Automasjon 2016

The analysis of C# to F#

Jostein Andreassen, Michael Blomli and Mikkel Eltervåg

Automation

12. May 2015

Prosjektrapport – Side 2

Universitetet i Tromsø
Institutt for ingeniørvitnskap og sikkerhet
9037 TROMSØ

Studieretning: Automasjon	**År:** 2016

Tittel: The analysis of C# to F#	Dato: 12. Mai Gradering:
Forfattere: • Jostein Andreassen • Michael Blomli • Mikkel Eltervåg	Antall sider: Vedlegg:
Fortrolighet:	
Veileder: **Puneet Sharma**	
Oppdragsgiver: **Serit IT Partner Tromsø**	Oppdragsgivers kontaktperson: Jonas Juselius

Sammendrag:
Stikkord: F#, C#, fungsjonell programmering, analyse, programmering

Summary

This report is written by 3 students from the final year in the Automation class at “UiT: University of Tromsø - Arctic university of Norway”.

The assignment is given by a company named Serit - IT Partner from Tromsø. They wanted us to find out the benefits, flaws and our experience from learning a new programming language called F#. Today Serit mostly uses the famous programming language C# which is widely used around the world today. The time spent writing code, debugging and stability in these languages is very valuable.

Preface

This thesis is written by graduates from the automation program at the department of engineering at UiT: University of Tromsø - Norway's Arctic university. This thesis will be used by Serit to help them decide if they are going to incorporate the programming language F# or not.

We chose this assignment because everyone in our group enjoys programming, it had some database management and set up, and because it looked like an interesting project overall. Since we already had some experience in C#, we thought it would be fun to find the "pros and cons" of this relatively new programming language.

We have written the thesis in LaTeX which is a word processor and a document markup language. When we were working on the raw text we used Google documents so that we all could work together simultaneously on writing and editing the text for the thesis. On the main assignment we used Visual Studio (with C# and F# tools), Atlassian Sourcetree and Github to write and manage our code, and SQL Server Management Studio to handle the database. By using both C# and database we covered a wide area of our education from the previous semester. We have learnt a lot and have had the privilege of testing out the new attractive programming language F# that may just be the future.

We want to give out a special thanks to Serit for all the great support and advice given, on all the regular follow up meetings we had and by email. We also want to give a big thanks to our mentor Puneet Sharma for his great support and contribution to the assignment.

Contents

1 Introduction

1.1 Background

One of the biggest problems in modern application development is the rapidly growing complexity of all major software systems. This complexity makes it almost impossible to ensure the quality and accuracy of the code. It also becomes harder and harder to make changes to existing code without introducing new errors. All these difficulties multiply when you also want utilize modern computers with many CPU (central processing unit) cores for increased performance.

The imperative object-oriented programming paradigm has been dominant in software development for over 20 years. In the imperative paradigm the state variables will be handled explicitly, which can quickly give too much complexity. The functional programming paradigm has been known since the 1930's, but has not been popular with professional developers because of the slightly lower (single core) performance and greater resource use. Today these obstacles are long gone, and functional programming is experiencing a new renaissance due to significantly better control over complexity and parallelism.

We've received an assignment from Serit to translate parts of an existing project from C# code to F# code. C# is meant to be simple, modern, flexible and object oriented programming language. It is developed and maintained by Microsoft and is inspired by previous popular object-oriented languages like C++ and Java. F# is a hybrid language that supports both the familiar object-oriented method and functional programming. F# is also developed by Microsoft, and like C# also has access to Microsoft's .NET framework. Serit wants us to find out the benefits of switching from development in the programming language C# development to F#.

1.2 Problem for discussion

C# and F# works in different ways, they both have benefits and drawbacks. The main question is if it is worth it for a company to change their main programming language. We have to look at what the company wants to achieve by making the change, and that boils down to making quality programs for a low price.

A modern IT company uses a lot of time developing, changing and fixing code. If we can use a programming language that takes less time to develop and at the same time works better without generating errors, that could be very cost saving.

The programming language F# claims to be a solution to these problems by using fewer lines of code, be more simple and have better error handling than other programming languages. Our task is to find out if those claims are true by answering these questions:

- What are the benefits of switching from C# to F#?
- To what degree can we reduce the number of lines written in the program code?
- How much time is saved in the debugging stage?
- How much time is saved in the development of the code?

1.3 Formulations of objectives

We want to find out if it is worth it for Serit to change their main programming language from C# to F#. To do this will we do a thorough analysis to find out what the good and bad sides of F# programming are. All this is done according to our goals:

- Learn to program in F# to:
  - get a good enough understanding of how to write simple programs.
  - see how it is for a programmer to learn the new language.
  - get our own opinions about the language.
- Learn from translating a real program from Serit by:
  - finding the best translating method.
  - looking at the development process for F#.
  - learning how F# handles the databases.
  - looking on how F# does with debugging and error handling.
- Learn from online research by looking at:
  - other people's opinions about F#.
  - how other companies take advantage of F#.
  - good code examples that others have written.
  - what are some good usages of F#.
- Write a complete analysis that contains information about:
  - development time.
  - readability and clarity.
  - debugging and error handling.
  - performance.
  - our experiences.
- Make a conclusion if we think that Serit should change their main programming language from C# to F#.

1.4 Project specification

1.4.1 Where Serit is now

- They have an ASP.NET Web application in C# where the user interface is based on ASP Web Forms. All code is written in English, as well as all the text in the user interface.
- Language support is dissolved in a separate library sCore.Translation which is called from the application and performs translation according to data recorded in a translation table.
- Translation tables are located in a SQL database.

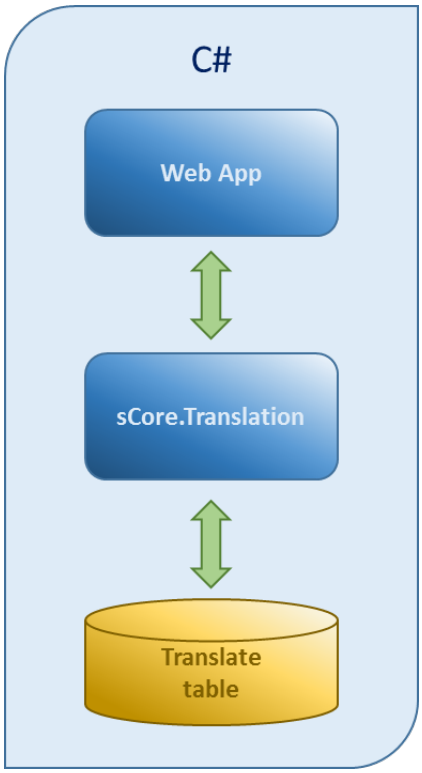

Figure 1: How the communication of Serit's sCore.Translation application looks now.

1.4.2 What Serit wants

- They want to have the existing translation library sCore.Translation developed as a separate library in the functional language F#. This should be able to be called from the present imperative program (C#) and from functional programs (F#).
- With the translation from C# to F# done, both languages and programming paradigms can be compared analytically. By this we can evaluate benefits (and possible disadvantages) with the functional paradigm in relation to an object-oriented imperative paradigm. The analysis will provide a better basis in the choice of programming language in future development projects.

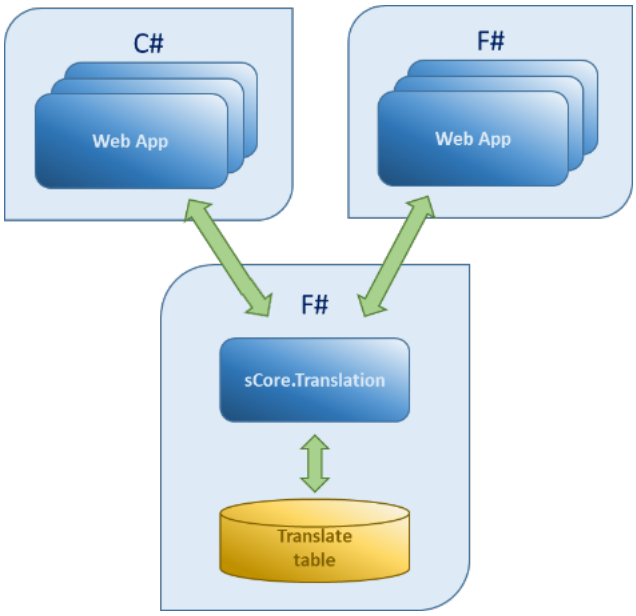

Figure 2: How they want the sCore.Translation application to communicate.

1.4.3 Method of translation

F# for fun and profit describes three levels of "sophistication" for porting code from C# to F#. The basic level is simply a direct port. Since F# supports imperative programming, we can translate directly. At the intermediate level, the code is refactored to be fully functional. The advanced level takes advantage of F#'s data type system.

There are two paths to achieve this goal: Either by first porting to F# and then refactoring to functional code, or by converting to functional code in C# before porting that to F#.

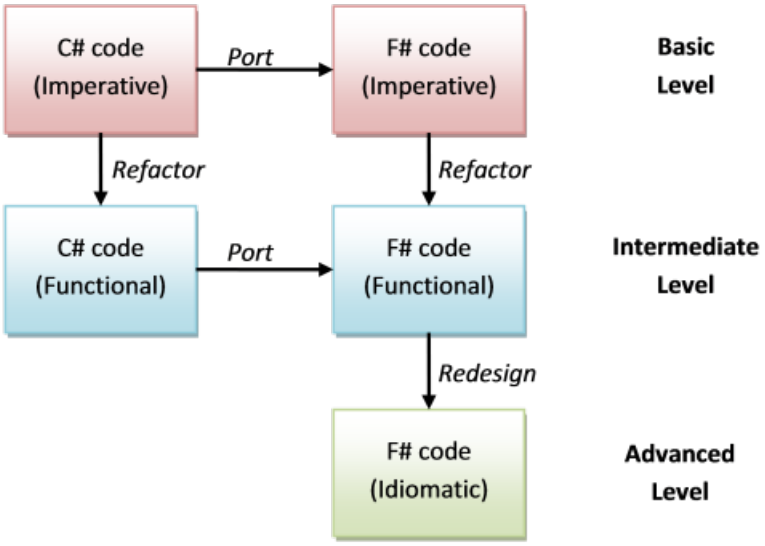

Figure 3: Method's of translating from C# to F#.

2 Different programming paradigms

There are many different ways of classifying programming languages and styles, the most common being imperative versus declarative programming. In imperative programming, statements are used to tell the computer program what to do. It is the programmer's job to tell the computer how to solve the problem. Declarative programming on the other hand, focuses instead on telling the computer program what the desired result is, and then the implementation of the programming language decides how to do it. For instance, if an imperative program should make a person get a cup of coffee, it would have to list all the individual tasks they had to do, like stand up, walk to the coffee machine, place the cup, press the button and so on. While a declarative program would simply state "Give me a cup of espresso". Examples of declarative programming include database query languages and functional programming.

Another distinction is that of procedural versus object-oriented. These are different styles of breaking the program down into smaller, reusable parts. In a procedural style, the program would be broken down into subroutines or functions which would take arguments and may return results to the caller. In object-oriented programming, tasks are broken down into objects that expose behavior and data using interfaces. Going back to the coffee example, a procedural program would call a "get coffee" function that would take two arguments: which person should get the coffee and what type of coffee he should get. Whereas an object-oriented program would invoke the "get coffee" method of an instance of the class "person".

(Part about functional programming goes here)

3 F# programming language

4 sTranslate

sTranslate are a library that are part of a bigger program. The purpose library is to take in a word as an argument and then use a language database to translate the word.

The library is written in C# and contains 2 major functions, one is designed to use when you are only going to translate one word and the other one is designed to be better at translating multiple words. Even though the function works different they have the same inputs and outputs, that means that they do the same thing if you look at it from outside. You can see a illustration of the library in figure 4.

Our task from Serit is to take this C# code and translate it into F# with 2 different approaches. The first one we translate line for line and make the code the same way that the C# is coded, the other one is to translate the program the way F# is meant to be written. After we have translated it we have to a lot of optimizing to make the code better, faster and shorter. Then look at performance, difficulty of programming and all other obvious differences.

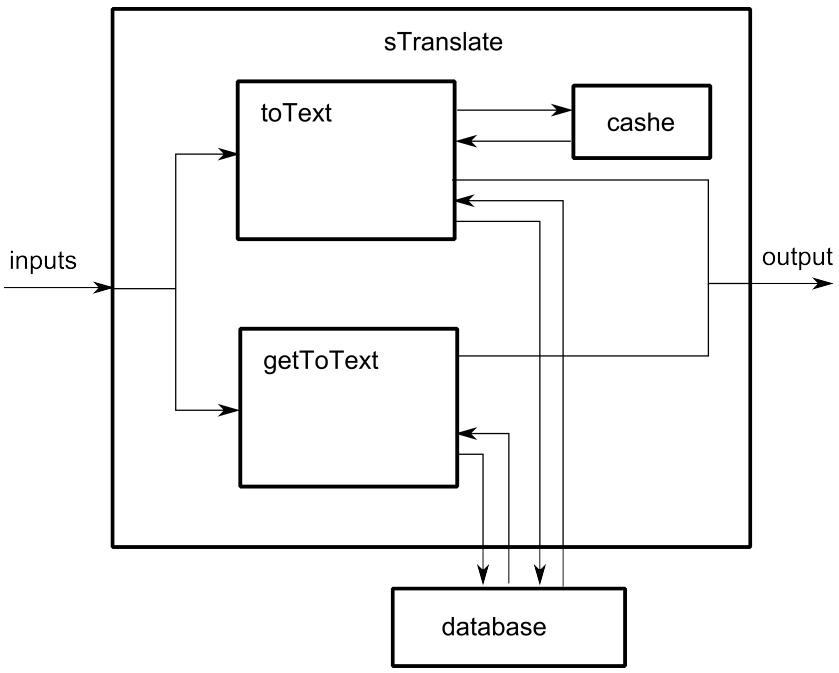

Figure 4: sTranslate library illustrated.

4.1 How it works

sTranslate library is supposed to be used with both C# and F# programs. It is important that the library is taking the same inputs and same outputs in both languages. Figure 5 show this communication.

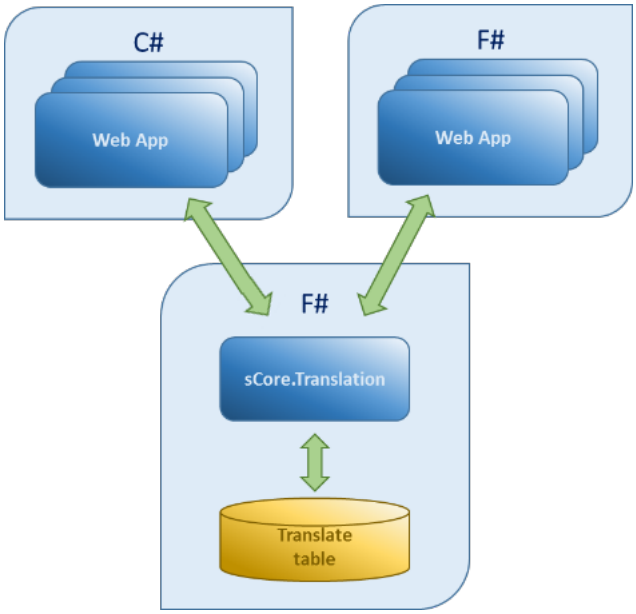

Figure 5: How Serit want the sCore.Translation application to communicate.

The main part of the library is two functions: toText and getToText, you can see this in figure 4. Both functions do the same thing but there are one major difference in them. In getToText you have to open the language database each time you want a word but in toText the database is cashed so it does not need to reopen the database.

4.1.1 Inputs and Outputs

The way a library like sTranslate works is that it takes in inputs and give out outputs just like a function. This make it really easy to use, change and reuse a library or a part of a program.

This works really well with C# and F# because their close integration with the .net system. A C# program can easy give inputs or take outputs from F# and the other way around. How we are going to use this feature can be seen in figure 5.

sTranslate is taking in 5 inputs:

- **fromText**, the word in English that the library is translating.
- **context**, information over how the word is used, examples: string, title, lable.
- **property**, that the word's property, examples: id, text, tooltlip...
- **criteria**, the criteria that the search is going to get a match, examples: startWith, endWith, Contains.
- **toLang** the language the library is going to translate to. examples: no, ge.

sTranslate gives only out one output and that is the translated word. All the inputs have to match up to the row in the database to send the output. In there is no match fromText (the English word) will be sent back as output and if there is two matches in the database only the first one will be sent back.

4.2 Solution

There are two ways of translating code, the first one is to directly translate the program and the other is to translate it how it shod be written in the new language. Here we look at both, the direct translation and the more functional approach and look at pros and cons of the methods.

```
1 if (a == b){
2   Console.WriteLine("equal");
3 }
4 else{
5   Console.WriteLine("not equal");
6 }
```

Figure 6: C# code example

```
1 if a = b then
2   Console.WriteLine("equal")
3 else
4   Console.WriteLine("not equal")
```

Figure 7: Direct translation code example

```
1 match a with
2     | b -> printfn "equal"
3     | _ -> printfn "not equal"
```

Figure 8: Functional approach code example

4.2.1 Direct translation

The direct translation is maybe the most easy and the fastest method to translate the program. But i maybe have more flaws and have more poor performance to do it this ways. Below we have some examples from the translation that show the direct translation.

```
1 Insert code here!!!
```

Figure 9: Direct translation code

4.2.2 Functional approach

The functional approach is the more proper way to do it but it can take longer time and be harder to do if the person is not fluent in the new language. But the benefits can be better preference, less flaws and bugs. Below we have some examples from the translation that show the functional approach.

```
1 Insert code here!!!
```

Figure 10: Functional approach code

4.3 Analysis

In this chapter we are gonna take a look at the analysis of the library sTranslate.

4.3.1 Performance

This chapter is not a demand from Serit, insted this was something we wanted to test out for ourselves and is just a bonus for Serit if there proves to be better performance in F# vs C#.

4.3.2 Experiences

The experiences that we can draw from this assignment is that in F#, it is firstly very easy to read the code and what it does, and you hardly need to comment the code. But the thing we appreciated the most was the direct response from the compiler and Microsoft's Intellisense technology in real time while writing the code. At first we thought that this was a bit annoying since you always saw errors made in code with the red lines everywhere. But since the language is very strict about types and such, you are forced to root out many errors before you even run the program. You don't have to deal with the problem later when running the program and making it crash instead. So we grew to really enjoy this feature. It makes you put a little more time in thinking about what you want the code to do, but you will spend less time dealing with problems later while running the program.

In Michael's experience the hardest part and the thing he struggled and is still struggling with is that he kept finding himself getting back in think in object-oriented "thinking". Its hard to think in a more functioning approch.

4.3.3 Lines of code

C# - about 280 lines, 3 files of source code, 1 project + 1 project of autogenerated code for entity framework
F# imperative style - 170 lines, 4 files, 1 project

F# functional style - 145 lines, 2 files, 1 project.

5 Complete analysis

5.1 Development time

5.2 Readability and clarity

Readability and clarity is really important when writing code. The reason for this is that it is much more easy to find bugs, remove bugs and add future content to the code. If it is a bad written code a programmer can use a lot of time just familiarize themselves with the code before he can do changes to it.

For the most part i is up to the programmer to write readable and clear code, but the programming language can have a lot to do with helping the programmer in this matter.

One programming language a lot of programmers can agree on being easy to read and understand is Python. Python is a very high-level language that have a syntax that help programmers write programs in few lines of code. They have a large library that are easy to understand and use. Examples of Python code can bee seen in figure 11 and 12.

```
1 print "Hello world!"
```

Figure 11: Hello world program in Python

```
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n-1)
```

Figure 12: Python code example of a recursion function (function that's call itself) that are calculating factorials. This can be shorted down by using a function from the math library.

One commonly used example for a programming language that is not easy to read or understand is Assembly. The reason that assembly can be hard to read and understand for some people is that it is for the most parts using only instructions and memory addresses. In figure 13 you can see an example of a "Hello world!" program, most other modern programming languages does this in only one line like you can see at the Python example in figure 11.

```
1  section .text
2       global _start     ;must be declared for linker (ld)
3  _start:                ;tells linker entry point
4       mov edx,len       ;message length
5       mov ecx,msg       ;message to write
6       mov ebx,1         ;file descriptor (stdout)
7       mov eax,4         ;system call number (sys_write)
8       int 0x80          ;call kernel
9
10      mov eax,1         ;system call number (sys_exit)
11      int 0x80          ;call kernel
12
13 section .data
14 msg db 'Hello, world!', 0xa  ;string to be printed
15 len equ $ - msg        ;length of the string
```

Figure 13: Assembly "Hello world!" code example from tutorialspoint.com [2].

So the question is how does F# do in this matter? Is it easy or hard to write readable and clear code. How would a bad or new programmers code look? And how readable is a perfect written code code?

5.2.1 Indentation and code structure

F# is a whitespace sensitive programming language this means that indentation do have a meaning and will be read when you compile the code. When you are using indentation the compiler will read the indented lines as a sub code of the code over. You can se this illustrated by code in figure 14.

```
1 let f =    // Line 2-5 is the sub code of this line
2   let x = // Line 3 is the sub code of this line
3     5+5
4   let y=1
5     x+y
6 let a = f+2 // This line is not a sub code
```

Figure 14: Example of how indentation works in F#.

That means that to some extent the F# language force the programmer to write code that have good structure in the code. This make it easy to understand and read the code even if you don't know it beforehand.

For an inexperienced programmer this can be a little confusing at fist. But as this is a really good practise to learn early on it can help the inexperienced programmer write better code.

Indentation and code structure wise is F# doing really good when it comes to readability and clarity.

5.2.2 File structure

In most programming languages do the file structure nothing to do in how the program is rune. F# does this a little different all the files have to be in a specific order to run the program correct.

Whenever you call a function you have to make sure that the function is written before the call in the code. And since all the files are read in order you have to make sure that if the function is in another file it have to be over in the file structure.

The most common way to organize the files in a project is to have all the files in the same folder. This makes it more transparent and easy to order the code. This way of organization can word really well for small and medium sized projects, but for really big projects it may be better to organize in multiple folders.

A good practice when organize the files is to only have one module per file and make the functions inside correspond to the file name.

5.2.3 Similarity to other coding languages

Even though F# have taking inspiration from a lot of different modern programming languages it still have a lot of roots to ML that is a old and not that popular programming language. This makes the syntax very different to how syntaxes normally looks today, and can make the syntax hard to read and write to any programmer that are not used to ML or other similar programming languages.

```
1 fun factorial 0 = 1
2   | factorial (n:int) = n * factorial (n-1)
```

Figure 15: ML code example of a recursion function (function that's call itself) that are calculating factorials. Example from Carnegie Mellon University[3]

One other thing that makes F# a little odd when compared to other languages is that is it a functional programming language. This makes the syntax a lot different to all other major languages.

Those 2 things makes F# really different to other languages. This can make it very hard for a experienced programmer learn to read and write this way. For an experienced programmer to learn F# you have to use a lot of time and resources. For a new programmer it can maybe be easier to learn F# just because he will not fall into old habits.

> “After working width an object oriented mindset in programming for several years I found it very difficult to write good functionally code. Often I went back to old habits and used code written in a object oriented way when there were much better ways to do it functionally.”
>
> *Mikkel Eltervåg*

5.3 Debugging and error handling

5.4 Performance

6 Conclusion

7 Reference list

References

[1] F# for fun and profit
https://fsharpforfunandprofit.com

[2] Tutorialspoint
http://www.tutorialspoint.com/assembly_programming/

[3] Carnegie Mellon School of Computer Science
https://www.cs.cmu.edu/~rwh/introsml/core/recfns.htm

8 Attachments