CAB402 Programming Paradigms
Assignment 1 - Evolution

Prepared for: Dr Wayne Kelly
Associate Professor Jim Hogan
Prepared by: Thanat Chokwijitkul n9234900

Queensland University of Technology (QUT)
18 April 2016

EXPERIENCE AND COMPARISON

This assignment is an application for simulation of the genetic evolution process. The application utilises the computational power of different programming paradigms to simulate a myriad of living organisms evolved from a single first organism on the patterns and mechanisms of evolution.

There are three approaches used in the development process which result in three different implementations of the same application. The first two approaches comprise the implementations using F# in a strictly pure functional style and stateful using mutable state respectively. The last approach is the implementation using C# in a strict object-oriented style.

The Efficiency and Effectiveness of the Three Approaches

The first approach is the pure functional paradigm, which treats all components of the application as mathematical functions. This approach avoids changing-state and mutable data, resulting in being self-contained and stateless. These features bring an array of benefits, including increases in readability and maintainability because each function is specially designed to accomplish a specific task by taking in arguments and return a single solution based solely on the input (MSDN, 2015). Additionally, due to its immutability, concurrent update problems cannot occur with this approach, which makes the program safer and cleaner but it also forces the program to create or bind new values every time when data modification is needed, such as when creating a new gene, the “create” function will return a new map consisting of all the existing genes plus the newly created gene. By applying this approach to the first implementation, it is efficient in terms of the mentioned features and benefits.

Even though F# strongly discourages the practice of having mutable data to avoid state, it does make the second implementation of this application more efficient. Instead of using the built-in F# immutable collection types, this approach utilises the generic collections from the .NET framework class library. This obviously introduces state to the program by the use of global variables and data mutability, which also allows the application to run more efficiently with direct data modification instead of binding or creating new values every time the program state changes.

With the object-oriented paradigm, it apparently differentiates the implementation from the first two approaches. Instead of thinking functionally by designing a set of functions to solve a specific task, each unit of composition that needs to be designed is an object. By treating each component as an object, computations can be expressed by data transferred between objects. Based on the object-oriented concept, it makes the application more comprehensive and organisable, which contributes to the ease of development and maintenance with slightly negative effect on its performance compared to the first two approaches.

Ease of Development

In terms of ease of development, most developers with the object-oriented programming (OOP) background, who have been accustomed to programming with the imperative or OOP paradigms, may consider the OOP approach as the easiest way to program. However, once the concept of thinking functionally is mastered, the strictly functional approach with no mutable state (the first approach) will become more comfortable to program due to various effective functional programming concepts and features that can be utilised.

Readability, Understandability and Maintainability

These qualities on the functional and object-oriented paradigms may vary due to the differences in the design patterns. By comparing the two approaches, the functional approach is reputable for its succinctness which is more difficult for OOP to offer. This quality results in the increases of readability and maintainability. However, even though the functional programming may offer the ability to make the program compact, whereas writing overly verbose code can cause a direct negative impact on both readability and understandability of the program.

Conciseness

Conciseness is one of the key features of the functional paradigm. In the world of F#, a piece of code that performs complex calculations can be constructed from a set of basic functions (Wlashchin, 2012), which increases the readability and compactness of the resulting function. Additionally, other functional features, such as pattern matching, tail-recursion, collection types .etc, can efficiently assist the application in performing multiple trivial tasks simultaneously in order to build up a complex computation that will be used to produce the final output. Hence, it can be concluded that the first two functional approaches are more concise compared with the object oriented programming approach in this case.

Code Efficiency

If evaluating the code efficiency of the three approaches based on the execution speed and programming paradigm used in the application development process, the impure functional version of the application takes the shortest execution time to produce both the gene results and homologous gene tracking results (618.125 milliseconds on average) following by the pure functional version of the application (1238.625 milliseconds on average), while the third approach (OOP) takes longest period of time (2646.375 milliseconds on average) to produce the same results. This is the result of the experiment on the Windows 10 virtual machine (a 32-bit OS with an x64-based processor) with Intel(R) Core(TM) 1.10-1.20 GHz processor and 4 GB of RAM. Therefore, the result may vary and cannot be determined using static quantitative data due to the unequal efficiency of different compilers and some related hardware issues.

REFERENCES

MSDN. (2015). *Functional Programming vs. Imperative Programming*. Retrieved April 10, 2016, from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/bb669144.aspx

Wlaschin, S. (2012). *Conciseness*. Retrieved April 10, 2016, from F# For Fun and Profit: https://fsharpforfunandprofit.com/posts/conciseness-intro/