The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

AN ENGINEERING APPROACH TO PERL AND RUBY OBJECT ORIENTED PROGRAMMING LANGUAGES

A Thesis in

Computer Science and Engineering

by

Angeline Brown

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2012

The thesis of Angeline Brown was reviewed and approved* by the following:

Mahmut Kandemir
Professor of Computer Science and Engineering
Thesis Advisor

Yuan Xie
Associate Professor of Computer Science and Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School

ABSTRACT

This study compares two object oriented programming languages, Perl and Ruby. Perl and Ruby are both extensively used scripting languages that are applied in a wide scope of modern software applications. Applications that use Perl are Amazon.com, TicketMaster.com, and Priceline.com. Applications that use Ruby are Twitter.com, YellowPages.com, and LivingSocial.com.

The purpose of discussing these languages is two-fold. First, the available quantitative comparison of these languages is limited. This paper provides essential quantitative as well as qualitative analysis that can be used as a tool for future references. Secondly, this work was performed as a trade study for the implementation of a real world database application known as PennData SpecEd Application. PennData SpecEd is used to collect records of special education students from the Commonwealth of Pennsylvania student census.

This comparative study focuses first on the performance and ease of programming software components using object oriented features of both Perl and Ruby. In addition to the SpecEd application, several variants of sorting, searching, and text manipulations are written in both languages and benchmarked to understand the performance benefits of either language. While this is not an exhaustive study of all language features, the analysis offers insight to practitioners seeking to understand the applicability of the two languages in their own applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I give all praises to my Lord and Savior Jesus Christ. Without God, I would be nothing and I am so glad I know Him and trust Him.

I would like to acknowledge the Pennsylvania State University Applied Research Laboratory and John Groenveld for their everlasting support.

I would like to thank my committee members and mentor, Dr. Yuan Xie, Dr. Mahmut Kandemir, and Dr. Kevin Irick.

A special thanks to my mom, Linda Brown, for her prayers. Mom, I love you so much, you are my hero. I would like to thank my brothers, Mario and Charles, for their tremendous support and love. To my sisters-in-law, Florence and Kai, my nieces, Amaya, Madison, and Kehinde, I love you very much. To my aunts and uncle, Shirley (Woodrow Jr.) and Gloria, I love you and I thank you for your support. To my cousins, Andrea, Woodrow IV, LaSondra, Greg, Karleetha, Jamie, and Jamal, I thank you and I love you dearly. To my baby cousins, Jordyn, Browni, Woodrow V, and Layla, I love you very much. To all of my family and friends near and far, I love and thank you for the encouraging phone calls, emails, and letters.

I would like to thank my church family, Metropolitan M.B. Church and my late Pastor, Rev. Trueville Black. I love you and I thank you for encouraging me throughout my journey. A special thanks to Dr. Yolanda Butler and Mrs. Bertha Blackburn. I smile knowing you are in my corner cheering for me and supporting me.

I would like to thank my church family here in State College, PA. Unity Church of Jesus Christ, I appreciate you and I am forever grateful. A special thanks to the Greater

Fairview church family in Jackson, MS. Thank you Greater Fairview and Mrs. Lannie Spann McBride for your continuous prayers and support.

I dedicate this thesis to my late grandparents, Mr. & Mrs. James and Bessie Brown. I miss you so much and I cherish the special memories we shared together.

Chapter 1

Introduction

Perl was created by Larry Wall in 1987[2]. Larry Wall used Perl as a bug reporting scripting language [2]. Scripting language or interpretive language programs' source code can execute directly without the use of a compiler. Compilers are used to generate programming languages' source code into machine or computer language. For example, compile programming languages' source codes are converted to machine code before becoming an executable code. Most scripting languages' source codes are executable and they do not need to be generated into machine or computer language.

Perl is a high level programming language [2]. High level languages are languages that resemble human languages. Low level languages are closer to the hardware instructions and require more attention to computer details (i.e. registers, stacks, pointers) [8]. Perl is also a dynamic language [2]. Dynamic type languages are languages that demonstrate different behaviors at run time as opposed to compile time [4]. For example, type errors can be seen after executing the program. Some behaviors in a static language are opposite of dynamic languages. Type errors in static language are detected at compile time and must be resolved before being generated into an executable code [4].

Since its creation, Perl has become popular for its Common Gateway Interface integration and parsing capabilities [2]. Common Gateway Interface (CGI) is the bridge between a web server and webpage. The CGI gives user the ability to interact with a webpage without having access to the web server [9]. Facebook, a social online network, uses CGI scripts so that users can access and modify their page without having access to the Facebook server. Perl‘s community consist of over 300 groups worldwide [17].

Yukihiro Matsumoto is credited for creating Ruby in 1995 [7]. Yukihiro Matsumboto, known as Matz, created Ruby because he was not fully satisfied with other languages [7]. Ruby is a complete object oriented language, dynamic and scripting language [7]. Object Oriented (OO) languages use a collection of objects that relate to each other logically to form a *class* [4]. More about Object Oriented languages and features are discussed in chapter three. Ruby draws some of its inspirations from Perl [7]. Since its creation, Ruby has become popular for its web application framework Ruby on Rails [11]. Ruby on Rails is discussed in chapter five.

Both Perl [2] and Ruby [7] have core class libraries that include a powerful Application Programming Interface, known as API. An API is a manual for programming languages as it describes how tasks are performed [1]. For example, if a programmer needs documentation for a particular language’s function, then the programmer can check with the programming language’s API.

This paper is organized for readers to gain an understanding of both languages, their utilities, and capture their valuable contributions to programming. In order to provide effective research, a number of applications examples are discussed as case studies. Examples of these case studies use both languages and serve as illustrations of using both Perl and Ruby in moderate scale software applications.

Chapter 2 compares standalone algorithms implemented in Perl and Ruby. These algorithms are benchmarked to compare execution time and average memory usage. More information about the Penn Data Special Education Application is given in the Chapter 3, along with the research methodology. A brief review of object oriented concepts and features are discussed in Chapter 4. For detailed coverage on OO languages, the book Object Oriented Programming Languages Principles and Paradigms by Allen Tucker and Robert Noonman is offered as a reference. Various program examples that highlight object oriented aspects of Perl and Ruby are discussed in Chapter 5. In Chapter 6, the topic Database Independent Interface also known as DBI is discussed. Chapter 7 discusses web design frameworks. Catalyst and Ruby on Rails are introduced in this chapter. In Chapter 8, the conclusion is given.

Chapter 2

Algorithms

In this chapter, algorithms and text extraction methods are written in both Perl and Ruby and benchmarked to compare execution times in seconds and the percentage average memory usage. The standalone algorithms used are Insertion Sort, Quick Sort, and Binary Search. The number of iterations done for each algorithm program was 10,000 times. The text extraction functions include regular expression pattern matching techniques and parsing techniques. The number of iterations done for the text extraction programs was 1000 times. The command *time* was used to collect the execution time for each algorithm being performed.

2.1 Insertion Sort

The insertion sort algorithm takes a sequence of random values known as an array, and sorts the numbers in increasing or decreasing order [6]. The array size and values are given by the user as an input into the insertion sort algorithm. The output order is also determined by the user. Insertion sort performs better on small array sizes compared to larger array sizes.

Insertion sort is implemented by comparing the first input with the second input. The lowest number value is placed in the first position. The second position

is compared with the third position and the lowest value receives the second position. A continuation of comparing the values in this manner is done until the last position is evaluated in the sequence of values. The last position should be the highest value in the sequence. Figure 2-1 shows the Perl's code for insertion sort [6]. Figure 2-2 shows the Ruby's code for insertion sort [6].

Insertion Sort Algorithm implemented in Perl

```
sub insertion_sort {
my ($array, $i, $j, $key) = @_;
   for $i (1 ..$arr)
    {
      $key = $array[i];
        for ($j = $i; $j >= 0 && $array[$j - 1] > $key;
         $j = j-1   )
         $ array[$j+1] = $array[$j]
       }
      $array[$j+1] = $key;
    }
@a
}
```

Figure 2-1 Perl's Algorithm for Insertion Sort

Insertion Sort Algorithm implemented in Ruby

```
def insertion_sort(array)
  1.upto(length - 1) do |i|
    key = array[i]
    j = i - 1
    while j >= 0 and array[j] > key
      array[j+1] = array[j]
      j = j -1
    end
    array[j+1] = key
```

Figure 2-2 Ruby's Algorithm for Insertion Sort

The execution time of insertion sort depends on the size of the array. For example, an array with 10 values takes less time than an array with 100 values, given that the arrays are not previously in sorted order. Four data sets were tested using insertion sort implementations for Perl and Ruby. The data sets included four different unordered array sizes. The first size of the array was a sequence of 0 – 256, the second array was 0 – 512, the third array was 0-1024, and the fourth array was 0-2048. Figure 2-3 shows the execution time in seconds of insertion sort implemented in Perl and Ruby. The blue column is Perl's running time in seconds and the red column is Ruby's running time in seconds.

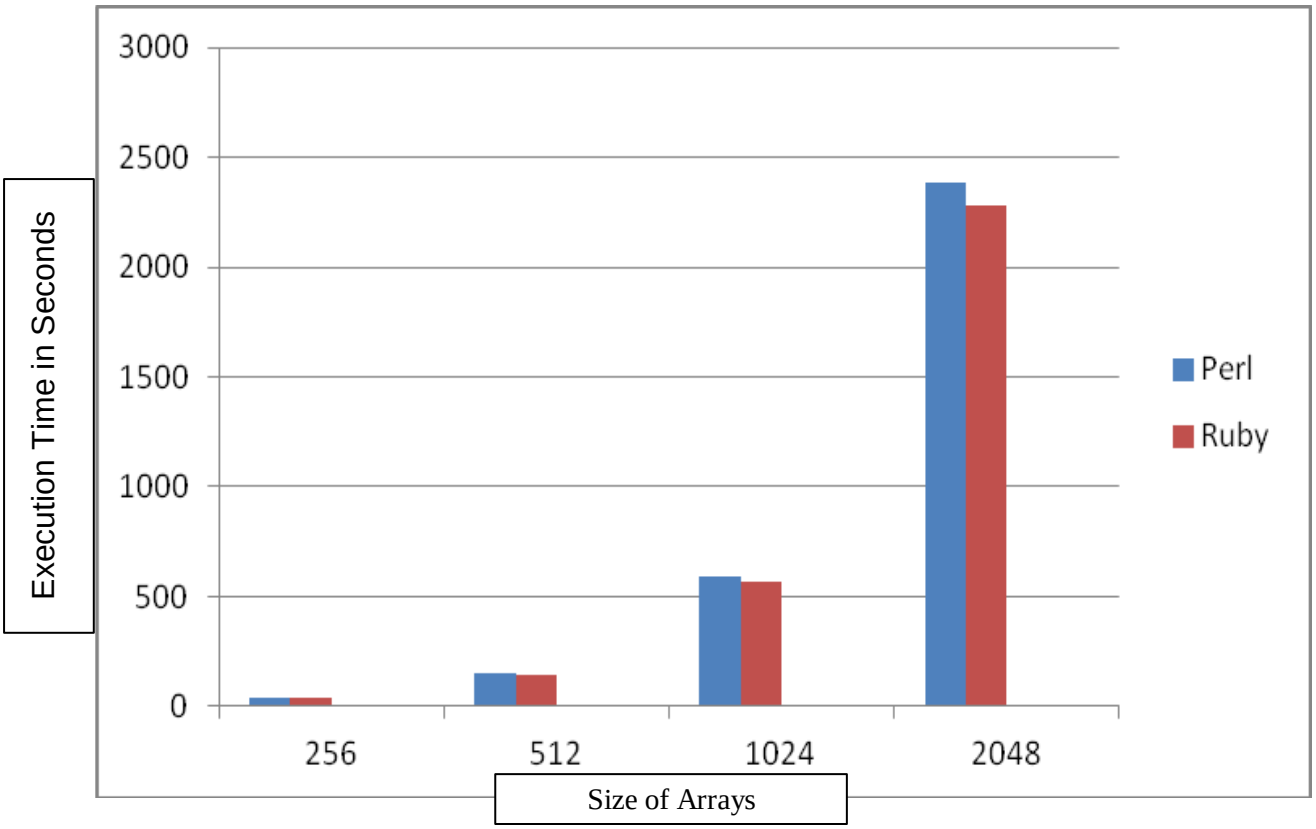

Figure 2-3 Execution times in seconds for Insertion Sort

2.2 Quick Sort

The quick sort algorithm takes a sequence of random values known as an array, and sorts the numbers in increasing or decreasing order [6]. The array size and values are given by the user as an input into the quick sort algorithm. The output order is also determined by the user. Quick sort performs better on larger array sizes compared to smaller array sizes.

Quick sort is implemented by dividing the array into two smaller arrays. The division is done by choosing a random position in the array called a pivot. The values in the array are compared to the pivot and the values less than the pivot are placed in the first group. The values greater than the pivot are placed in the second group. Values that are equal to the pivot can go to either group. After the values are compared to the pivot and placed into groups, then the values in each group are sorted recursively. Quick sort is completed when the last element of the first group is less than or equal to the first element of the second group. Figure 2-4 shows the Perl's code for quick sort [6]. Figure 2-5 shows the Ruby's code for quick sort [6].

Quick Sort Algorithm implemented in Perl

```
sub quicksort {
    my ($list, $p, $r) = @_;
    if ($p < $r) {
        my $q = &partition(\@$list, $p, $r);
        &quicksort(\@$list, $p, $q - 1);
        &quicksort(\@$list, $q + 1, $r);
    }
}
sub partition {
    my ($list, $p, $r) = @_;
    my $pivot = $list[$r];
    my $i = $p - 1;
    for (my $j = $p; $j < @$list - 1; $j++) {
        if ($list[$j] <= $pivot) {
            $i++;
            ($list[$i], $list[$j]) = ($list[$j], $list[$i]);
        }
    }
    ($list[$i + 1], $list[$r]) = ($list[$r], $list[$i + 1]);
    return $i + 1;
}
```

Figure 2-4 Perl's Algorithm for Quick Sort [6].

Quick Sort Algorithm implemented in Ruby

```
def quicksort(list, p, r)
    if p < r then
        q = partition(list, p, r)
        quicksort(list,  p,  q-1)
        quicksort(list, q+1, r)
    end
end

def partition(list, p, r)
    pivot = list[r]
    i = p - 1
    p.upto(r-1) do |j|
        if list[j] <= pivot
            i = i+1
            list[i], list[j] = list[j],list[i]
        end
    end
    list[i+1],list[r] = list[r],list[i+1]
    return i + 1
end
```

Figure 2-5 Ruby's Algorithm for Quick Sort [6].

The execution time of quick sort depends if the sub-arrays are balanced or unbalanced [6]. For example, a sub array that is not balance will take more time than two balanced sub arrays. Four data sets were tested using insertion sort implementations for Perl and Ruby. The data sets included four different array sizes. The first size of the array was a sequence of 0 – 256, the second array was 0 – 512, the third array was 0-1024, and the fourth array was 0-2048. Figure 2-6 shows the execution time in seconds of insertion sort implemented in Perl and Ruby. The blue column is Perl's running time in seconds and the red column is Ruby's running time in seconds.

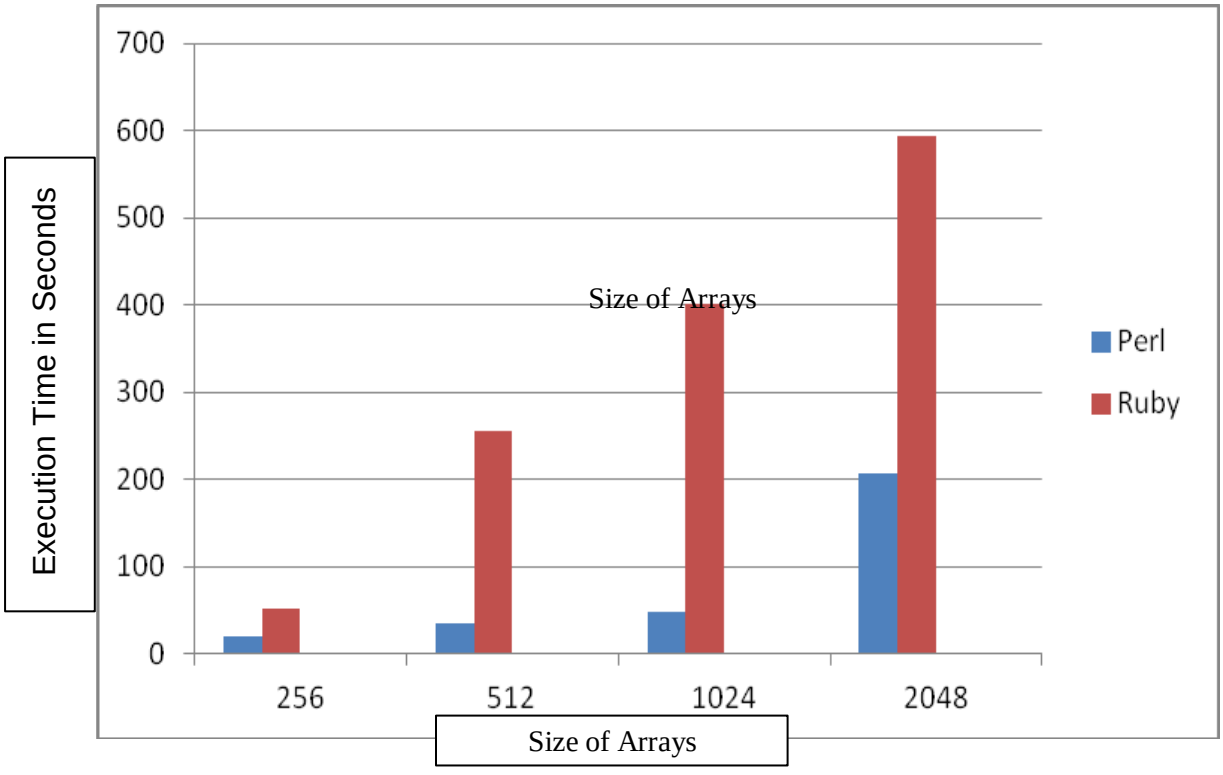

Figure 2-6 Execution times in seconds for Quick Sort.

2.3 Binary Search

The Binary Search algorithm finds the position of a specific element in an array. The specific value is given by the user [6]. The binary search algorithm performs only on sorted sequences [6]. Binary Search chooses a median value, that value is result of the sequence length divided by 2. For example if there are ten values in the sequence, the median is five. This value is used as the position in the sequence. For example, the median five is now used to locate the fifth position.

The value in that position is compared to the specific element given by the user. If the value is less than the compared element then binary search removes the compared value and all values to less than that element. For example, if the specified is element number 7 and the median position's value is number 4, then all the values preceding number 4 is removed. Since the list is in sorted order, the values preceding number 4 are not needed. Binary search then chooses another median value by dividing the rest of the sequence in half. That value is used as the new position in the sequence. The specified element is compared to the value of the element that is in the median's position and the process is complete once there is a match. Figure 2-7 shows the Perl's code for binary search [6]. Figure 2-8 shows the Ruby's code for binary search [6].

Binary Sort Algorithm implemented in Perl

```
sub binarysearch {
  my ($array, $elem, $low, $high) = @_;
  if ($high < $low) {
    return 0;
  }
  my $middle = ($high + $low) / 2;
  if ($array_ref->[$middle] == $elem) {
     return $elem;
  }
  if ($elem < $array ->[$middle]) {
    binarysearch($array, $elem, $left, $middle - 1);
  } else {
    binarysearch($array, $elem, $middle + 1, $right);
  }
}
```

Figure 2-7 Perl's algorithm for Binary Search [6].

Binary Search Algorithm implemented in Ruby

```
def binarysearch(arr, elem, low, high)
  mid = low+((high-low)/2).to_i
  if high < low
    return nil
  end
  when arr[mid] > elem
    then search(arr, elem, low, mid-1)
  when arr[mid] <elem
    then search(arr, elem, mid+1, high)
  else
    return mid
  end
end
def search(list, value)
  return binary_search(list, value, 0, a.length)
end
```

Figure 2-8 Rub's Algorithm for Binary Search [6].

The execution time of binary search depends on the size of the array and the number of misses performed after each comparison. For example, a large size array taking multiple times to match the specific element will result in a longer wait period. Also, if the number being match is not in the array, then the algorithm will also result in a longer wait period. Four data sets were tested using binary search implementations for Perl and Ruby. The data sets included four different array sizes. The first size of the array was a sequence of 0 – 256, the second array was 0 – 512, the third array was 0-1024, and the fourth array was 0-2048. Figure 2-9 shows the execution time in seconds of insertion sort implemented in Perl and Ruby. The blue column is Perl's running time in seconds and the red column is Ruby's running time in seconds.

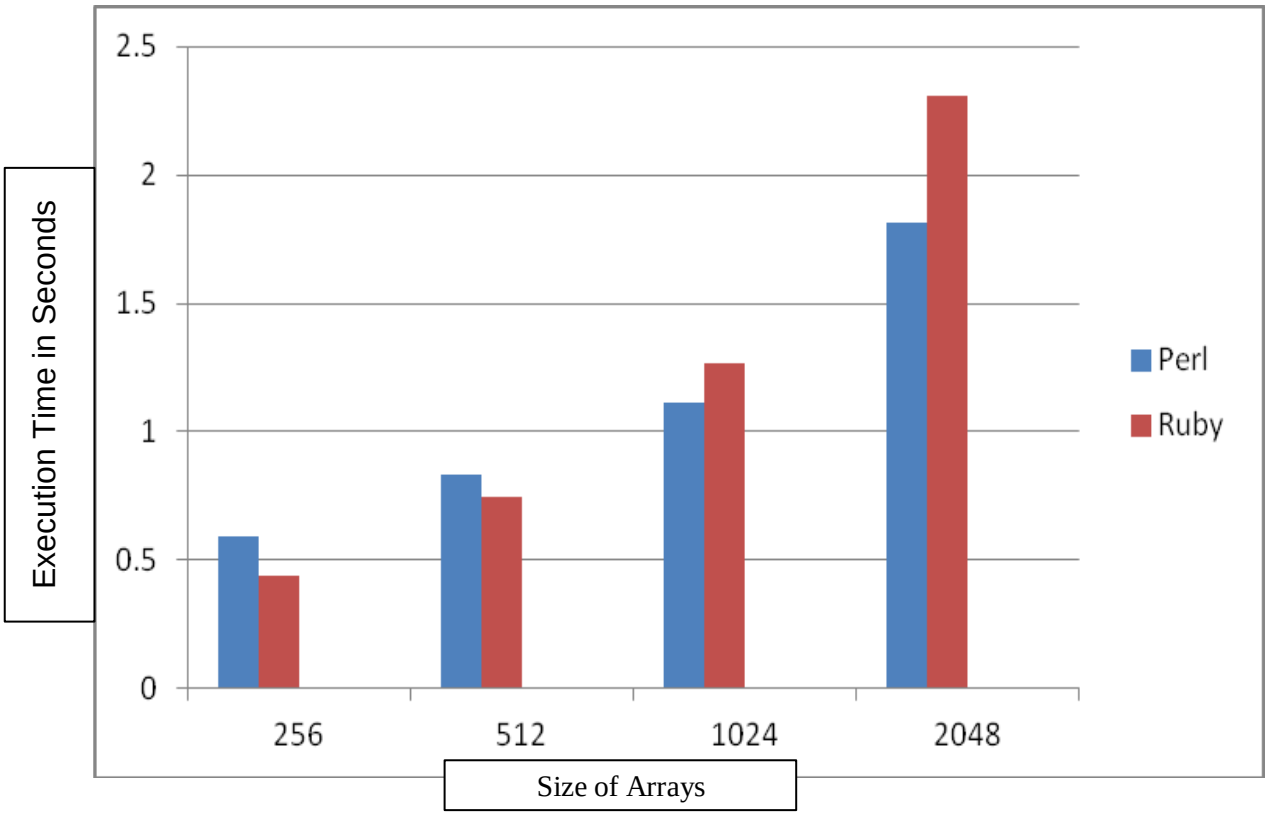

Figure 2-9 Execution times in seconds for Binary Search

2.4 Regular Expressions

Regular expression in computer programming is matching specified expressions such as words, characters, or patterns within a given text file [2]. Regular expression is similar to using the find feature in a text file such as Microsoft Word to immediately find a specific character or word. Figure 2-10 shows Perl's code for regular expression [2]. Figure 2-11 shows Ruby's code for regular expression [7].

Pattern Matching in Perl

```
sub Program {

my $file = $ARGV[0]; # file that the data comes from
my @doc;             # Array of all the lines in the file
my $counter=0;

open INPUT, '<', "$file" or die "The file couldn't be opened : $!";

@doc = <INPUT>;

for my $line (@doc) {

  chomp($line);
  if ($line =~ m/WARNING:(.*)./)
   {
      print $line;
     $counter++;
    }
print "Found $counter times.\n";
}
close INPUT;

1;
```

Figure 2-10 Perl's algorithm for regular expression

Pattern Matching in Ruby

```
def express
count = 0
filename = 'patternmatch.txt'
 file = File.open(filename, 'r')

while (line = file.gets)
        if line.match("WARNING:")
        puts line
   count = count +1
   end
end
puts count
end
```

Figure 2-11 Ruby's algorithm for regular expression

The Regular expression programs were used to find all the "WARNINGS" in an example text file and output the warning messages into a separate file and include the number of times a warning message was found. Three separate files with different numbers of lines in each program were used to test the time of Ruby and Perl [2, 7]. The number of lines each set was 161999, 97316, and 79898. Figure 2-12 displays the execution time in seconds of implementing of regular expression pattern matching Perl and Ruby. The blue column is Perl's running time in seconds and the red column is Ruby's running time in seconds.

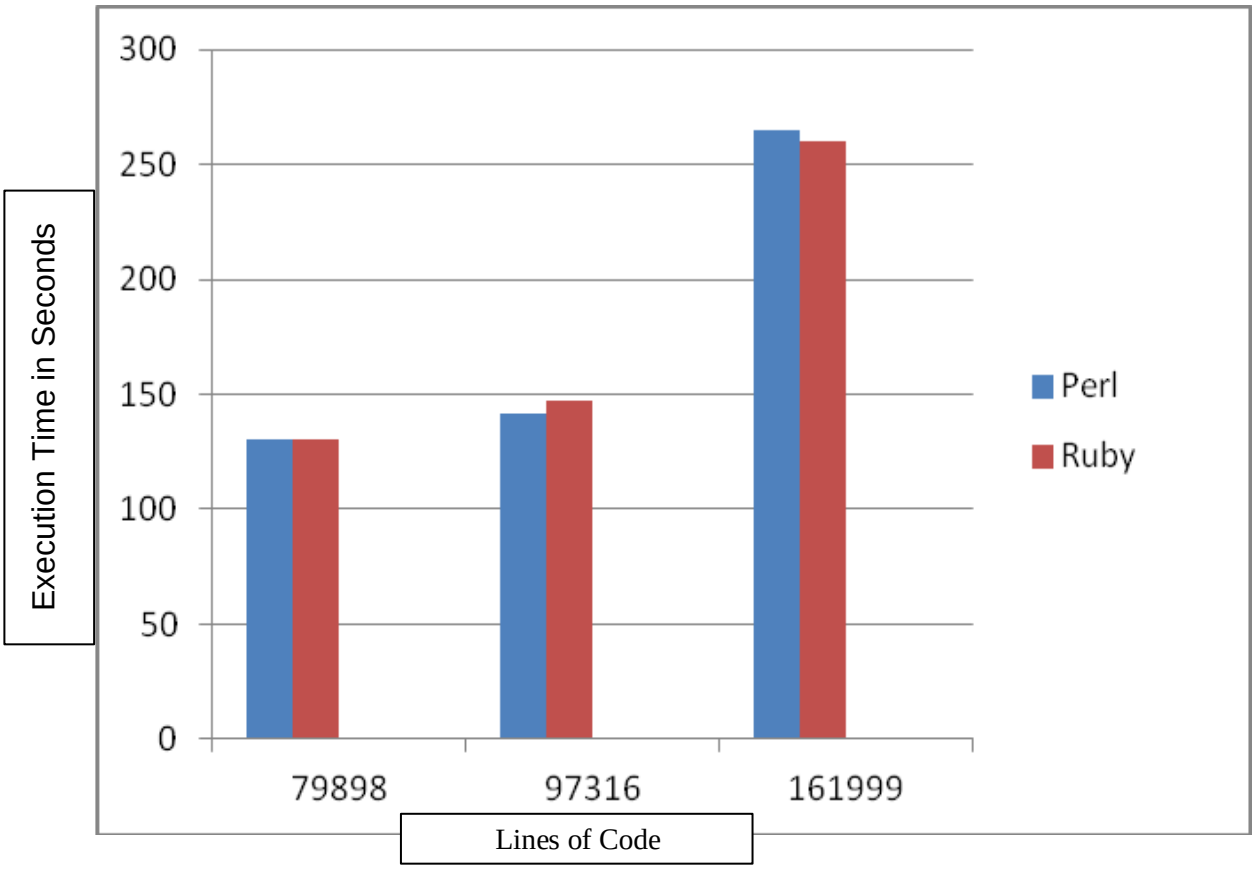

Figure 2-12 Execution times in seconds Pattern Matching

2.5 Parsing

The parser function processes a text file by scanning through text and parsing the data using a given string or character. For example, parsing through a CSV file using a comma will separate the text by commas and producing a more legible file. Figure 2-13 shows Perl's code for parsing data [2]. Figure 2-14 shows Ruby code for parsing data [7].

Parsing Data in Perl

```
my @doc;            # Array of all the lines in the file
my @parsed;         # Array of all the parsed string

open INPUT, '<', "$file" or die "The file couldn't be opened : $!";

@doc = <INPUT>;

foreach my $line (@doc) {
  chomp($line);

  # split the entry using the ',' deliminator
  @parsed = split(/,/, $line);

   }
close INPUT;
1;
```

Figure 2-13 Perl's algorithm for parsing function

Parsing Data in Ruby

```
filename = 'data_to_parse.txt'
 file = File.open(filename, 'r')

 file.each_line("\n") do |values|
   parse = values.split(",")

end
```

Figure 2-14 Ruby's algorithm for parsing function

The parsing programs implemented in Perl and Ruby was used to separate a CSV file. Three separate files with example students from the SpecEd Application were used to test the execution time of Ruby and Perl [2, 7]. Each student had 32 individual records that required separation by a comma. The number of students for each set was 3695, 7390, and 14780. Figure 2-15 displays the execution time in seconds of Perl and Ruby's parsing function on each data set. The blue column is Perl's running time in seconds and the red column is Ruby's running time in seconds

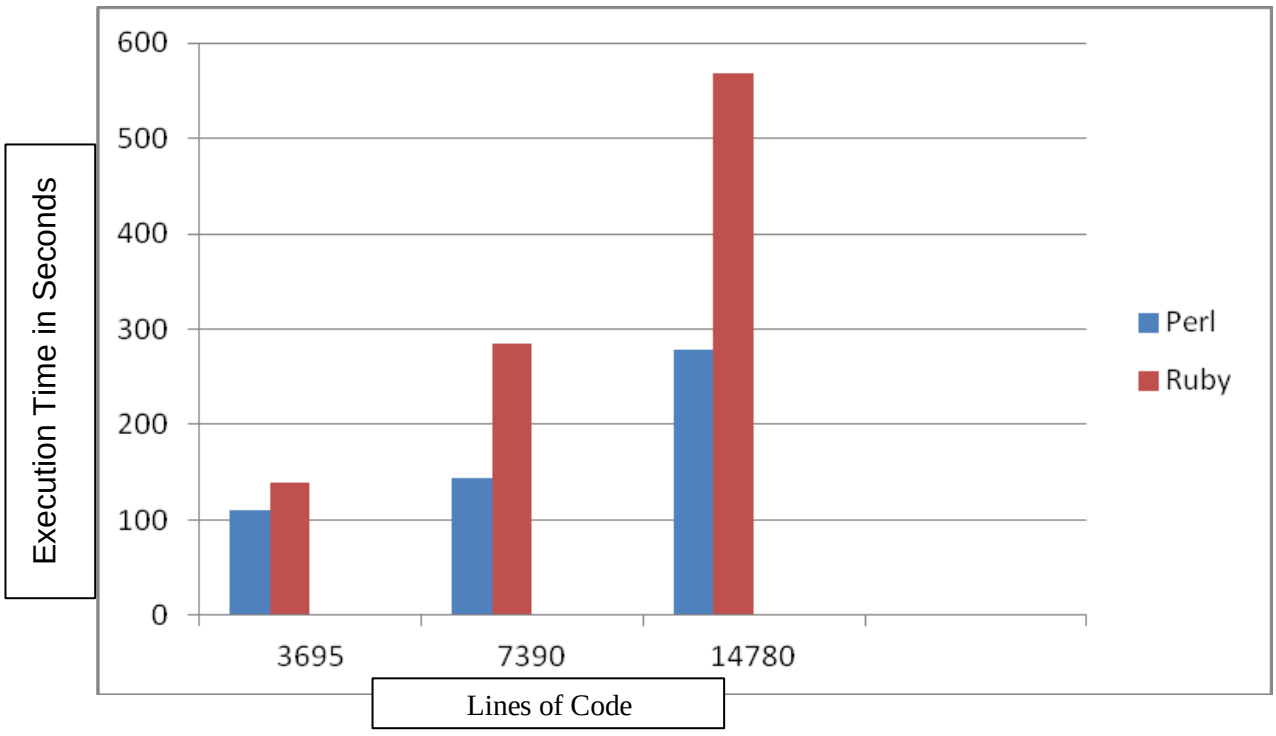

Figure 2-15 Execution times in seconds Parsing Function

2.6 Memory Usage

The benchmarks for the algorithms and test extractions with the highest data set were run consecutively to ensure the same amount of memory and CPU time for each test. The command *top* was used to collect memory usage for each algorithm being performed. Figure 2-16 shows the percentage memory usage for each benchmark performed by Perl and Ruby.

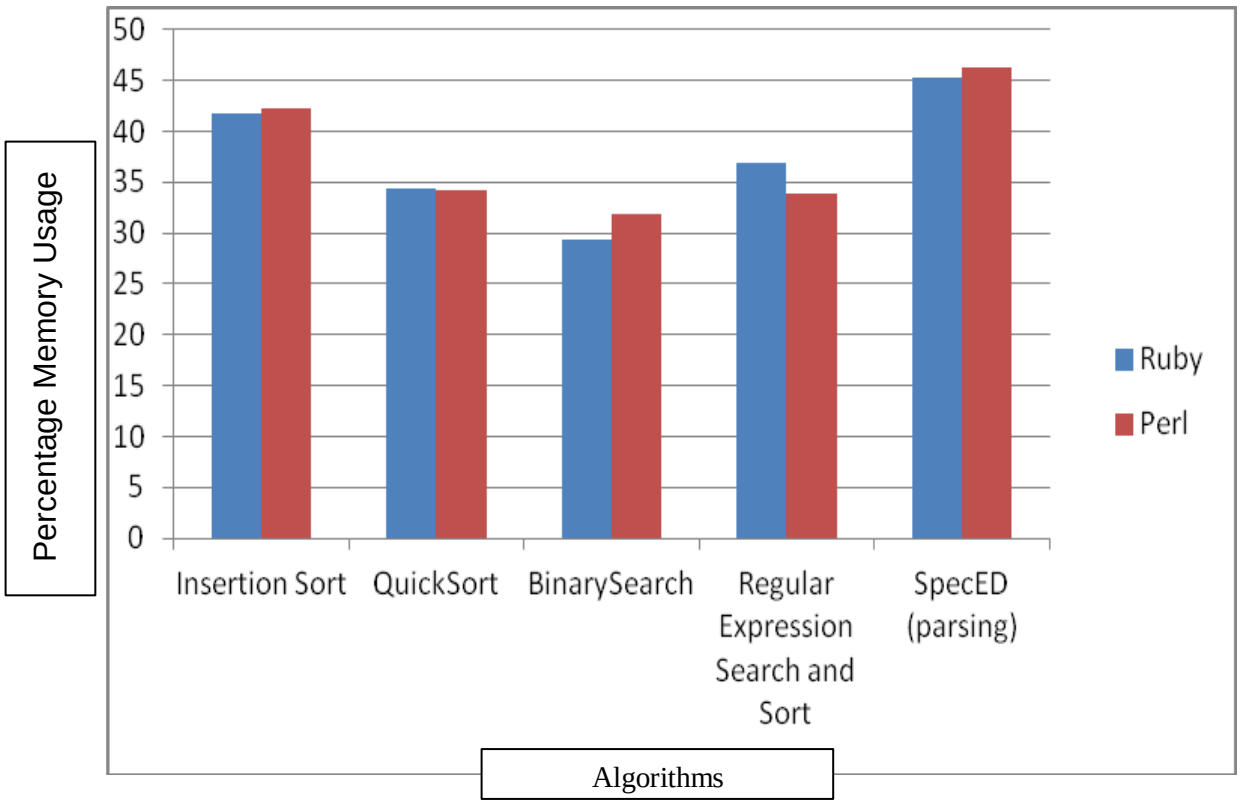

Figure 2-16 Percentage memory usage for Perl and Ruby benchmarks

2.7 DTrace

DTrace is a tracing framework used to analyze the behavior of programming languages [20]. DTrace is used to examine the performance of Perl and Ruby programming languages and identify specific time measurements within the program. DTrace provides information such as CPU time, memory usage, object creation times, and latency [20]. For example, the quick sort execution times for Perl and Ruby in figure 2.7 are significantly different. The DTrace tracing framework is used to investigate Perl and Ruby's programming algorithms and locate what is responsible for latency within the program.

Scripts were written using DTrace to log which method was responsible for latency and if garbage collection occurred. Quick sort algorithm includes two main methods. The first method is quick sort procedure which sorts the entire array [6]. The second method is the partition procedure which rearranges the sub array in place [6]. Figure 2-17 illustrates the DTrace analysis on quick sort algorithm.

Figure 2-18 illustrates DTrace on the parsing algorithm. DTrace is used in the Perl and Ruby algorithms to identify the time to read in the data from the given file and the time to parse the data in the file. DTrace is also used to determine if the garbage collector is used while executing the Perl and Ruby programs for parsing data.

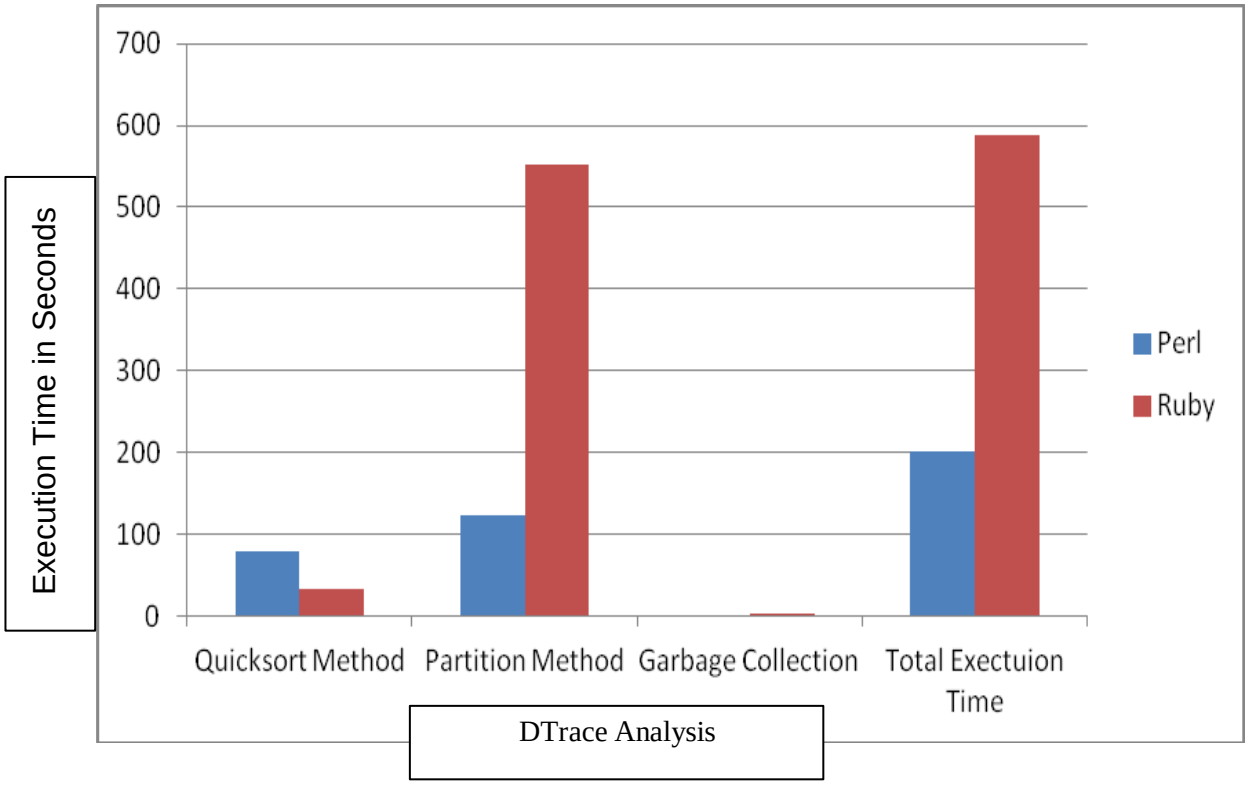

Figure 2-17 DTrace Analysis on Quick Sort

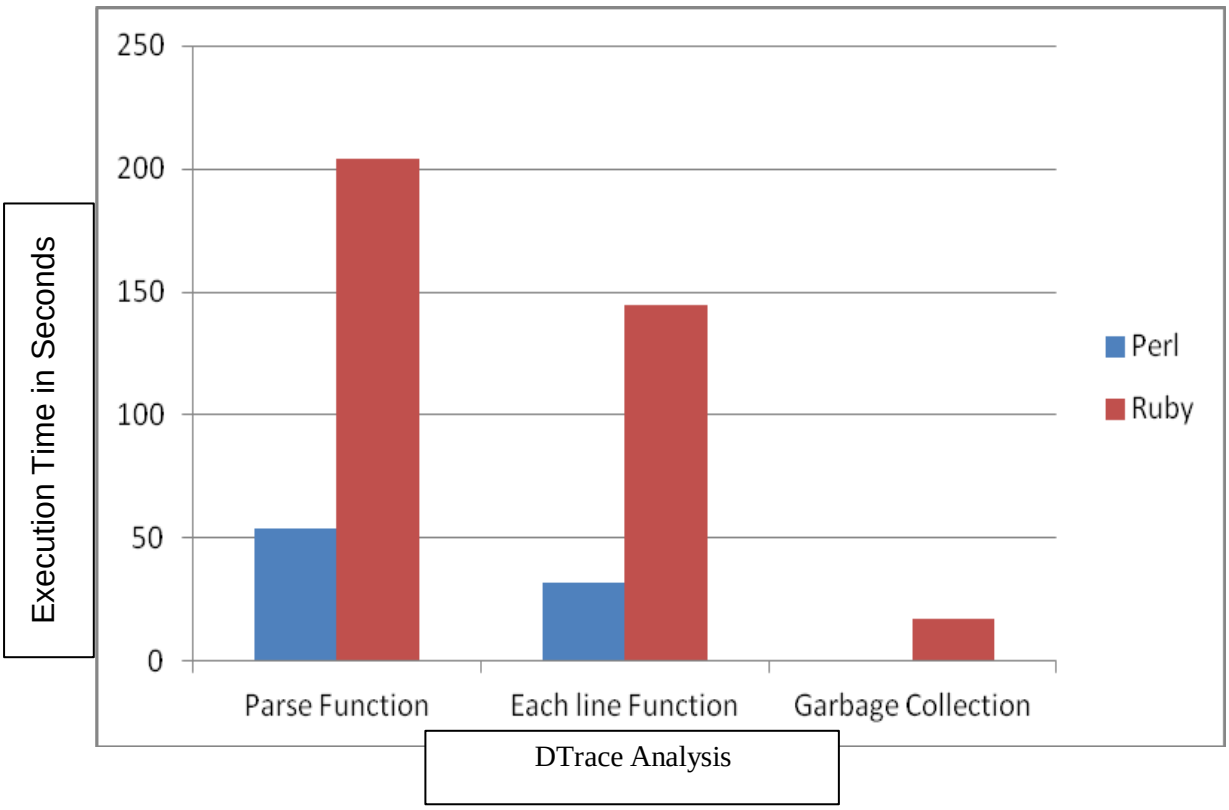

Figure 2-18 DTrace Analysis on parsing function

Chapter 3

SpecEd Application

The implementation of an application interface to the PennData Special Education database is used as a case study. PennData SpecEd is a desktop and web application. This application is used to keep records of special education students from the Commonwealth of Pennsylvania student census. PennData's web interface is used by Commonwealth officials to enter their summary reports for Special Education students. The Commonwealth officials are known as Intermediate Units (IUs) and there are thirty one IUs representing the Commonwealth's 501 school districts. Records include the sex, gender, disabilities, and special education environments for each special education child. The public can use the site to retrieve information about each school in the Pennsylvania area. The data obtained does not show each child's record but a collective data set by school.

PennData SpecEd reporting system began in 1998. A cluster of HP minicomputers were used to run PennData. The web server used to host PennData was Apache and the Operation System was Windows NT. Perl CGI scripts ran on the Windows NT server.

PennData faced a number of problems in the beginning stages. The early implementation of Apache for Win32 was single process and single threaded. Multiple clients could not concurrently access the CGI scripts to upload their data and would thus timeout.

The second problem was poor connectivity with the Intermediate Units and lack of student records. As the year 2000 approached, the minicomputers were found not to be Y2K compliant. The computers were also loosing vendor support.

Progress was made in the following years. Intermediate Units are able send their individual data files and summary reports as Comma-Separated values via snail mail. Presently, the UNIX operating system, Solaris is the host for the web interface which handles the thirty one Intermediate Units uploading their data simultaneously, while continuing to run the database server on Oracle on Microsoft Windows server. Table 3-1 shows a description of the server specifications.

System Configuration	Specifications
PennData Web Interface – Solaris Systems	
System Architecture	Oracle Corporation i86pc
Memory Size	8192 Megabytes
PennData Server – Linux Systems	
System Architecture	Sun Microsystems SUN FIRE X4150
BIOS Configuration	American Megatrends Inc. 1ADQW062 09/10/2009
Baseboard Management Controller (BMC)Configuration	IPMI 2.0 (KCS: Keyboard Controller Style)
Processor Sockets	
Version	**Location Tag**
Intel(R) Xeon(R) CPU	E5345 @ 2.33GHz CPU 1
Intel(R) Xeon(R) CPU	E5345 @ 2.33GHz CPU 2

Table 3-1 Server specifications for SpecEd Application

3.1 Perl and Ruby applied to SpecEd

Presently, SpecEd Data consists of over 20,000 records; table 3-2 illustrates the records/IU. Each record is less that 1KB of data. Files are checked via Perl scripts and through the use of a client-server GUI written in Perl. The performance for SpecEd is not a critical factor. The files processed and checked that are done by the desktop application do not cause task latency to become a critical factor.

Since, this is the case, scripting languages are suitable to use. Compiled languages are efficient for applications that have critical performance because they usually run faster [8]. Scripting languages also provide an ease of programming [7]. Using the appropriate Perl and Ruby libraries is critical. In order to eliminate erratic behavior, the correct Perl modules and correct versions of Perl must be packaged with the desktop app.

The web interface was revisited and Ruby on Rails was chosen to implement the web application because of its ease of development and the ability to interface with Perl code that validated the files. The correct version of the Rails Gem is needed for running the web application since Rails tends to differ quite a bit from version to version.

Perl and Ruby allow SpecEd to be a cross platform application. As long as the client's machine has the correct Ruby Gem and Perl versions, the software can be ran on any platform.

IU	TOTAL
2	6190
18	7195
7	7339
19	7825
4	8385
8	8990
1	9232
21	9819
5	10274
14	12691
20	13276

IU	TOTAL
31	407
9	2283
11	2886
30	3224
28	3668
29	3896
10	4754
27	4783
6	5031
16	5542
17	5582

IU	TOTAL
12	13930
25	13979
13	14171
15	14447
24	14578
22	15944
23	18038
3	19146
26	27441

Figure 3-2 Records per Intermediate Unit

3.2 SpecEd applied to Research

Using SpecEd as a case study, Perl and Ruby implementations are compared and analyzed based on performance and ease of programming. Figure 3-1 illustrates the SpecEd application layout.

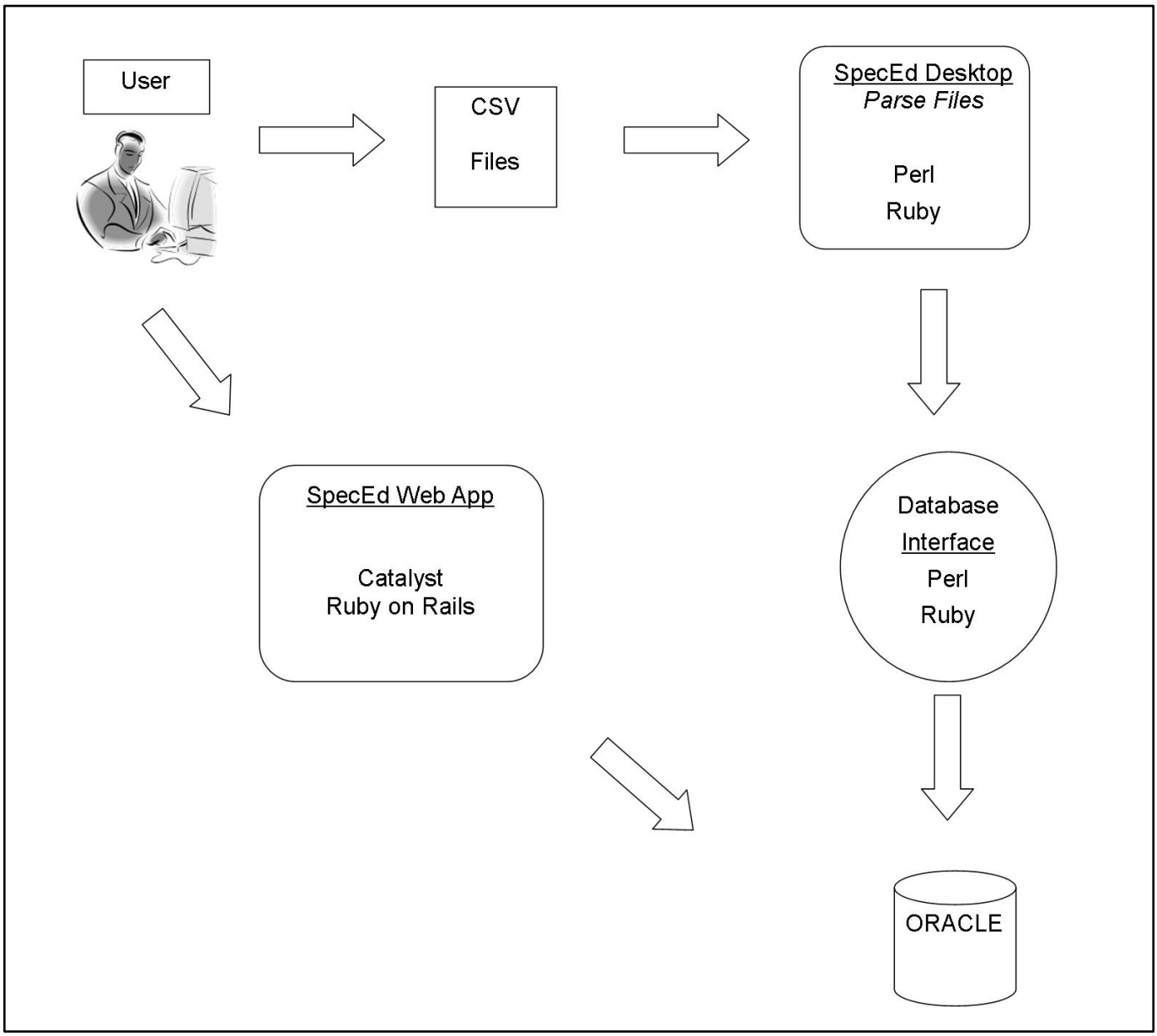

Figure 3-1 SpecEd Application Layout

The parsing functions for Perl and Ruby are programmed and analyzed. Chapter 2 gives more details on the programming differences between Perl and Ruby. The next step is populating the files into the database. The database interface (DBI) is the medium between Perl and Ruby and the database. Chapter 3 gives an overview of the DBI and how Perl and Ruby programming languages connect to a desired database. As previously mentioned earlier in this chapter, the web interface is used to access data. Perl and Ruby uses the model-view-controller (MVC) framework to build web applications. Chapter 5 expounds the Perl and Ruby web applications.

Chapter 4

Object Oriented Programming

The object oriented model uses classes [4] which are objects grouped together based that share a common relationship. This relationship that objects form is then used as programs to solve software applications. An object in an object based language has two characteristics [4]. The first characteristic is the object's state which is known as variables or attributes. The variables describe the data stored in the object [4]. The second characteristic is the behavior commonly referred to as *methods*. The *methods* change the state of the object through message passing [4].

The concept of using classes comprised of objects to program software applications makes OO paradigm different from other programming paradigms [4]. There are three other distinct programming paradigms that other programming languages are patterned by [4]. These paradigms are imperative programming, functional programming, and logic programming.

Imperative paradigm's state uses variables that have only values [4]. Commands that operate on this state are conditional terms such as for, if, while, and else. Functional programming resembles computational problems. An example of a functional programming is the factorial program [4].

Logic programming also uses mathematical logic for computer programming [4]. Logic programming are considered rule based programs because follow a set of rules that declare the outcome of a program [4]. An example of the Logic programming concept is the modus ponens logic operation which state if A then B [4].

The chart in table 4-1 gives a comprehensive comparison of how Perl [2] and Ruby [7] native languages contain the OO features. Inheritance, Polymorphism, and Encapsulation are three of several distinct features of Object Oriented Programming [4]. Inheritance in an OO class allows codes to be reused [4]. Classes in OO are designed with a hierarchical approach which means that a program can have a super class and sub classes. A subclass can inherit variables and methods from its parent or super class [4]. A child inheriting features from his or her parents is an example of Inheritance.

Polymorphism in the OO languages refers to an object, variable, or function having more than one implementation [4]. For example, consider the method Add. Add can be used in three different classes but each class can have a different data type. Class one can use the data type integer, class two can use the data type float, and class three can use the data type double. Depending on which method is called, the function Add will respond differently. Examples of encapsulation are classes or packages [4].

Methods and variables that relate to each other in terms of how they function in a specific program are bundled in classes [4].

OO	Perl (since version 5)	Ruby
Classes	☑	☑
Instances of Classes	☑	☑
Methods	☑	☑
Data Abstraction	☑	☑
Dynamic Typing	☑	☑
Reflection	☑	☑
Polymorphism	☑	☑
Encapsulation	☑	☑
Inheritance	☑ (Use of Moose)	☑ (Not multiple inheritance)
Information Hiding		
Automatic Garbage Collection	☑ (reference counting)	(mark and sweep)
Interpolation	☑	☑

Table 4-1 OO Characteristics used in Native Perl & Ruby Languages

The two languages are similar in reference to OO programming but there are differences. Perl has the feature of using multiple inheritances and Ruby does not [2, 7]. Multiple inheritances means a class can have more than one super class [4]. Single inheritance means a class can have only one super class [4].

Automatic garbage collection is vital in terms of memory management [4]. Garbage collection is responsible for retrieving all memory blocks used by objects that are no longer used in the program, and returning them so they can be reused by new objects [4]. Garbage collection can be done by reference counting, mark-sweep, and copy collection [4].

Perl uses the method reference counting [12]. Reference counting is done by giving each object a count, starting with zero [4]. If the object is reused the count is incremented. The object is decremented if the reference is deleted. The garbage collector only deallocates objects referenced with zero and returns them to the memory system [4].

Ruby uses the method mark and sweep [7]. Mark and sweep was the first garbage collection algorithm [4]. Mark and sweep is implemented when the memory is full [4]. The first step of Mark and Sweep is marking all the objects that are used. After marking the used objects, the garbage collector reclaims the objects that are not marked [4].

Perl was not initially designed as an object oriented language [2]. Moose is an extension of Perl's object oriented language [13, 14]. Perl's Moose was designed to make Perl resemble the Object- Oriented model and features [13, 14]. Ruby is an objected oriented language from the ground up [7]. Figure 4-1 is code implemented by Perl illustrating the three features; inheritance, encapsulation, and polymorphism. Figure 4-2 is code implemented by Ruby. The programs are written in the same manner to show the number of lines in the program and code styles used by Perl and Ruby. The programs demonstrate 1) encapsulation feature which is the classes student_name, school_name, and parent_name, 2) inheritance feature which is the subclasses inheriting from the super class 3) polymorphism feature using the method enter_name.

```
#Inheritance of classes using the symbol <
#Encapsulation example using classes and subclasses
#Polymorphism using the method enter_name
-----------------------------------------------------------------------------------------------------
#Encapsulation Create a class
package Name;
  sub new{
      my $class = shift;
      my $self = {
        enter_name => shift, };
         print “Enter name”\
             my $line = <STDIN>;
             chomp($line);
         return $self->{enter_name}=$line;

package student_name;
@ISA = (“Name”)
       print “Your name”
       my $self = shift;
   return ($self->{enter_name};

package school_name;
  @ISA = (“Name”)
      print “Your school name”;
      my $self = shift;
    return ($self->{enter_name};

package parent_name;
@ISA = (“Name”)
      print “Your parent name”
      my $self = shift;
    return ($self->{enter_name};
```

Figure 4-2 Perl Script with OO features

```
#!/usr/ruby/1.8/bin/ruby

#Inheritance of classes using the symbol <
#Encapsulation example using classes and subclasses
#Polymorphism using the method enter_name

-----------------------------------------------------------------------------------------------
#Superclass Name

class Name

#Method for getting name

  def get_ name
     #puts "Enter name”
      line = gets
      line = line.chomp
     return line
end

#Subclass to get student name

class Student_name < Name
      puts “Your name”
          def get_name
end

#Subclass to get school name

class School_name < Name
      puts “Your school name”
            def get_name
  end

#Subclass to get parent name

class Parent_name < Name
      puts “Your school name”
            def get_name
  end
```

Figure 4-3 Ruby Script with OO features

Chapter 5

Perl and Ruby

This chapter describes the implementations of Perl and Ruby as it relates to the SpecEd application. In the SpecEd application, Perl and Ruby programs are designed to parse Comma Separated Value files taken from the Intermediate Units. By parsing the files, the data is legible and separated by fields. An object class program that includes the parser algorithm and other specific methods is implemented to migrate the comma separated values into a table format.

The first object class program is written using Perl's Moose programming language. Moose allow Perl's features to resemble the Object Oriented model [13 14]. The object class consists of attributes and methods. The Moose term attributes defines variables and the term methods are functions in a class that control state [13].

The methods used in the object class program are mutator and accessor methods. Mutators and Accessors allow variables to read and written [15]. The accessor method, commonly referred to as getter, returns the value of a member variable [15]. The mutator method commonly referred to as setter, controls the changes to a variable. [15]

Figure 1-1 illustrates a Moose script that demonstrates how the mutator and accessors are used. Also shown in figure 5-1 is Moose programming style. Creating a class in Moose is executed by the term *package* [13]. The term *has* is used to declare attributes [13]. The attribute ro is short for read only.

The second object class program is written using Ruby programming language. Ruby uses similar methods as Perl's object class; mutators and accessors. Figure 5-2 illustrates a Ruby script that demonstrates how the mutator and accessors are used. In Ruby programming language, mutators and accessors are programmed using the code attr_accessor [5]. This code can be used if the attribute is read and written. Figure 4-3 illustrates a program using mutators and accessors. Also shown in figure 4-2 is Ruby programming style. Creating a class in Ruby is executed by the term *class* [5].

The parser function separates data in a manner given from the programmer [2]. For example, a programmer can separate data by comma, white space, or periods. The parsing function in the SpecEd application separates the CSV file by commas. Figure 5-6 is an example of modeled CSV data file. Perl's parsing script is shown in figure 5-4.

Ruby programming language uses the split operator which functions similar to the parser function used in Perl [5]. Ruby's split function is shown in Figure 5-5. The result of the parsing function and splitting function is shown in Figure 5-7.

```
package SpecEd;
use Moose;
use strict;

#Function that defines the property for attribute
has ‘given_student_id’      => (is => 'ro');
has ‘last_name’              => (is => 'ro');

#Function that creates an accessor for the attribute given
# given_student_id accessor
    sub get_given_student_id {
       my $self = shift;
        return $self->{given_student_id};


#Function that creates a mutator for the attribute given
# given_student_id mutator
     sub set_given_student_id {
        my $self = shift;
        return $self->{given_student_id};
```

Figure 5-1 Excerpt of Moose Script

```
#initalize a class
class SpecEd

#accessor method
attr_accessor
:given_student_id,
:last_name,

end
```

Figure 5-2 Excerpt of Ruby's Script

```
class SpecEd

#setter method
def set_given_student_id(s)
    @given_student_id = s

#getter method
def get_given_student_id
    @given_student_id

end
```

Figure 5-3 Ruby's Accessor method

```
# split the entry using the ',' deliminator

@parsed = split(/,/, $line);
```

Figure 5-4 Perl's Parser script

```
# split the entry using the ',' deliminator

file.each_line("\n") do |values|

   parse = values.split(",")
```

Figure 5-5 Ruby's Parser script

```
3681372179,BOUCH,Daniel,,09011993,02,05,02,12,2127,,,128327303,Be
acon Light BHS,9999,,,04,04,06,,01;03;07;12,Teacher Outside of
IU,01;02;03,,02,,,,,,,

6852109658,JUNOD,Eric,,10311995,02,05,02,09,2127,,,128327303,Beac
on Light BHS,9999,,,04,04,16,,01;05;06;12;18,Teacher Outside of
IU,01;02,,02,,,,,,,

2648772162,SHILLING,Connor,,05282001,02,05,02,04,2124,2129,,12803
0852,Clelian Heights,9999,,,04,02,16,,01;05;11;18,Teacher Outside
of IU,,,03,,,,,,,
```

Figure 5 -6 Unparsed modeled CSV file

given_student_id	3681372179
last_name	BOUCH
given_student_id	Daniel
last_name	
first_name	09011993
middle_initial	02
birth_date	05
gender	02
ethnic_background	12
limited_english_proficient	2127
grade	
disability	128327303
secondary_disability	Beacon Light BHS
residency	9999
home_district	
building	
location_code	04
area_office	04
service_provider	06
amount_spec_ed	
support_type	01;03;07;12
ed_env	Teacher Outside of IU
ed_env_percentage	01;02;03
related_services	
teachers_lname	02
transition	
service_plan_non_public	
neighborhood_school	
attends_reg_ch_prog	
calc_time_ed_reg_ch_prog	
time_ed_reg_ch_prog	6852109658
ps_environment	
ps_funding_stat	
exit_date	
exit_reason	

Figure 5-7 Parsed modeled data file

Chapter 6

Database Interface

This chapter explains the database implementation which was first designed by Tim Bunce using Perl programming language [2]. A brief overview of database operation implementations, create, retrieve, update, and delete are given. Also introduced in this chapter is Standard Query Language which is database language used to perform database manipulations [10].

The Database Independent Interface (DBI) is an interface between the SpecEd class object program written in Perl and Ruby and the database [1]. The architecture of the DBI is shown in Figure 3-1 [1]. A database is a collection of organized data or information that is related in some way. There are three types of database models; hierarchical model, network model, and relational model [10]. The hierarchy and network are tree-like model databases and the relational model is organized as a table consisting of rows and columns [10].

The model concept that is used in this comparative study is the relational database model. Relational is information within that database is similar or data is grouped together based on equivalent traits [10]. The parsed data file in Figure 1-7 is resembles a relational database table that can be uploaded to a standard database.

A relational database table consists of rows and columns. The columns are the attributes and the rows contain data values for each attribute [10].

The Application Programming interface (API) is the interface between programming languages and the database [1]. The API has a predefined list of connections that allow programming languages to interact with database servers regardless of the database type [1]. Each database server has a driver [1]. A driver is used to access an application [1]. For example, before a webcam can be used on a computer, the user must download the webcam's driver so that the computer will recognize the webcam and ensure that it performs properly.

The DBI architecture drivers allow connections to databases such as MySql, Oracle [1]. There are three types of database handles [1]. The first handle is the driver handle. The driver handles are responsible for taking the connection request to the desired database server [1]. The second is the database handle. The database handles permits connection to the requested database [1]. The third is statement handles. The statement handles allows interaction and manipulation within a database [1]. The Standard Query Language (SQL) is standard database language used to interact within the database [10]. Figure 6-1 illustrates the Database Implementation Interface [1].

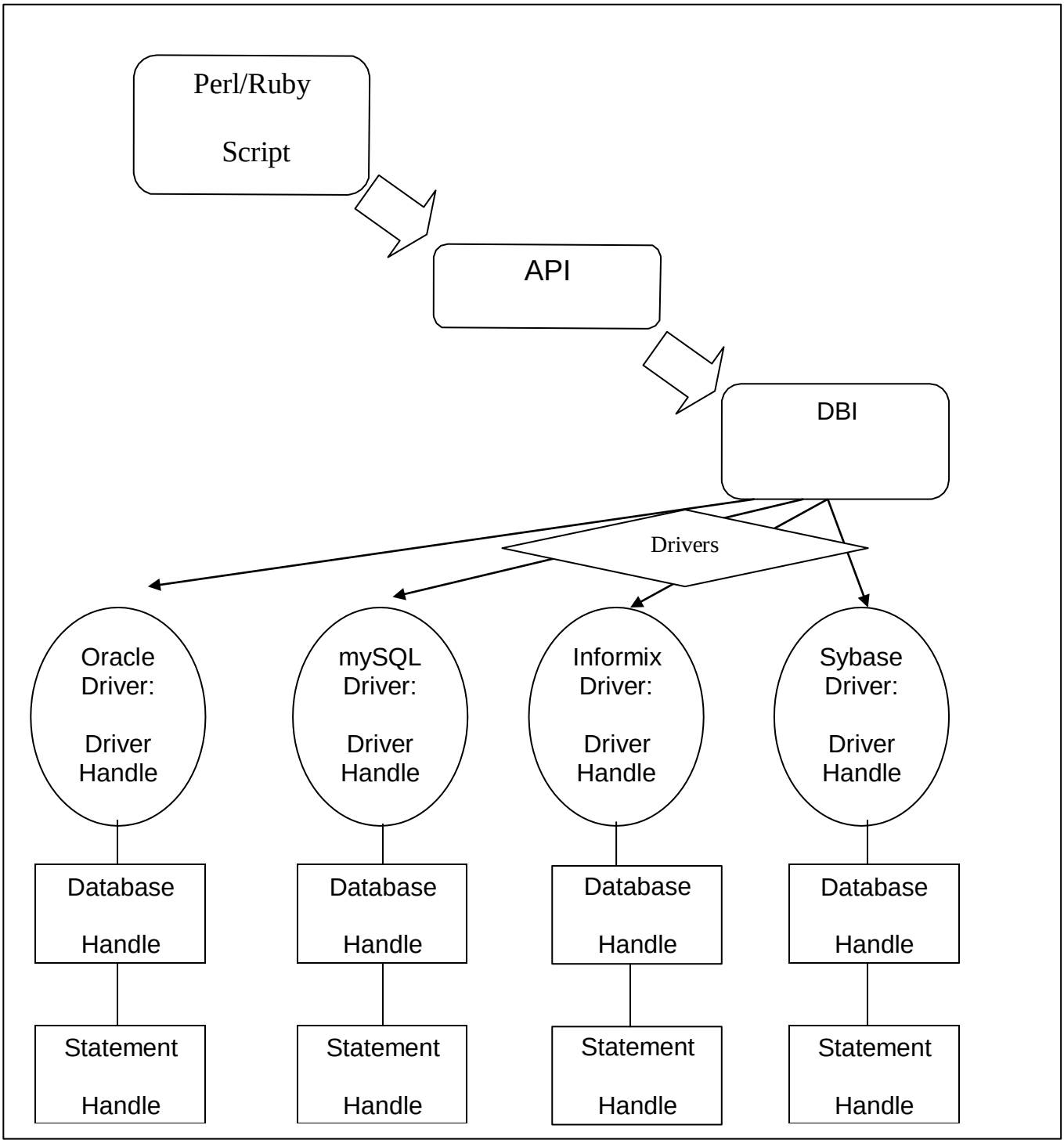

Figure 6-1 Database Interface Architecture

The Perl code to connect and disconnect to a standard database is shown in Figure 6-2 [1]. Ruby follows a similar pattern in Figure 6-2 to connect to a standard database.

```
#Connect to the Oracle Database
eval {

 $ora_dbh = DBI->connect( "dbi:Oracle:ED11GR2" , "sy2010_2011",
                  "sy2010_2011", {

  AutoCommit=>0,
  RaiseError=>1,
  PrintError=>0,
 },)

 or die "Failed to connect to the database\n";

#Disconnect to the Oracle Database

eval { $ora_dbh -> disconnect };
```

Figure 6-2 Perl's database connection and disconnection script

The *AutoCommit* method defines the how a standard database handles transactions [10]. For example, if table is deleted accidently within a database, disabling *AutoCommit* allows the database to roll back to the table being present in the database [10]. The zero is disabling a method and the one is enabling a method. The *RaiseError* method terminates a program if the DBI detects an error [10]. The *RaiseError* prints an error message if there is an error detected.

Once a connection to the database is successful, then the SpecEd object class program can be uploaded into the database. Connecting to the database can be directly using the command line prompts [9]. For example, launching the MySQL database is done by typing *mysql* on the command line prompt. SQL statements to enter data are done individually. For example, using the data from the SpecEd application is done by entering each student's record one at a time. Figure 6-3 is an example of SQL statements. The statements are examples of creating, retrieving, updating, and deleting (CRUD) database operations [1].

Creating a Table: Create Table SpecEd_data (given_student_id INTEGER,
last_name VARCHAR(16),
first_name VARCHAR(16)
)

Inserting Data: INSERT INTO SpecEd_data (given_studen_id, last_name)
Values (3681372179, 'Bouch')

Retrieving Data: SELECT given_student_id, last_name FROM SpecEd_data

Updating Data: UPDATE SpecEd_data given_student_id = 6852109658
Where last_name = 'Junod'

Deleting Data: DELETE FROM SpecEd_data

DROP TABLE SpecEd_data

Figure 6-3 SQL statements for C.R.U.D. operations

Chapter 7

Web Application Frameworks

Catalyst and Ruby on Rails is discussed in this chapter. Catalyst and Ruby on Rails are open source web frameworks. Catalyst and Ruby on Rails patterned the Model View Controller architecture to develop web applications [9 11]. Catalyst is the MVC framework for Perl. Ruby on Rails is the MVC framework for Ruby.

Trygve Reenskaug was first to portray the Model View Controller [16]. The controller performs as a dispatcher [9 11]. The controller communicates with the model and view [9 11]. The controller handles the user requests on a web application. For example, if the user requests an action, the controller's responsibility is to pass the request to the view and model.

The view generates content to the user in HTML format [9 11]. The view receives communication from the controller and communicates with the model [9 11]. For example, if the request from the controller is to delete data in the database, the view will communicate to the model this action. The model will reflect the deletion and the view will display that the delete action to the user.

The model is the source for data [9 11]. The model is the interface between the database and the model and controller [9 11]. The model stores database information which describes how the database is organized and the contents within the database [9 11]. For example, if the database is password protected, the password or other sensitive information is stored in the model.

7.1 Catalyst Application

Generating a Catalyst application is performed by first executing the command [9],

catalyst.pl nameofapplication

The next step is connecting to the server. The web server displays the application during the developing stages. Connecting to the web server is also a method to ensure that Catalyst application is running successfully [9]. For example, if there is an error within the application, Catalyst will not connect to the web server. Catalyst has built in helper scripts that generate the controller, model, and view automatically [9].

The controller helper script generates actions which are displayed as URLs [9]. For example, using the helper script to generate the controller [9] SpecEd is

perl script/ nameofapplication_create.pl controller SpecEd.

The URL displays http://url/SpecEd. The view is used to display the SpecEd content [9].

The view helper script generates HTML content using Template Toolkit and FormBuilder. Template Toolkit and FormBuilder are extension of Catalyst modules [9]. Template Toolkit (TT) generates templates with headers and footers so that the user can continue to write without using HTML [9].

FormBuilder generates form - templates that validate data submitted by the user [9]. For example, if the field entry for a form takes a minimum of 10 characters, then the user has to submit 10 or more characters for successful submission. Figure 7-2 illustrates an example of a form generated using the view helper scripts.

Forms generated in the Catalyst application must have templates [9]. Also all the actions in the Catalyst application must have templates [9]. The templates are needed so that the view can properly generate content upon request [9]. For example if there is an *edit* action, then there must be an *edit* template to generate the content if requested by the user If there is an *edit* form, then there must be an *edit* template to generate the content if requested by the user. Figure 5-3 gives an example of the data flow for *edit* to generate properly.

The model helper scripts are used to append existing standard databases to the Catalyst application [9]. For example, if there exists a database and desired Catalyst application wants to use it, then the model script helper will add the database to the application. The model script is also used to deploy any database [9].

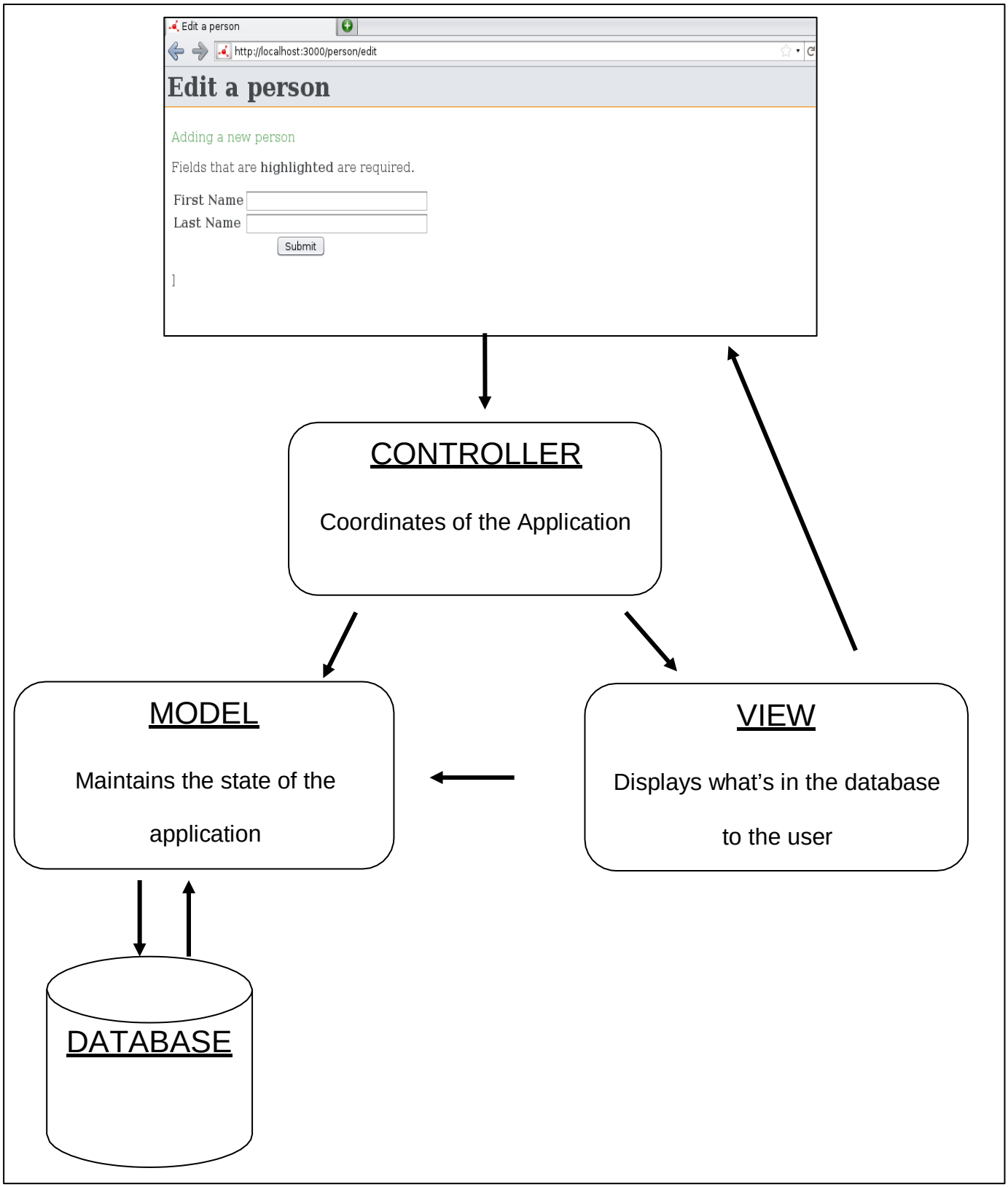

Figure 7-1 MVC Architecture [16]

```
name: student_edit
method: post
fields:
    firstname:
     label: First Name
      type: text
      size: 30
    required: 1
    lastname:
      label: Last Name
      type: text
      size: 30
      required: 1
    middlename:
      label: Middile Initial
      type: text
      size: 1
    required: 1
    schoolname
      label: School Name
      type: select
      required: 1
```

Figure 7-2 E*dit* form created FormBuilder [9]

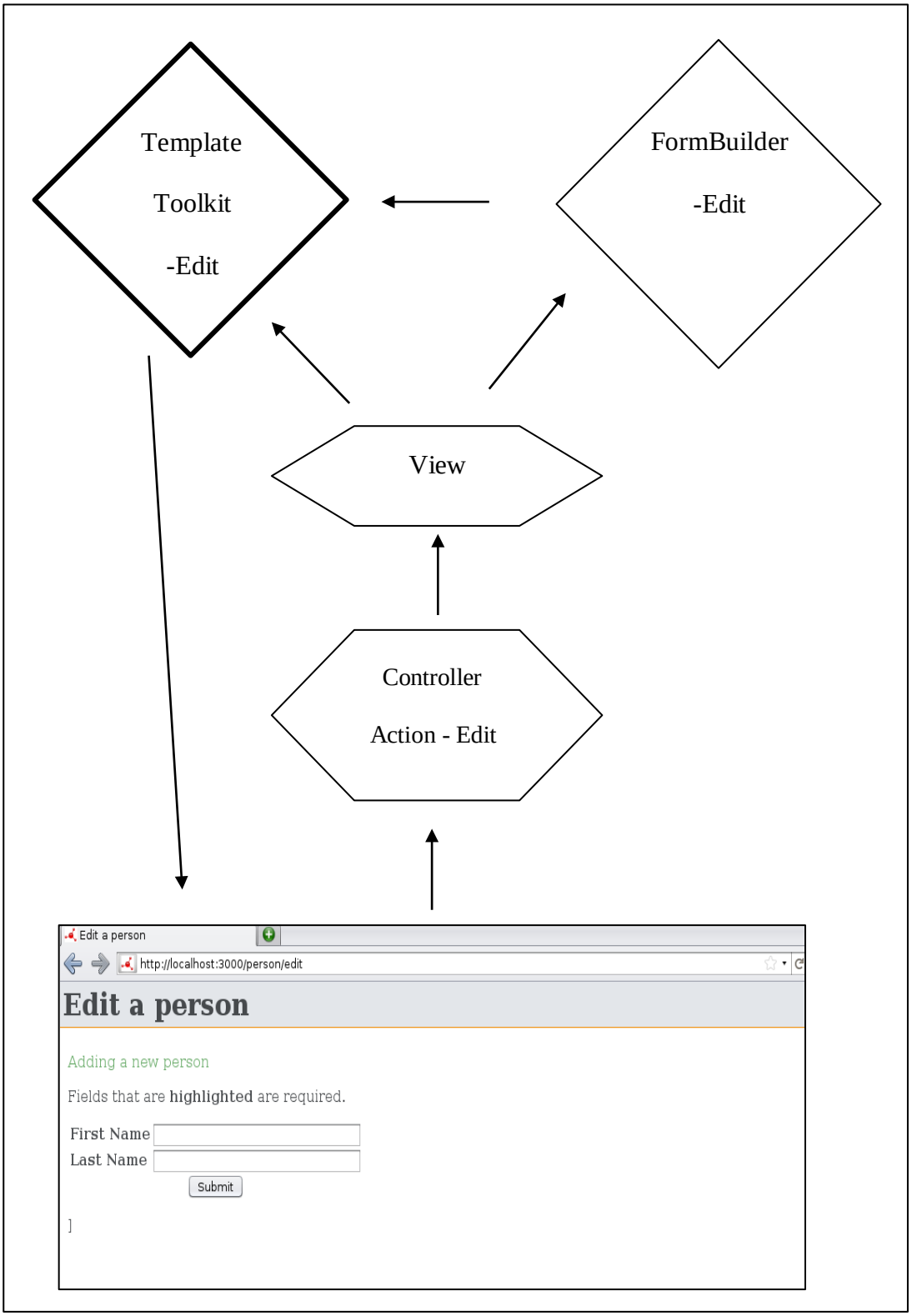

Figure 7-3 *Edit* action in Catalyst [9]

7.2 Ruby on Rails Application

Generating a Generating a Ruby on Rails application [11] is performed by first executing the command,

rails nameofapplication

The next step is connecting to the server,

r*ails server*

A connection to the web server ensures that Ruby on Rails is working [11]. Generating a controller and view can be executed with a single helper script [11],

rails generate controller Spec_Ed home

The helper script generates Spec_Ed as the controller and home as the view. Executing the controller and view helper scripts in this manner generates HTML files for the view director and the URLs for the controller directory [11].

Ruby on Rails uses the scaffolding helper script. Scaffolding is used to generate CRUD operations for Ruby on Rails applications automatically [11]. Scaffolding generates HTML code for creating, retrieving, updating and deleting [11]. Figure 7-3 illustrates the results of a Ruby on Rails application after executing the command

rails generate scaffold Student student_name:string student_School:string student_ID:binary

The view creates HTML for *Show*, *Edit*, and *Destroy* links automatically when using the scaffolding helper [11]. The view templates that are generated from the scaffolding helper script can be modified and customized.

The helper script to create and manipulate database tables in the Ruby on Rails application is [11]

rake db:create

This helper script generates a table in the application's directory and can be altered after creation. For example, if a row needs to be added or deleted, then changes can be made on the specified database file in the directory. Appending existing databases can be done by using the same database file and changing the existing parameters. Ruby on Rails supports MySQL, SQLite3, and PostgreSQL databases and has built in configurations the databases.

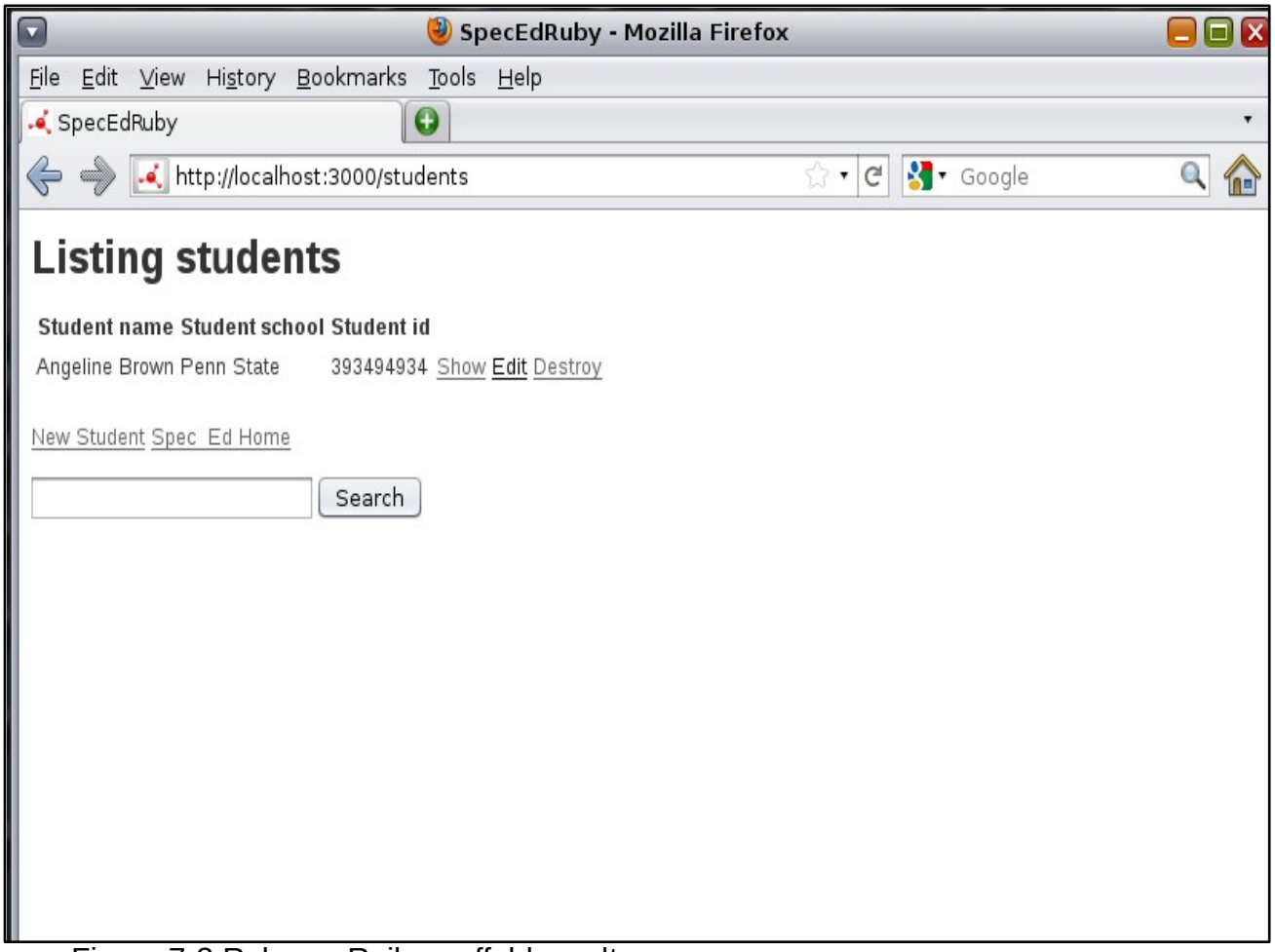

Figure 7-3 Ruby on Rails scaffold results

Chapter 8

Conclusion

Using the experiments and analysis provided in this research, one can decide on the best programming language to apply to a software application. Ruby and Perl have similar features and programming characteristics. A characteristic that distinguishes Ruby from Perl is that Ruby's object oriented model is designed from the ground up. Perl's Moose, an extension of Perl's programming language, gives Perl more OO flexibility.

Based on the experiments performed in this research, Perl and Ruby are close in performance. Ruby experienced a longer execution time compared to Perl on the quick sort and parsing algorithms. Using the benchmarks in this research, an explanation of the performance behavior was analyzed using the DTrace framework analyzing tool. Ruby and Perl memory usage is approximately the same, therefore time performance can be considered an idea factor when considering which language to apply to a software application.

The DTrace framework help locate where latency occurred in Ruby's quick sort algorithm and parsing algorithm. In Ruby's quick sort algorithm, latency occurred in the partitioning method. In Ruby's parsing algorithm, DTrace identified that the splitting method and reading each line from the file shared longer execution times compared to Perl. These analyses help explain the reason Ruby's execution time is longer than Perl's execution time in the quick sort and parsing algorithms.

Another reason Perl and Ruby have different execution times is due to the garbage collector. DTrace scripts on the experiments in this research were used to monitor the garbage collector for Perl and Ruby programming languages. Ruby's garbage collection algorithm suspends the program to execute. Perl's garbage collection algorithm executes when the program is terminated to reclaim objects.

The Catalyst and Ruby on Rail web frameworks share the model-view-controller concept. Based on the case study used in this research, Ruby on Rails is suitable for creating a CRUD (create, retrieve, update, delete) database framework. Ruby on Rails uses a scaffolding helper script that generates a webpage template with a CRUD database interface. Catalyst can also be used for the same purpose, but if the design time is a critical factor Ruby on Rails is an ideal application.

References

[1] Bunce, Tim. *Programming the Perl DBI*. O'Reilly Media, Inc., 2000.

[2] Schwartz, Randal. *Learning Perl, 5th Edition*. O'Reilly Media, 2008.

[3] Gamma, Erich. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[4] Tucker, Allen B. *Programming Languages: Principles and Paradigms*. McGraw-Hill Higher Education, 2007.

[5] Olsen, Russ. *Design Patterns in Ruby*. Addison-Wesley Professional, 2007.

[6] Cormen, Thomas H. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[7] Flanagan, David. *The Ruby Programming Language*. O'Reilly Media, 2008.

[8] Hennessy, John L. *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann, 2006.

[9] John, Antano S. *Catalyst 5.8: the Perl MVC Framework*. Packt Publishing, 2010.

[10] Ramakrishnan, Raghu. *Database Management Systems*. McGraw- Hill Science/Engineering/Math, 2002.

[11] Ruby, Sam. *Agile Web Development with Rails, Third Edition*. Pragmatic Bookshelf, 2009.

[12] "perlobj - perldoc.perl.org." Perl programming documentation - perldoc.perl.org. Jon Allen, n.d. http://perldoc.perl.org/perlobj.html

[13] "The Moose is Flying (part 1) (Jun 07)." Stonehenge Consulting Services, Inc.. Randal L. Schwartz, n.d. <http://www.stonehenge.com/merlyn/LinuxMag/col94.html>.

[14] Cabal, Moose . "Moose - search.cpan.org." The CPAN Search Site - search.cpan.org. N.p., n.d. <http://search.cpan.org/~doy/Moose-2.0401/lib/Moose.pm>.

[15] Press, Bruno R. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*. Wiley, 1998.

[16] "Trygve's homepage.." *heim.ifi.uio.no*. Trygve M. H. Reenskaug, n.d. Web. <http://heim.ifi.uio.no/~trygver/>.

[17] "The Perl Community - www.perl.org." *The Perl Programming Language - www.perl.org*. N.p., n.d. <http://www.perl.org/community.html>.

[18] "Ruby's Style Guide." *A community-driven Ruby coding style guide.* GitHub Inc, n.d.. https://github.com/bbatsov/ruby-style-guide.

[19] "Perlstyle - perldoc.perl.org." *Perl programming documentation - perldoc.perl.org*. Jon Allen (JJ), n.d. <http://perldoc.perl.org/perlstyle.html>.

[20] Gregg, Brendan, and Jim Mauro. *DTrace: Dynamic tracing in Oracle Solaris, Mac OS X, and FreeBSD*. 1. print. ed. Upper Saddle River, NJ: Prentice Hall, 2011. Print.