translation patterns for conversion into pseudo-code. Proposed techniques are one of the best application of SMT for translating the code into natural language [19].

B. S. K. Vorobyov et al worked on source to source Translator from Datalog to SQL for Static Program Analysis. The process starts with the processing of facts and rules by a lexer and parser. An intermediate

representation of the Datalog program is constructed. The intermediate representation is translated to SQL queries by using simple syntactic translation schemes. The proposed work cannot be matched with the performance of industrial-strength tools [16]. L. Thévenoux et al presented his work on automatic source to source Error compensation of floating-point programs. Numerical programs may suffer inaccuracies as finite precision arithmetics is an approximation of real arithmetic. The research considers IEEE 754 floating-point arithmetic and used Error Free Transformations that are lossless of basic floating-point operations. The proposed approach resulted in many accurate values; moreover, it is the first step to automatic generation of multi-criteria program optimizations [17].

C2J [7] is a C to Java translator that translates large volumes of C code correctly to Java. The translated code is difficult to read which circumvents run-time checking system and Java types which make it hard to interface with mainstream Java programs. It requires a lot of modifications in the translated code to be run correctly. Java Backend for GCC also translates C code to Java and has the same disadvantages as C2J[8]. Some translators only focus on the extension of original language like migration from legacy code into object-oriented languages, such as C to C++ language [3]. Ephedra [4] is a tool that translates the legacy C code to Java. It does not translate fully C code but supports a heavy subset of C. Ephedra [9] defines three steps to conversion C/C++ code into Java code. In step 1, conversion of K&R style C code which doesn't contain function prototypes. It limits the capability of the compiler to perform checking. All function prototypes are inserted in this step. In step 2 type conversion and data types are analyzed, as a result, Java incompatible types are removed. In step 3 C/C++ code is translated into Java code, compiled with any Java compiler, and verified. Also, it doesn't support many features such as external libraries, assembly code, and goto statements. Experiments have shown that Ephedra demands source code need to be manually altered for processable [5]. C2Eiff [14], [10] is a tool that performs automatic translation of complete C code into Eiffel, an object-oriented programming language. It supports the complete entire C language such that (Function pointers, pointer arithmetic, unrestricted branch instructions). It compiles GNU C compiler extensions and ANSI with the help of CIL (C Intermediate Language) framework, also support native libraries. The generated code is functionally equivalent to the C code. The completeness of code is evaluated by an attested set of programs to which translation was performed. C2Eif automatically fully translated over 900,000 lines of C code to functionally equivalent 2 Eiffel code. This translation introduces contracts that help in detecting errors such that null pointer referencing etc. This will improve the readability of the code. There are two methods to reuse the source code written in a foreign language into the host language: First is wrapping foreign code that uses the foreign language implementation by API of bridge libraries. The second is Translating foreign code. C2Eiff uses a wrapping foreign code approach to translate only assembly code and external functions.

Martin et al. Presented automatic source code transformation between octave and R fourth-generation languages [6]. The main goal is to convert octave algorithms into R for scientists to use in their applications.

TXL programming language was used for analysis and transformation. TXL was created for the transformation of source and target languages that somehow similar. TXL does not give conversion code into the correct format, for this purpose authors used PERL scripts for correcting the format of code. As a result, the author evaluated its effect on the performance and readability of converted code.

Multi-online Converter [22] and Instant VB [23]. Blanker, another tool that searches and unifies equivalent statements available in the language before feeding the source to an existing code clone detector limited to type-2 clones. Gabriel Sebastián has explained A comprehensive approach to Model Driven Architecture (MDA), from the definition of the computational independent model (CIM layer) to the implementation-specific model (ISM layer) and the process of transformations required for automatic source code generation (in HTML and JavaScript) from Language Learning Apps [26].

3. METHODOLOGY

Our proposed methodology is presented in figure 1. We took a dataset of 50 C# programs from [1] and tested their auto-conversion into Visual Basic on Telerik, Varycode, Multi-online converter, and InstantVB. These tools transform C# code into visual basic code automatically. We have taken a list of features (as given in table

There are numerous online and offline tools available for performing source to source transformation. These tools transform source code of different languages into different target languages such that C++ to C# converter, C# to VB and VB to java, etc. [23]. We have taken C# as a source language and VB (Visual Basic) as the target language to have a comparative study of different tools. The tools we have selected for this purpose are Telerik [20], VaryCode [21], 1) on which we have tested auto-transformation of code from C# to visual basic. Later on, we have checked the code manually and alter it accordingly. We also checked for the areas that are not covered by any of these tools as well as the validity of programs converted.

We have presented the result of our testing in Table 1 considering 18 different features. Some of the features are covered by all of these tools such that Loops, Data Types, Conditions, Access Specifiers, Modifiers and Read and Write methods but the other features vary. These tools do not produce a full running code as we need to amend some of the factors such that placement of Namespaces and Libraries etc. A common problem in all of these tools is that the converted code contains a shared main function within the class that is not supported by the VB compiler so we need to add another main function outside the class and place the main code over there.

4. CONCLUSION

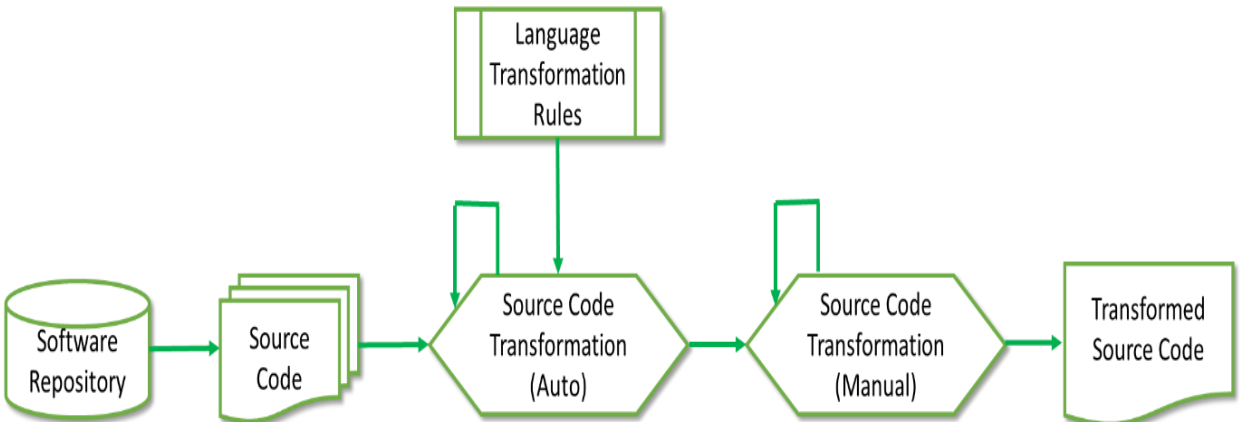

Figure 1: Source Code Language Transformation Model

Due to rapid improvement in the programming languages, organizations want to shift their software to the new updated/upgraded code. Every time, it's hard to develop code from scratch because it's a complex, time-taking activity. So automatic code transformation always very supportive and expedite the work.

In this paper, we have performed a feature-based comparison of four different tools that can transform code automatically from C# to VB. Results of comparative analysis conclude that VaryCode is best among the four tools as its converted programs throw fewer errors and require minor changes while running the program. Moreover, it also exhibits that none of the tested tools support all features and appropriate modifications need to be performed in our converted code to take full benefit.

As future work, our comparative analysis could guide the developers to improve the transformation tools by covering the features that are not supported yet. Also, a mature practice for such transformation could support the shifting of legacy code into the code of modern language. Such transformation will also support software refactoring with better reusability also. Along with all such options, we will extend this work to find out its financial impact

REFERENCES

[1] "Types of compilers". Compilers.net. 1997–2005. Retrieved 28 October 2010.

[2] A. J. Malton, "The software migration barbell." ASERC Workshop on Software Architecture. 2001.

[3] Z. Ying and K. Kontogiannis, "A framework for migrating procedural code to object-oriented platforms." Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific. IEEE, 2001.

[4] J. Martin and H. A. Muller, "Strategies for migration from C to Java." Software Maintenance and Reengineering, 2001. Fifth European Conference on. IEEE, 2001.

[5] T. Marco, et al., "C to OO translation: Beyond the easy stuff." Reverse Engineering (WCRE), 2012 19th Working Conference on. IEEE, 2012.

[6] J. Martin and J. Gutenberg, "Automated source code transformations on fourth-generation languages." Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on. IEEE, 2004.

[7] J. Martin and H. A. Muller, "C to java migration experiences." Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on. IEEE, 2002.

[8] T. Waddington, Java Backend for GCC. http://archive.csee.uq.edu.au/˜csmweb/uqbt.html#gcc- jvm, November 2000.

[9] K. Cashion, S. Ravindran, N. Powar and J. Gold, "IOT device code translators using LSTM networks," 2017 IEEE National Aerospace and Electronics Conference (NAECON), Dayton, OH, 2017, pp. 88-90.

[10] M. Trudel, C. A. Furia, and M. Nordio, "Automatic C to OO translation with C2Eiffel." Reverse Engineering (WCRE), 2012 19th Working Conference on. IEEE, 2012.

[11] Li, Dan, et al., "Gpu-s2s: a compiler for source-to-source translation on gpu." Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third

International Symposium on. IEEE, 2010.

[12] D. Grigoris, C. Lezos, and K. Masselos, "MEMSCOPT: A source-to-source compiler for dynamic code analysis and loop transformations." Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on. IEEE, 2012.

[13] A. Cfir, Y. B. Asher, and G. Haber, "Parallelization Hints via Code Skeletonization." IEEE Transactions on Parallel and Distributed Systems 26.11 (2015): 3099-3107.

[14] S. Ribic, "Concept and implementation of the programming language and translator, for embedded systems, based on machine code decompilation and equivalence between source and executable code," 2006 13th Working Conference on Reverse Engineering, Benevento, 2006, pp. 307-308.

[15] C. Johan, B. Videau, and V. Marangozova-Martin, "BOAST: Bringing optimization through automatic source-to-source transformations." Embedded Multicore Socs (MCSoC), 2013 IEEE 7th international symposium on, IEEE, 2013

[16] S. Bernhard, et al., "A datalog source-to-source translator for static program analysis: an experience report." *Software Engineering Conference (ASWEC), 2015 24th Australasian*. IEEE, 2015.

[17] T. Laurent, P. Langlois, and M.Martel, "Automatic source-to-source error compensation of floating-point programs: code synthesis to optimize accuracy and time." *Concurrency and Computation: Practice and Experience* 29.7 (2017).

[18] Khelifi, N. Yamouni, M. Śmiałek, and R. Mekki, "Generating database access code from domain models." *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*. IEEE, 2015.

[19] F. Hiroyuki, et al., "Pseudogen: A Tool to Automatically Generate Pseudo-Code from Source Code." *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015.

[20] "Code Converter". Telerik Code Converter by progress.Converter.telerik.com/. Web. Accessed 10 Jan 2018.

[21] "Online Code Converter: C#, VB, JAVA, C++, Ruby, Python, Boo". VARYCODE. www.varycode.com/. Web. Accessed. 18 Jan 2018.

[22] "code Translator: Code Translation from VB.NET <-> C# <->TypeScript <-> Java". Carlosag.net. www.carlosag.net/tools/codetranslator/. Accessed. 21 Jan 2018.

[23] "Source Code Converter". Tangible Software Solutions. www.tangiblesoftwaresolutions.com/product_details/csharp-to-vb-converter.html. Web. Accessed. 25 Jan 2018.

[24] "1000 C# Programs with Example Code and output-Sanfoundry". Sanfoundry Technology Education Blog.

www.sanfoundry.com/csharp-programming-examples/. Web. Accessed. 12 Jan 2020.

[25] G. Sebastián, R. Tesoriero and J. A. Gallud, "Automatic Code Generation for Language-Learning Applications," in IEEE Latin America Transactions, vol. 18, no. 08, pp. 1433-1440, August 2020, doi: 10.1109/TLA.2020.9111679.

[26] D. Pizzolotto and K. Inoue, "Blanker: A Refactor-Oriented Cloned Source Code Normalizer," 2020 IEEE 14th International Workshop on Software Clones (IWSC), London, ON, Canada, 2020, pp. 22-25.

Table 1. Feature based Comparison

Features:	**Telerik**	**Varycode**	**Multi online converter**	**Instant VB**
Comments:	Do not convert comments	Convert comments	Do not convert comments	Convert comments
Namespaces + Libraries:	**Predefined Name Spaces:**(Error) in converted code **user-defined Name Spaces:** Do not convert	**Predefined Name Spaces:**(Error) in converted code **user-defined** n**ame Spaces:** Do not convert	**Predefined Name Spaces:** Converted **user-defined NameSpaces:** Do not convert	**Predefined Name Spaces:** Converted **user-defined NameSpaces:** Converted
Loops:	Supporting loops	Supporting loops	Supporting loops	Supporting loops
Data type:	**Primitive Data Types** (Converted: *Int, Date, Unsigned int, char, Double, String, Nullable datatypes*) **User-Defined DataTypes** Converted	**Primitive Data Types** (Converted: *Int, Date, Unsigned int, char, Double, String, Nullable datatypes*) **User-Defined Datatypes** Converted	**Primitive DataTypes** (Converted: *Int, Date, char, Double, Arrays, Strings, Nullable datatypes*) **User-Defined Datatypes** Converted	**Built-in Data Types** (Converted: *Int, Date, char, Double, Arrays, Strings, Nullable datatypes*) **User-Defined Data Types** Converted
Conditional Statements:	Supports conditions *if+ switch*	Supports conditions *if+ switch*	Supports *if+ switch*	Supports *if+ switch*
Access specifier:	Converted	Converted	Converted	Converted
Modifiers:	Converted	Converted	Converted	Converted
Shift Operation:	Not supported	Supports Shift Operation	Not supported	Converts but throws Exception
Read and write Input:	Converted	Converted	Converted	Converted
Classes:	Multilevel inheritance converted Single level inheritance converted Abstract class Inherited classes require Must Inherit in VB but not available in converted code	Multilevel inheritance converted Single level inheritance converted Abstract classes (Converted)	Multilevel inheritance converted Single level inheritance converted Abstract classes (Converted)	Multilevel inheritance converted Single level inheritance converted Abstract classes (Converted)
Macros:	Handles i.e. By making functions	Takes macros in converted code as they are in original program so error	Takes macros in converted code as they are in original program so error	Macros Converted

Exceptions:	Converted (*IndexedOutOfRange, DividingbyZero(datatype conflict), InvalidTypeCasting*	Converted (*IndexedOutOfRange (no proper syntax), DividingbyZero(dividec on flict) , InvalidTypeCasting,*	Converted (*IndexedOutOfRange, DividingbyZero(divideco n flict),invalidTypeCasti ng, nullReference)*	Converted (*IndexedOutOfRa nge, DividingbyZero, invalidTypeCastin g, StackOverflow(Error),*
	, MultipleException, StackOverflow (converted but not working due to specifier with main function), nullReference)	StackOverflow, nullReference)		null reference)
Typecasting:	Typecasting in VB converted but do not follow syntax in some programs. i.e. case of alphabets in statement	Typecasting in VB converted but do not follow syntax in some programs. i.e. case of alphabets in statement	Typecasting in VB converted but do not follow syntax in some programs. i.e. case of alphabets in statement	Typecasting in VB converted.
Functions:	Built-in functions converted (toString, GetType, Copy, Sort, Split, Concat,Substring,Index O F,Replace,Reverse, BinarySearch,Trim) User-defined functions converted Function return type conversion Virtual function converted Method Hiding (Converted) Static method Pass by reference (public/private shared main error)	Built-in functions converted (toString, GetType, Copy, Sort, Split, Reverse, IndexOf, BinarySearch, Trim, Concat, SubString) User-defined functions converted Function return type conversion Virtual function converted Method Hiding converted (Error: overriding demands over-ridable function that is not in conversion) Static method pass by reference not converted	Built-in functions converted (toString, GetType, Copy, Sort, Split, Reverse, Concat, substring, IndexOf, Replace , BinarySearch, Trim) User-defined functions converted Function return type conversion Virtual function converted Method Hiding (Converted) Static method Pass by reference (public/private shared main error)	Built-in functions converted (toString, Copy, Sort, Split, Reverse, Concat, substring, Trim, IndexOf, Replace, BinarySearch) User-defined functions Virtual function converted Method Hiding (Converted) Static method Pass by reference (public/private shared main error)
Online/ Downloadable	Online	Online	Online	Downloadable
Syntax	Sometimes do not follow the syntax	Sometimes do not follow the syntax	Sometimes do not follow the syntax	Follows the syntax
Pointers	Not Converted	Converted but error as not able to create a pointer class (No pointers in VB)	Converted	Converted but error as not able to create a pointer class (No

				pointers in VB)
Jump statements:	Converted (goto)	Converted (goto)	Converted (goto) but do not follow the syntax	Converted (goto)