

The Use of C# as a First Programming Language

William Bishop

wdbishop@uwaterloo.ca

George Freeman

freeman@uwaterloo.ca

Department of Electrical and Computer Engineering

University of Waterloo

Waterloo, Ontario, Canada, N2L 3G1

Abstract

This paper examines the use of the C# programming language for a first course in computer programming in an engineering curriculum. C# is an object-oriented programming language that incorporates language features from C, C++, Java, and Delphi. The Department of Electrical and Computer Engineering at the University of Waterloo has been using C# in its Introduction to Computing course for the past two years. This paper summarizes the experiences of the students, staff, and instructors with respect to the use of the C# language. The strengths and the weaknesses of the C# language are exposed. The goal of this paper is to provide practical insight into some of the unique features of the C# language based on actual classroom experiences. This paper does not attempt to advocate the use of C# for a first course in computer programming. It simply presents the pros and the cons of the use of the language in a classroom so that instructors may make an informed decision.

Keywords:

C# programming language, object-oriented design, teaching experiences

1. Introduction

In an engineering curriculum, there are many criteria to be considered when selecting a programming language for a first course in computer programming. Some of the criteria that should be considered are the following:

- Difficulty of learning the programming language
- Suitability of the programming language for teaching
- Applicability to real-world engineering applications
- Availability of qualified faculty and staff members

- Influence of the language selection on other courses
- Cost of development tools

It is important to remember that the first programming language that a student uses will likely define how the student perceives important programming concepts [2]. The first programming language serves as a reference for all future programming experience. A bad experience with programming may cause a student to dislike programming and in some extreme cases, choose a completely different career path. Clearly, the selection of a first programming language is not a decision that should be taken lightly.

A large body of research on the subject of the selection of a first programming language exists in the literature. Some researchers have attempted to identify the undesirable features of programming languages from a teaching perspective [6]. Others have tried to advocate the use of a particular language [4] for a first course in computer programming by proposing a framework for teaching [3]. Finally, some researchers have tried to quantify programming language use to gain insight into the selection process [8].

One of the problems with advocating a particular programming language is that an exhaustive list of selection criteria is difficult to build. In addition to the selection criteria previously mentioned, there are many factors that can complicate the selection process. For example, instructors may be given the academic freedom to select development tools and programming languages within certain well-defined limits imposed by their institutions. If a course instructor is a strong advocate for a particular programming language, it is more likely that the language will be selected. In some situations, instructor selection is more important than language selection.

Another factor that may complicate the selection process is industry sponsorship. In an ideal world, funding issues would not play a role in the selection of a programming language. Unfortunately, we do not live in an ideal world. The cost of development tools and support is not negligible. The availability of free tools for most programming languages

helps to mitigate the costs of adopting a new programming language. However, the availability of affordable, commercial development tools can still play a significant role in the selection of a programming language.

This paper does not attempt to advocate the use of the C# language or any other programming language for a first course in computer programming. It is the opinion of the authors that any one of a set of programming languages may be equally suitable for use in a particular programming course. Any language covering the essential programming concepts will suffice in most cases.

This paper simply presents some of the important features of the C# language from a teaching perspective. The goal of this paper is to provide the reader with the information required to make an informed decision about the use of the C# language in a classroom setting. Section 2 describes some of the unique and interesting features of the C# language. Whenever possible, code samples and insights gained from actual classroom experiences are presented to augment the discussion. Section 3 summarizes the feedback provided by students, staff, and faculty with respect to the use of the C# language. Section 4 makes recommendations for the practical use of the C# language based on the experiences of the authors.

2. Important C# Language Features

C# possesses a number of important language features that make the programming language unique and interesting from a teaching perspective. Some of these language features are clearly strengths while other language features can be viewed as significant weaknesses. Whether you view a particular language feature as a strength or a weakness depends upon your perspective and your teaching philosophy. Like language selection, the relative importance of language features can be highly subjective.

Ten of the most important C# language features that should be assessed when deciding whether C# is a suitable language for a first-course in programming are the following:

- object-oriented design paradigm
- similarities with other languages
- industrial strength development tools
- managed code model
- input / output statements
- value types vs. reference types
- parameter passing
- unified type system
- rectangular arrays
- unsafe code

```
using System;

// This program creates a structured type named Hello World to encapsulate
// the program entry point
class HelloWorld
{
    // The program entry point is defined with the modifiers public and
    // static. These modifiers must be explained very early in a programming
    // course that uses C#.
    public static void Main( )
    {
        Console.WriteLine( "Hello World" );
    }
}
```

Figure 1. A Hello World Program in C#

2.1. Object-Oriented Design Paradigm

C# is an object-oriented programming language that enforces the use of an object-oriented design paradigm. Every C# program must create a structured type in the form of a `struct` or a `class` that encapsulates a program entry point. An implementation of a Hello World program in C# is shown in Figure 1. This program clearly illustrates the development of the `HelloWorld` class and the `Main` program entry point encapsulated within the class. Using C#, students learn to think in terms of objects and the methods encapsulated by objects.

The use of an “objects-early” approach to teaching programming is the subject of much debate [1]. While some people advocate an “objects-late” approach to teaching programming, it is clear that an “objects-early” approach has benefits if it can be taught successfully. Certainly for simple programs such as the Hello World program, the object-oriented design paradigm introduces complications that may seem unnecessary. However, this approach has the advantage of introducing the concepts of objects, encapsulation and access rights very early in a course so that students have sufficient time to fully understand the concepts. Based on our experiences at the University of Waterloo, we believe that a language that assists with an “objects-early” approach to teaching programming is a significant advantage when combined with the right course materials. When our programming course first transitioned to C#, we noticed a 5% increase in student averages over the long-term trend and a drop in the failure rate of approximately 40% [7]. Students clearly had a better understanding of object-oriented design when they started using objects early in their first programming course.

2.2. Similarities with Other Languages

C# incorporates language features from many popular programming languages including C, C++, Java, and Delphi. The syntax of C# is similar to the syntax used by the C programming language developed in the 1970s. The C language syntax has been in use for nearly 30 years so it is familiar to faculty and staff members. The C# language re-

```

using System;

class Properties
{
    public static void Main( )
    {
        // The next line of code defines a variable p1 of Point type and
        // calls the default constructor to initialize it.
        Point p1 = new Point( );

        p1.Data = 2;
        Console.WriteLine( "p1.Data = {0}", p1.Data );
    }
}

class Point
{
    // This class declares data field as private.
    private int data;

    public int Data
    {
        get{ return data; } // Defines an accessor
        set{ data = value; } // Defines a mutator
    }
}

```

Figure 2. Properties in C#

sembles Java closely in terms of functionality. Microsoft even offers a tool that automatically converts simple Java programs to C# programs. Advanced features such as properties were borrowed from Delphi. A property is a convenient way of implementing accessors and mutators. An example of a property is shown in Figure 2. The borrowing of syntax and language features from other programming languages can be beneficial or detrimental, depending upon your perspective.

2.3. Industrial Strength Development Tools

C# is supported by commercial development tools and public domain development tools. The .NET Framework Software Development Kit includes a free C# command-line compiler (`csc.exe`) that is suitable for use on a campus network. With relatively little effort, it was possible to install the command-line compiler on our campus network. The command-line compiler allows students to focus on computer programming rather than on the use of a complex integrated development environment (IDE). Of course, C# is also supported by a rich set of integrated development environments including Visual Studio .NET, Borland's C# Builder, and the Mono C# compiler. The Mono C# compiler extends support to Linux and Mac OS platforms. The availability of industrial strength development tools on a wide range of computing platforms makes C# a potential choice for use in a programming course.

However, there is one important limitation to note about the use of C# development tools on a campus network. If you intend to use the command-line compiler or the Visual Studio .NET environment, the security model prevents the use of certain language features (e.g., file output, debug methods, trace methods, etc.) on network drives. For example, a program that resides on a network drive will not be permitted to open a file for writing. Despite our best ef-

orts, we could not find a suitable solution to this problem without compromising the security of our campus network. Instead, students are instructed to use a local drive when working on programs that require language features prohibited by the security model of C#.

Occasionally, the flexibility and power of the C# language and the .NET framework can complicate the teaching of the language. The C# language and the .NET framework have been designed for professional programmers. C# incorporates advanced language elements that are beyond the comprehension of most first-year students. For example, language elements such as the "is" and the "as" operators can be used to build sophisticated, polymorphic code that is very difficult for a novice programmer to fully understand. Similarly, the .NET framework provides a rich library of routines that can easily overwhelm students. For example, novice programmers may have difficulties understanding the subtle differences between `int.Parse()` and `Convert.ToInt32()` routines. Either routine can be used to convert a `string` to an `int` but the routines are quite different in their behaviour.

2.4. Managed Code Model

C# uses a managed code model where objects are managed by a sophisticated memory management system. For a first-time programmer, this is particularly important since the finer details of object construction and destruction can easily be misunderstood. The most noticeable impact of a managed code environment is that computers rarely need to be rebooted due to side effects of bugs in student's programs. When C++ was used in our course in previous years, students would frequently crash their computers when working on their linked list assignments. Crashes can be both demoralizing and time-consuming to a first-time programmer. Using C#, it is very difficult for a student to build a program that will accidentally corrupt the operating system. When a student builds an incorrect program that attempts to overflow a buffer or access an incorrect memory location, the exception handling system catches the problem and reports it back to the student immediately. Students are able to learn from their mistakes quicker in this manner.

2.5. Input / Output Statements

C# provides a convenient set of classes and routines for console input and output as well as file input and output. The input / output statements for both console I/O and file I/O utilize a consistent interface. Once a student has mastered console I/O, the understanding of file I/O is a trivial task. For example, the `WriteLine()` method is used when working with the console or when working with text

```

using System;
using System.IO;

class FileIO
{
    public static void Main( )
    {
        // Create a reference to a new text file.
        StreamWriter outputStream = new StreamWriter( "output.txt" );

        // Output a Hello World message to the file.
        outputStream.WriteLine( "Hello World" );

        // Close the file.
        outputStream.Close( );
    }
}

```

Figure 3. File I/O in C#

files. This makes file I/O easier to teach to novices. An example of the simplicity of file output in C# is shown in Figure 3.

However, some members of our teaching team expressed disappointment with the fact the file input can be more difficult in certain situations. The `Read()` method reads one character at a time. The `ReadLine()` method reads one line of text at a time. C# does not provide a convenient method to read one integer at a time or one floating point value at a time. Instead, strings of characters must be parsed to obtain the values desired. There are ways to avoid parsing data in this fashion but they are crude compared to other languages. While the consistent interface is excellent from a teaching perspective, it limits some of the flexibility desired by experienced programmers.

2.6. Value Types vs. Reference Types

C# attempts to make a clear distinction between a value type and a reference type. A variable of value type directly contains data. A variable of reference type stores a reference to data. Value types are useful for storing simple data while reference types are useful for creating complex objects and abstract data types such as linked lists. C# programmers may create structured types using a value type in the form of a `struct` or a reference type in the form of a `class`. Figure 4 illustrates a few of the important differences between value types and reference types.

First, storage for a `struct` variable is allocated automatically while storage for an instance of a `class` requires the use of the `new` operator. Second, `struct` variables are stored on the stack while instantiations of a `class` are stored on the heap. Finally, it should be noted that an instance of a `struct` uses value semantics during assignment statements while an instance of a `class` uses reference semantics. In other words, the programmer must know the precise implementation of the structured type to use it correctly. In the example shown, the underlying implementation of the types is not clear in the code that declares instances of the structured types. This can be quite confusing to a programmer, regardless of their experience level.

```

using System;

class StructuredTypes
{
    public static void Main( )
    {
        Type1 c1 = new Type1( );
        Type1 c2 = new Type1( );
        Type2 s1;
        Type2 s2;

        // This line of code only works if a new instance of a Type1 class
        // has been created.
        c1.data = 1;

        // This line of code instructs the compiler to have c2 refer to c1.
        // The original object constructed for c2 is left for garbage
        // collection.
        c2 = c1;

        // This line of code works regardless of whether the constructor
        // for the Type2 struct has been called to initialize the fields.
        s1.data = 1;

        // This line of code copies the data stored in s1 to s2. Value
        // semantics are used in this case.
        s2 = s1;

        // This line outputs the value stored on the heap and referred to
        // by the class reference
        Console.WriteLine( c2.data );

        // This line outputs the value stored on the stack within the struct.
        Console.WriteLine( s2.data );
    }
}

class Type1
{
    public int data;
}

struct Type2
{
    public int data;
}

```

Figure 4. Structured Types in C#

Another curious feature of the C# language is that all structured types may encapsulate methods. While not shown in the previous example, methods could have been created inside both `Type1` and `Type2`. This differs significantly from other languages which implement `struct` as a collection of data fields. It is very interesting to note that a `struct` in C# may define constructors, methods and properties. In fact, a `struct` may even be used to encapsulate a program entry point. The C# use of the `struct` keyword is unique. Conceptually, it is very easy for a novice programmer to understand. For an experienced C++ programmer, the structured type system of C# seems strange at first. With practice, the structured type system can be used to create highly efficient code.

Finally, it should be noted that the `string` type is an anomaly. It blurs the line between value types and reference types. The `string` type is a reference type that uses value semantics. The assignment operation for a `string` causes the reference to be updated to refer to a new instance of the original `string` that contains a copy of the original character data. Similarly, the concatenation operation creates an entirely new `string` that is a concatenation of the original data to be referenced by the variable of `string` type. This is a significant point of confusion for students, staff, and faculty members.

2.7. Parameter Passing

C# has excellent support for parameter passing. The language supports four distinct kinds of parameter passing [5]:

1. Value parameters
2. Reference parameters
3. Output parameters
4. Parameter arrays

C# introduces the modifiers `ref` and `out` to distinguish reference parameters and output parameters, respectively, from value parameters. Arrays of parameters are denoted using the `params` modifier. When parameters are passed using an array, the parameters are passed using value semantics. The test program shown in Figure 5 illustrates the use of the four kinds of parameter passing supported by C#. The use of the `ref` and the `out` modifiers both at method invocation and method declaration makes parameter passing unambiguous. This is particularly useful for first-time programmers.

2.8. Unified Type System

C# uses a unified type system such that the value of any type can be treated as an object or any interface implemented by the type. The `object` class is the base class for all types including all of the built-in types. The unified type system of C# enables conversions between value types and reference types using the concepts of boxing and unboxing. The process of boxing an object involves an implicit conversion of a value type to a reference type by copying the value of the original type into a new instance of type `object`. The process of unboxing an object involves an explicit conversion of a reference type to a value type. When unboxing an object, the unboxing operation first checks if the object instance is a boxed value of the appropriate type and then copies the value to the desired location. Boxing and unboxing operations can be used to develop elegant code. However, these conversion operations require some practice to learn and master. When used improperly, boxing and unboxing operations can lead to inefficient code. In our course, the concepts of boxing and unboxing were introduced towards the end of the programming course.

The unified type system also introduces aliases for types that sometimes lead to confusion. For example, the types `Int32` and `int` refer to the same type. The alias is used to integrate the C# language into the .NET Framework which utilizes `Int32` to represent a 32-bit integer value. Error messages presented by the command-line compiler often indicate the type used within the framework, not the type actually used by the programmer. For novice programmers, type aliases can be quite confusing.

```
using System;

class Parameters
{
    public static void Main( )
    {
        int a = 1;
        int b = 2;
        int c = 3;

        // Value parameter test
        Test1( a );
        Console.WriteLine( "Test 1: {0}", a );

        // Reference parameter test
        Test2( ref b );
        Console.WriteLine( "Test 2: {0}", b );

        // Output parameter test
        Test3( out c );
        Console.WriteLine( "Test 3: {0}", c );

        // Parameter array test
        Test4( a, b, c );
        Console.WriteLine( "Test 4: {0} {1} {2}", a, b, c );
    }

    public static void Test1( int p1 )
    {
        // The value of p1 may be read and assignments are not reflected
        // in the calling method.
        Console.WriteLine( p1 );
        p1 = 10;
        Console.WriteLine( p1 );
    }

    public static void Test2( ref int p2 )
    {
        // The value of p2 may be read and assignments are reflected in
        // the calling method.
        Console.WriteLine( p2 );
        p2 = 20;
        Console.WriteLine( p2 );
    }

    public static void Test3( out int p3 )
    {
        // The value of p3 does not need to be initialized to a value.
        // This method must assign a value to p3 prior to use.
        // Console.WriteLine( p3 ) would result in a compilation error.
        p3 = 30;
        Console.WriteLine( p3 );
    }

    public static void Test4( params object[] p4 )
    {
        // All parameters passed to this method are automatically stored
        // in an array of type object. Boxing occurs.
        foreach( int i in p4 )
        {
            Console.WriteLine( i );
        }
    }
}
```

Figure 5. Parameter Passing in C#

2.9. Rectangular Arrays

C# provides two mechanisms for building multi-dimensional arrays. Like C and C++, C# supports the notion of an array of arrays. In C#, this type of multi-dimensional array is known as a jagged array. However, C# also supports the notion of a rectangular array. Rectangular arrays are ideal for engineering applications since they allow the rapid development of code involving the use of matrices and multi-dimensional arrays. An example of a simple matrix calculation using multi-dimensional arrays is shown in Figure 6. Note that rectangular arrays provide a method named `GetLength()` to determine the size of a particular array dimension. This method reinforces object-oriented thinking by encapsulating all of the relevant data and methods into the object itself.

```

using System;

class RectangularArrays
{
    public static void Main( )
    {
        // Construct two instances of a rectangular array of integers.
        int[,] a = { {1,2}, {3,4} };
        int[,] b = { {1,1}, {2,2} };

        // Compute the sum of the rectangular arrays and store it in
        // rectangular array a.
        for( int row = 0; row < a.GetLength(0); row++ )
        {
            for( int col = 0; col < a.GetLength(1); col++ )
            {
                a[row,col] += b[row,col];
            }
        }

        // Display the new rectangular array a.
        for( int row = 0; row < a.GetLength(0); row++ )
        {
            for( int col = 0; col < a.GetLength(1); col++ )
            {
                Console.WriteLine( "a[{0},{1}] = {2} ", row, col, a[row,col] );
            }
        }
    }
}

```

Figure 6. Rectangular Arrays in C#

C# provides a clean implementation of rectangular arrays that allows students to develop powerful engineering applications without fully understanding how the data is stored in memory. Of course, for those students who are interested, they may easily examine both rectangular arrays and jagged arrays to get a deep understanding of the way in which multi-dimensional data is stored in memory. It should be noted that the C# array syntax differs slightly from C and C++. The brackets denoting the array are associated with the type, not the identifier. This syntax helps to reinforce the notion that the array is a type rather than just an attribute of a variable name.

2.10. Unsafe Code

C# provides the keyword `unsafe` to allow advanced programmers to utilize low-level programming constructs borrowed from the C language. In unsafe mode, a C# program may create and manipulate objects using pointers. Unsafe mode gives programmers the freedom to do almost anything but at a price. When working in unsafe mode, code is not managed. Memory leaks can be created and array bounds may be exceeded. At first glance, unsafe code may seem like something to avoid in a first programming course. However, it allows instructors to teach students about the detailed inner workings of the language and it helps prepare students for restricted environments such as those encountered in embedded computing.

Consider the example code shown in Figure 7. This code, when compiled with the `unsafe` option, illustrates how temporary variables are allocated on the stack and it illustrates a stack traversal using pointer arithmetic. A student that understands this code will have a sound understanding of how data is stored and how methods are called. How-

```

using System;

class UnsafeCode
{
    unsafe static int EnterNumber( )
    {
        int result;

        Console.Write( "Enter a number: " );
        result = int.Parse( Console.ReadLine( ) );
        Console.WriteLine( "\nAddress of result: {0,8:X}", (int)&result );
        HackIt( );

        return( result );
    }

    public static void Main( )
    {
        int i = EnterNumber( );

        Console.WriteLine( "HackIt( ) already knew you entered " + i );
    }

    unsafe static void HackIt( )
    {
        int x = 0;
        int *a = &x;

        Console.WriteLine( "Address of x: {0,8:X}\n", (int)&x );

        for( int i = 0; i < 6; i++ )
        {
            Console.WriteLine( a[i] );
        }
        Console.WriteLine( "\nYou entered " + a[3] );
    }
}

```

Figure 7. Unsafe Code in C#

ever, it should be noted that it is not possible to directly obtain the address or size of an object allocated on the heap, even if the address of the object is pinned. When demonstrating unsafe code, examples must be explained carefully to avoid confusion. Unsafe code, when introduced properly, can yield some very interesting discussions.

2.11. Other Language Features

C# has many other language features that have not been discussed in this paper. The many features of C# that aid in reducing programmer error also tend to aid in pedagogy. For example, the treatment of booleans, enumerations, and strings as distinct types simplifies the teaching of these important concepts. Booleans, enumerations, and strings are integrated into the unified type system. Also, the clear syntactic distinction between static members and instance members makes the use of these members easier for a novice to comprehend. For example, static members are associated with the type itself rather than all instances of the type. It makes sense that an instance of a type should not be required to access a static member.

This paper has attempted to present the ten “most significant” C# language features in the context of a first course on computer programming for an engineering curriculum. For further information on other C# language features, the reference by Hejlsberg [5] is highly recommended. When combined with the .NET Framework, the C# language provides many opportunities for teaching the fundamental concepts of computing.

3. Feedback on C#

Over the past 2 years, we have received quite a bit of feedback on the use of the C# language in our programming course. Overall, the feedback has been positive with a few exceptions. Some faculty members feel that C# is too abstract for a first programming course. Some staff members feel that C# lacks the power of C or C++. Some students simply dislike the language because of its Microsoft roots. Typically, the people most disappointed with the language selection share the following characteristics:

1. Learned another programming language first
2. Understand a small subset of the C# language
3. Have attempted to build a particular application that seemed easier to build in another language

Disappointment in such a scenario is a normal reaction. C# is not the perfect language for every application nor was it intended to be the perfect language. No programming language is suitable for all applications. At the University of Waterloo, C# is not the only language of instruction. Students receive exposure to C#, C, C++, Java, and assembly language. We believe it is very important to expose students to a wide variety of programming languages and programming environments.

Based on the result of course critiques, the overall feedback has been more positive than negative. Those people most satisfied with the language selection admit to having either no knowledge of programming or a very strong knowledge of programming. One interesting observation is that experienced programmers enrolled in the C# programming course find it refreshing to learn a different programming language¹ A survey of students in the class has shown that fewer than 10% of the students possess any experience in C# software development prior to the start of the course.

4. Recommendations

This paper does not advocate C# or any other programming language for a first course in programming. The selection of a programming language is largely a subjective matter best left to the individual instructor. For those instructors who wish to pursue the use of C# in a first course in computer programming, the following three recommendations should be taken seriously:

1. Instructors should focus on the essential programming concepts. It is impossible to teach the entire C# language in a single semester, regardless of the framework used. C# incorporates language features from 4

distinct programming languages. It is a rich and complex language.

2. The concepts of value types and reference types should be introduced early and reinforced throughout the course. The distinction between value types and reference types is subtle but very important.
3. Object-oriented design must be taught early in a C# programming course. The language enforces an object-oriented design paradigm that cannot be explained out-of-context. An “objects-early” approach to teaching is the right approach for teaching the C# programming language.

References

- [1] K. B. Bruce. Controversy on How to Teach CS 1: A Discussion on the SIGCSE-Members Mailing List. *Inroads SIGCSE Bulletin*, 36(4):29–34, December 2004.
- [2] A. Elliot Tew, W. M. McCracken, and M. Guzdial. Impact of Alternative Introductory Courses on Programming Concept Understanding. In *Proceedings of the 2005 International Workshop on Computer Education Research*, pages 25–35, Seattle, Washington, October 2005.
- [3] S. Hadjerrouit. A Constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum. In *Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education*, pages 105–107, Dublin City University, Ireland, August 1998.
- [4] S. Hadjerrouit. Java as First Programming Language: A Critical Evaluation. *Inroads SIGCSE Bulletin*, 30(2):43–47, June 1998.
- [5] A. Hejlsberg, S. Wiltamuth, and P. Golde. *The C# Programming Language*. AddisonWesley, Boston, Massachusetts, second edition, February 2004.
- [6] L. McIver and D. Conway. Seven Deadly Sins of Introductory Programming Language Design. In *Proceedings of the 1996 International Conference on Software Engineering: Education and Practice*, pages 309–316, University of Otago, New Zealand, January 1996.
- [7] Microsoft Canada. University of Waterloo’s Engineering Faculty Partners with Microsoft to Offer Students 24-7 Learning Flexibility, November 2005. <http://www.microsoft.com/canada/casestudies/uwaterloo.msp>.
- [8] C. Stephenson and T. West. Language Choice and Key Concepts in CS1. *Journal of Research on Computing in Education*, 31(1):89–95, September 1998.

¹ “ECE 150: Introduction to Computing” is a compulsory course in the Department of Electrical and Computer Engineering at the University of Waterloo regardless of a student’s programming background or experience.