A Look at Dynamic Languages

Luis Rei
Faculty of Engineering University of Porto
luis.rei@gmail.com

Sara Carvalho
Faculty of Engineering University of Porto
ei05072@fe.up.pt

Marco Alves
Faculty of Engineering University of Porto
ei05099@fe.up.pt

João Brito
Faculty of Engineering University of Porto
ei07052@fe.up.pt

November 30, 2007

Abstract

This paper aims to present an overview of Dynamic Languages in comparison with the more traditional languages, namely Java and C++. The definition of the term "dynamic language" is given and what is commonly understood nowadays when the term is used. Then one lists the most common features of these languages and the advantages and disadvantages pointed by their proponents and opponents. Furthermore is enumerated some of the domains in which dynamic languages have been more successful and explain the reasons that justify it, as well as the domains in which there are few or no reports of success. Finally, some common examples of fanaticism sorrounding dynamic languages are given.

1. Introduction

Hundreds of programming languages exist and more are being developed every year. Some believe that all programming languages are pretty much equivalent and thus advocate the use of a standard software development language, currently Java but in the past it was C++ and before that Cobol. But if languages are all equivalent why should the developers of Java have even bothered to create a new language? Presumably, if one creates a new language, it is because one thinks it is better in some way compared to what people already have. Java was designed to fix some problems with C++[1]. Ergo, languages are not all equivalent. Some languages are better, for certain problems, than others. Java was designed to be better, for certain problems, than C++. What problems? When is Java better and when is C++? More importantly are there situations where other languages are better these? [2]

Dynamic languages were created to solve particular problems[4], to be better than existing languages for certain situations. Some of the features required by those particular problems and situations unintentionally made them better in other circumstances, as well.

The scope of this article will be limited to the most successful languages in general software development as well as in certain particular fields pointing out the reasons, technical or otherwise, that made that success possible.

2. What is a Dynamic Language

Like many other terms in the IT business, there is no universally accepted definition of what constitutes a dynamic language. A widely accepted abstract definition [4] [5] states that any programming language that allows one to write applications that can change the structure of the program while it is running, often referred to as run-time code modification, is a dynamic programming language. The original definition [3] also included the need of the language to be high level, dynamically typed and open source. As these languages

were previously commonly known as scripting languages and that they are generally interpreted, some definitions also state that dynamic languages are interpreted [6]. One particular expert has a working definition based on whether a language can be easily used based on an example he gave [7].

The fact remains, however, that the most easily identified dynamic languages are still the ones that were originally defined as such. Typically, when the term is used it refers to high level, dynamically typed, open source, interpreted languages such as Python, Perl and Ruby. When discussing general software development, the focus will be on Python and Ruby as they are the most popular general purpose dynamic languages in use.

What all the criteria have in common is that they themselves have no universally accepted definitions. In the fast-paced world of technology, definitions can quickly become outdated. Thus the definitions of the terms generally used to define the concept of Dynamic Language is given here.

2.1. Run-time Code Modification

Dynamic languages allow programs to change their own structure while running (run-time). The run time modifications include loading arbitrary code at run-time [3] through the use of statement's such as Python's import [8] – which allows a module containing new code to be added – the module name may even be a string supplied by a user interacting with the program.

A common use of this ability is extending a class or an object by adding new methods to it.

2.2. High-Level

In this context, level is a relative measure of abstraction. C is considered a high-level language relative to assembly because it has a higher level of abstraction. It managed to abstract away the hardware.

Python and other dynamic languages are another level of abstraction up. The features expected from a high-level language today are the automation of routine tasks like memory management and exception handling, more abstract built-in data types like lists and dictionaries, non-static typing mechanisms and specific syntax choices to promote better code readability [3] and significantly reduced code verbosity [9].

Dynamic languages are designed under the assumption that there already exists a collection of useful components to create applications and thus are intended for plugging together those components further increasing the level of abstraction from system languages which were designed to create those components [9]. A good example of this ability is Python and Ruby's use Java classes [10] [11] or .NET libraries [12] [13].

2.3. Interpreted

Dynamic languages are often interpreted as opposed to compiled. This means that source code is read at run-time by an interpreter - a computer program that translates source code into a target representation that it immediately executes and evaluates [14]. This process is opposed to compilation in which a compiler reads the source code and transforms it into machine code, creating an executable to be run at a later time. C is an example of a language that is usually compiled.

Proponents of interpretation advocate that it results in faster development by eliminating compile times but also accept that it is less efficient [9].

Nowadays the line between interpretation and compilation is blurred. Many languages compile into intermediate forms such as byte-code which is executed by a byte-code interpreter. This is done for both dynamic languages such as Python [15] and more traditional system languages such as Java [16]. There are also techniques such as Just-In-Time compilation meant to increase performance of byte-code interpreted languages which make the distinction between interpreted and compiled languages less significant [17].

Some interpreted dynamic languages have not only a reference implementation, consisting of an official interpreter and a set of libraries, but also other implementations. Python's reference implementation is CPython, written in C [10], but there is also Jython, written in Java and targeting the JVM capable of using any Java class [12], and Iron Python, written in C# and targeting .NET capable of using any .NET library available [18]. These two implementations are meant to increase the power of Python by giving it access to other large library collections. Interpreted languages often have other implementations for increased performance, sometimes in specific areas, much in the same way as compiled languages have different compilers for increased performance, sometimes for specific processors like ICC for C++ [19]. Ruby for instance has alternative implementations YARV [20] and Rubnius [21] that seek to provide a high-performance ruby environment by compiling ruby source code to byte-code (at the time of writing, the reference ruby interpreter [22] currently does not use byte-code).

In the case of Python, when performance is critical, new extension modules written and compiled in C [23].

2.4. Typing

2.4.1. Dynamic Typing

Dynamic typing was pioneered by Lisp [9]. A language that uses dynamic typing is said to be a dynamically checked language [24] since type checks occur at run-time (during execution) as opposed to compile-time as happens with static typing. Type checking consists of verifying that code respects type constraints preventing the application of operations to objects of incompatible types. If a language does not require that the type of a variable be known at compile time, then a language is said to be dynamically typed [25].

Advocates of dynamically typed languages argue that they are more suited for prototyping systems with changing or unknown requirements [26] such as systems that are not precisely specified (the problems addressed aren't yet well understood) and systems that are are evolving fast (due to changing standards or changes of opinion) [3].

In a dynamically checked, strongly typed language such as Python, the pseudo-code

```
1+"1"
```

which would add a number to a string would generate a run-time error while in a statically typed language such as C, the compiler would display an error message and refuse to generate an executable.

For example, the Python run-time when evaluating the expression a+b, it must first inspect the objects a and b to find out their type, which is not known at compile time. It then invokes the appropriate addition operation, which may be an overloaded user-defined method.

2.4.2. Weakly Typed

Some dynamic languages are weakly typed, which means they allow implicit conversions (or casts) of types [9]. The previously mentioned example of adding a number to a string would be a valid operation. In JavaScript it would result in the string "11" while in Perl (version 5) it would result in the number "2". The dynamic languages most commonly referred as weakly typed are JavaScript, Perl and PHP.

2.4.3 Strongly Typed

System languages introduced strong typing. In a strongly typed language a programmer most declare how each piece of information will be used and the language prevents it from being used in any other way [9]. The dynamic languages most commonly referred as strongly typed are Python and Ruby.

2.4.4 Type Inference

Type inference allows programmers to omit type information when declaring a variable [26]. Instead of writing

```
Integer var;
var = 3
```

the programmer can just write

```
var = 3
```

and the compiler or interpreter will know that "var" refers to an integer value.

2.5. Grassroots Open Source

The official definition of Open Source [27] states that Open Source essentially means free redistribution of both the software and the source code, possibly modified to create a derivative work. However, the term is also used to refer to a development model in which code is developed over the Internet, in view of the public, by volunteer developers forming a "community of peers" open to new members often spontaneously created following an initial release by the original author [3] [28] [29]. Such was the case with Python [30] and Perl [31].

2.6. Philosophy Behind The Development of Dynamic Languages

The most popular dynamic languages were designed to solve technical problems faced by their inventors and tend to remain focused on solving technical problems as opposed to being tools to further a corporate agenda. These languages are built on a philosophy of optimizing person-time instead of computer-time - they sacrifice efficiency to increase productivity. The very development of these languages differs significantly from the traditional model of programming language development, it's so deeply open source that there is an almost total transparency about how the language evolved through the bug tracking records, the version control software's logs and the discussions in the mailing lists [3] [32] [33]. All of which being public. Another peculiarity of the development process is that the language's core evolves separately from the libraries. The core is controlled by a small team of individuals who ensure that the language

does not derivate from it's design principles. Anyone can develop libraries, modules and extensions and make them available to the general public [3].

2.7. Dynamic Languages *versus* System Languages

System languages, the most popular of which being C\C++, Java and C#, are characterized by strong typing and the ability to build tightly-coupled efficient systems from scratch [3]. They replaced assembly languages providing higher abstraction from the underlying hardware, allowing applications to be developed more quickly at the cost of some efficiency [9]. Dynamic languages make the same trade-off, they allow applications to be developed more quickly at the cost of some efficiency [3] in part due to being interpretted and in part because their basic components are chosen for power and ease of use rather than an efficiency [9]. Dynamic languages tend to be more expressive or less verbose, requiring less words to accomplish the same task [9].

```
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello, world!");
    }
}
```

Example 10a: Java code - “Hello World”

```
print "Hello, world!"
```

Example 10b: Python code - “Hello World”

Java 6 -server *x times* better ~ Python *x times* better		
Program & Logs	Faster	Smaller: Memory Use
binary-trees	10.5	~1.5
fannkuch	**71**	~4.1
fasta	**33**	~4.3
k-nucleotide	3.8	~2.6
mandelbrot	**107**	~4.2
meteor-contest	**23**	~4.9
n-body	**131**	~4.6
nsieve	3.7	2.2
nsieve-bits	**22**	1.0
partial-sums	2.2	~4.1
pidigits	1.0	~3.5
recursive	**97**	1.3
regex-dna	~1.1	~2.2
reverse-complement	1.0	~1.2
spectral-norm	**135**	~3.9

Image 1: Java vs Python (performance and memory use)[91].

x	language
1.0	C++ **g++**
1.1	C **gcc**
1.3	Pascal **Free Pascal**
1.6	Lisp **SBCL**
1.7	Java **6 -server**
1.8	BASIC **FreeBASIC**
1.8	Ada 2005 **GNAT**
2.7	C# **Mono**
9.0	Smalltalk **VisualWorks**
11	**Lua**
17	**Python**
20	**Perl**
22	**PHP**
43	JavaScript **SpiderMonkey**
55	**Ruby**
62	Prolog **SWI**

Image 2: performance comparison between several languages. (x times slower in comparison with the first reference)[91].

2.8. Prototyping

One of the often advocated uses of dynamic languages is prototyping [3]. "Plan to throw one away;

you will anyway" - the first attempt at a software design often turns out to be wrong unless the problem is very simple [34]. New requirements and features can become apparent once development has started.It might not be possible to cleanly incorporate it into the program's structure. Dynamic languages facilitate quickly developing an initial prototype that lets you get the overall program structure and logic right. Once satisfied with the interface or program output, the code can be translated into a System language like Java [35].

3. Dynamic Languages in Science and Education

3.1. Introduction To Programming

Both Python and Ruby are often mentioned as good first programming languages. The arguments in favor of using them to teach programming are usually the fact that they are high-level [36] and their syntax is clean, easy to read and is very similar to the language neutral pseudo-code used in many textbooks [36] [37]. The greatest criticism leveled against them is their non-static typing. Wether that is truly a disadvantage or an advantage is debatable [38].

The proximity between dynamic languages and the language neutral pseudo-code can be seen in the following examples (taken from [37] and [43]):

```
Insertion-Sort(A)
 for j <- 2 to length[A]
      do key <- A[j]
            i <- j - 1
            while i > 0 and A[i] > key
                  do A[i+1] <- A[i]
                        i <- i - 1
             A[i + 1] <- key
```

Example 2a - Algorithm from textbook

```
def InsertionSort(A):
    for j in range(1, len(A)):
        key = A[j]
        i = j - 1
        while (i >=0) and (A[i] > key):
            A[i+1] = A[i]
            i = i - 1
        A[i+1] = key
```

Example 2b - Python code

```
procedure bubbleSort( A : list of sortable items
) defined as:
  for each i in 1 to length(A) do:
       for each j in length(A) downto i + 1 do:
         if A[ j ] < A[ j - 1 ] then
           swap( A[ j ],  A[ j - 1 ] )
         end if
       end for
  end for
end procedure
```

Example 3a - wikipedia description of the bubble sort algorithm

```
def bubbleSort(a)
  0.upto(a.length-1) do |i|
     (a.length-1).downto(i+1) do |j|
         if a[j] < a[j-1]
             a[j],a[j-1] = a[j-1],a[j]
          end
       end
  end
   return a
end
```

Example 3b - Ruby Code

Many higher education institutions such as the MIT and the University of Cambridge use functional languages in introduction to programming courses for computer science students[39][40], which is sometimes justified by claiming a better mathematical basis [41] or the need to teach concepts of functional programming [42]. However, Cambridge recommends their future students to first learn Python in order to explore algorithms without getting bogged down in details and warning students away from C and C++ claiming self-teaching these languages might lead students to picking up bad habits [43]. Python wins in terms of simplicity when compared to functional languages, such as Scheme and ML: functional concepts can still be presented in a less rigorous way and more detached from the the mathematical foundations of computing. Python is actually closer to pseudocode presented in language-neutral textbooks used in many computer science courses [36]. The creator of C++ himself argues that learning C before C++ leads to an early focus on low-level details and obscures programming style and design issues by forcing the student to face many technical difficulties to express anything interesting and argues that it is a better approach to first present higher level code before going into the lower-level details [44]. Clearly, the use of a dynamic language to to teach basic programming is consistent with the spirit of those recommendations.

Dynamic languages allow students to concentrate on important programming skills such as problem decomposition and data type design while with System languages the students are trying to learn to think like a computer, decompose problems, design consistent interfaces, and encapsulate data [45].

It should also be important to retain students interest in programming by letting them practice with real problems without going into many low-level implementation's details that might require more advanced knowledge or time. This is made possible by

the large standard libraries of dynamic languages [45]. A student's enjoyment and interest in programming increases if they can quickly write a functional and useful program [46] instead of a simple "hello world" or calculator. The following is an example of code in C, modified (comments and error handling removed) from an exercise of the third year course of Computer Networks in Electrical Engineering at FEUP:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <netdb.h>
#include <strings.h>
#include <string.h>


#define SERVER_PORT 80
#define MIN_BUFF 1000
int ligarServidor(char * url){
    int    sockfd;
    char    host;
    struct    sockaddr_in server_addr;
    struct     hostent * servidor;
    servidor=gethostbyname(url);

bzero((char*)&server_addr,sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr =
inet_addr(inet_ntoa(*((struct in_addr
*)servidor->h_addr)));
    server_addr.sin_port = htons(SERVER_PORT);
    sockfd = socket(AF_INET,SOCK_STREAM,0);
    connect(sockfd, (struct sockaddr
*)&server_addr, sizeof(server_addr));
    return sockfd;
}
void main(){
    int    sockfd;
    char    buf[] = "GET /nwshp?hl=pt-
PT&tab=wn&q=&output=atom HTTP/1.1\nHost:
news.google.pt\n\n";
    char    * bufferPtr;
    int    bytesSent;
    int    rec=0;
    int stop=0;
    int    nBytesRec=0;
    int    i=0;
    char *url="news.google.com";
    int    bufferSize=1;
    bufferPtr = (char *) malloc(MIN_BUFF);
    sockfd = ligarServidor(url);
    bytesSent = write(sockfd, buf, strlen(buf));
    do {
        rec = read(sockfd,
(bufferPtr+nBytesRec), 1);
        if(nBytesRec == (bufferSize*MIN_BUFF)) {
            bufferSize++;
            bufferPtr = (char *)
realloc(bufferPtr, MIN_BUFF*bufferSize);
        }
        if(rec == 1)
            nBytesRec+=1;
        else
            stop=1;
    } while(!stop);
      printf("%s",bufferPtr);
    close(sockfd);
     free(bufferPtr);
}
```

Example 1a: C code, get the an atom feed from news.gogle.com, written by a student

```
import feedparser
feed_url = "http://news.google.com/?output=atom"
feed = feedparser.parse(feed_url)
print feed
```

Exmple 1b: Python code - with the added advantage of the retrieved data already being in an appropriate structure (already parsed)

Obviously, the complex C code in example 1 is beyond the scope of an introductory programming course as it requires knowledge of sockets, TCP/IP and of the HTTP protocol while the code in Python does not. There is also a clear difference in readability. To experienced programmers, with no knowledge of the language, simple programs are usually immediately comprehensible [46].

Furthermore the interpreted nature of dynamic languages encourages experimentation while their self documentation features, such as Python's dir() [47] function, facilitate further learning.

Python is well designed for beginners being easy to use and easy to learn [48], it was, in fact, designed to be easier to learn, read and use yet powerful enough to illustrate essential aspects of programming languages and software engineering [49]. These advantages not only make Python well suited for teaching programming to Computer Science students but also for students in other areas such as Biology [50].

There are also many projects that aim to teach dynamic languages to broader audiences, including children [49] [51].

3.2. Artificial Intelligence

Lisp, which some consider a dynamic language, has been an almost de-facto standard in the Artificial Intelligence field. Prolog, another language that can also be considered a dynamic language, is another favorite in the field.

Python is a dynamic language that has been gaining popularity in Artificial Intelligence, mainly thanks to it's proximity to Lisp. Peter Norvig, co-author of the leading textbook in Artificial Intelligence [52], concludes that Python is a better language than Lisp (or

Java) for teaching AI, on the basis that it not only has almost all the features of Lisp but it is easier to read for someone with no experience in either of the languages, the implementation is closer to the pseudo-code used in the book, it is easier to use and it has standard GUI and Web libraries [53]. Python is now used in Artificial Intelligence courses in many Universities, including The University of Pennsylvania [54].

3.3. Data Analysis and Statistics

The R programming language is the de-facto standard in data analysis and statistics [55]. R implements a dialect of the S language that was developed at AT&T Bell Laboratories by Rick Becker, John Chambers and Allan Wilks. Versions of R are available for Windows, many Unix variants (Linux, BSD, ...) and Mac OS X. It is available through the Comprehensive R Archive Network (CRAN) which has over 60 mirrors word-wide and distributes both R and user submitted packages [56].

The citation for John Chambers' 1998 Association for Computing Machinery Software award stated that S has "forever altered how people analyze, visualize and manipulate data". The R project enlarges on the ideas and insights that generated the S language. It has extensive and powerful graphics abilities that are tightly linked with its analytic abilities [57].

Like the popular general purpose dynamic languages, R is free, open source and cooperatively developed. Many of the leading figures in statistical computing are closely associated with the development of R. The basic R system is also supplement by a sort of large standard library of recommended packages, in addition there are many user submitted packages available from CRAN, many representing the state of the art in various areas of statistical computing [55].

The R system is developing rapidly, new features and abilities appear every few months [57]. This is in part due to its flexibility and the speed at which new methods can be developed [55].

R is used in many statistics and data analysis courses taught around the world, including at FEP [58].

```
est.pi <- function(n=2000) {
    x <- runif(n) # n random numbers
    y <- runif(n) # between 0 and 1
    # square and add them (Xi^2 + Yi^2)
    aux <- (x[]^2) + (y[]^2)
    aux2 <- c() # create an empty array
    # if the point is inside the (1/4th of a)
circle of radius 1
    for (i in 1:length(aux)) {
       if (aux[i] < 1)
              aux2[i] <- TRUE
       else
            aux2[i] <- FALSE
    }
    # pi = (number of points in the circle /
total points) * 4
    result <- (sum(aux2) / n) * 4
    print(result)
}
```

Example 4 - R code to estimate the value of Pi using a statistical approach

3.4. Biology

Dynamic languages, namely Python, Ruby, Perl, Lisp and R, have been gaining popularity in Bioinformatics research with toolkits such as BioPython [46], BioPerl [59], BioRuby [60], BioBike (formerly BioLisp) [61] and BioConductor (written in R) [62]. There is also available a BioJava toolkit [63]. While BioConductor has a less general purpose, being dedicated to the analysis and comprehension of genomic data, the rest are general purpose Bioinformatics toolkits and direct competitors. A comparison between BioPython, ByoPerl, BioLisp and BioJava found that for small programs (<500 lines) used only by their authors Perl/BioPerl is the ideal. However for beginners and for larger programs, especially those to be shared and supported by others, Python/BioPython was better due to Python's clarity and brevity [64]. Python is well suited for biology because it is easy to learn, it has the features necessary to scale from small components to large applications and a rich library of modules for scripting and network programming that are essential to bioinformatics due to it's common reliance on the integration of existing tools [50].

Some success cases of Python in biology [10] include Simulating Biomolecules with Python [65] and AstraZeneca's use of Python for collaborative drug discovery [66].

Python is taught at the Pausteur Institute in Bioinformatics [67].

3.5. Mathematics

Python has become a successful language in Mathematics. Python is similar to MATLAB, a very popular language in scientific computing [68].

```
function [x,det] = gaussElimin(a,b)
n = length(b);
for k = 1:n-1
    for i = k+1:n
        if a(i,k) ~= 0
            lam = a(i,k)/a(k,k);
            a(i,k+1:n) = a(i,k+1:n) -
lam*a(k,k+1:n);
            b(i)= b(i) - lam*b(k);
        end
    end
end
det = prod(diag(a));
for k = n:-1:1
```

```
    b(k) = (b(k) - a(k,k+1:n)*b(k+1:n))/a(k,k);
end
x = b;
```

Example 5a: MATLAB code - solution of simultaneous equations Ax = b by Gauss elimination (from [68]).

```
from numarray import dot
def gaussElimin(a,b):
    n = len(b)
    for k in range(0,n-1):
        for i in range(k+1,n):
            if a[i,k] != 0.0:
                lam = a [i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n] -
lam*a[k,k+1:n]
                b[i] = b[i] - lam*b[k]
for k in range(n-1,-1,-1):
    b[k] = (b[k] -
dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
return b
```

Example 5b: Python code - solution of simultaneous equations Ax = b by Gauss elimination (from [68]).

Python's main advantage in Mathematics seems to be it's easy to read [68]. There are also many modules for Numeric and Scientific computation available [63] beyond the standard library.

```
def gcd(a,b):
   while b:
      a,b = b, a % b
   return a
```

Example 6: Python Code to determine the greatest common denominator of two number written by Kirby Umer

A project that deserves special attention is SAGE. SAGE is a computer algebra system written in Python for studying a huge range of mathematics, including algebra, calculus, elementary to very advanced number theory, cryptography, numerical computation, commutative algebra, group theory, combinatorics, graph theory, and exact linear algebra [65]. Also worthy of special note are NumPy, the fundamental package for scientific computing with Python [69] and SciPy which is an Open Source library of scientific tools for Python [70].

4. Other Successful Use of Dynamic Languages

4.1. Web Applications

Dynamic Languages can be used to Web Applications development. There are some languages/projects directed specially to this type of development like Perl, PHP, JavaScript, Ruby on Rails and Django.

JavaScript is a dynamically-typed language, is high-level, and has at least two open source implementations. It is supported by all major web browsers, and is part of a huge number of websites. Too many applications have been built using it, especially on the client-side of the web transaction, like webmail interfaces and blogging tools. JavaScript is defined as the language of the browser as it has to combine strict security requirements with odd user interface challenges.

Zope is an open source web application written in Python that is used by many companies like NASA, U.S. Navy, Red Hat and Viacom[71].

Perl is one of the most popular languages of web development because of its text manipulation capability and its quickly development cycle. It's widely known as "the duct-tape of the Internet". It works with HTML and XML and supports unicode, the Perl's CGI.pm module makes handling HTML forms simple. Perl can handle encrypted data and if embedded into Web servers can speed up processing by 2000%. Amazon and Slashdot use Perl [80].

PHP is a powerfull imperative and object-oriented language. It is open source and it is widely used to build dynamic web pages, like Wikipedia and digg.com[72].

Ruby on Rails is a free web application framework that aims to increase the speed and ease of development of database-driven web sites because it is possible to create applications using pre-defined structures. It's written in Ruby programming language and it is also known by RoR or Rails. The applications created with Rails framework are developed based on MVC (Model-View-Controller). It's a meta-framework because it is a union of five frameworks: active record, active pack, active mailer, active support and active webservices. Ruby on Rails is very appreciated by web developers as it has a rapid application development for web applications, such as a enforced good architecture, support for agility, an instant feedback, support for Web 2.0 and innovative ideas [73]. Ruby on Rails allows one to create complex web applications spending less time, as it is presented at [94] and [95] , there are some real time videos where is created complex application in ten or fifteen minutes. There are some sites that use Rails, like odeo [96] that is a site to record and share audio and basecamp [97] an online collaboration software.

Django [98] is a high-level web framework written in Python and allows a rapid development. It was developed to manage news' sites of the World Company, and today is very used by some developers around the world. World Online is still using Django for all its web sites, Washington Post's web site uses Django for database projects.

4.2. Games

Game programming is usually focused on runtime performance and memory use, typically built in C/C++, which results in much optimized systems that take considerable time to develop and maintain. Dynamic Languages can be used to prototype or develop small arcade-like games.

Python is being used by professional game developers. Some popular games including Totally Games award winning Star Trek Bridge Commander, Freedom Force and Earth & Beyond have already been released that use the Python programming language in varying degrees. Python has found a niche in game development because it has characteristics that are very different from traditional game development languages. It allows developers to quickly build systems that are flexible and can be easily driven by data. Systems written in traditional game development languages (usually C or C++) take much longer to write, and although they might execute faster, are more difficult to change. Python has found a place in the dynamic environment of game development as a tool that allows developers to implement ideas quickly and provides flexibility during the development process. Programmers who switch from C and C++ to Python find that their productivity rises dramatically. Python is being used in three major ways in game development. It is used as a full-fledged programming language to develop real software systems; it is used as a scripting language to control and interface between systems written in other languages; and it is used as a data language to describe game rules and game objects. [74]

Pygame is a set of Python modules designed for writing games which has been used by many games [75]. Another use of python was that of the game Civilization IV where it is used to develop game modds [76].

Another important dynamic language that is very used in game development is Lua [77], a powerful, fast, light-weight, embeddable scripting language that gives a viable scripting interface.

As a compiled binary, Lua is very small by code standards. Coupled with it being relatively fast and having a very lenient license, it has gained a following among game developers for providing a viable scripting interface.

In the popular fantasy massive multiplayer online role-playing game, World of Warcraft [78], Lua is used to allow users to customize its user interface. In Far Cry [79], it is used to script a substantial amount of the game logic, manage game objects, configure the HUD and store other configuration information. it's also used in Crysis [80] , Dawn of War [81], Company of Heroes [82].

5. Few or No Successful Use of Dynamic Languages

5.1. Safety-Critical Applications

Safety-Critical Systems [83]are systems whose failure can result in death, or injury, to people; lost, or severe damage, of equipment or environmental danger. Developing safety-critical applications is a field where Dynamic Languages are yet to be proven successful, or not. However, that same success can be predictable by comparing some of the features of these languages with those of languages used in critical applications development.

ADA is a high-level language with advanced functionalities, clear and unambiguous syntax and semantics that has long been used to develop efficient and analyzable safety-critical software. Its most important features include static, strong typing (the opposite of most languages considered dynamic) which can demand more attention but will prevent many potential flaws; compile-time verifications and run-time checking; exception handling and object-oriented[84][85].

These features encourage clarity in the design, readability-modularity and predictability which fits perfectly in the concept of what the emerging security-safety market demands[84].

Dynamic Languages have some crucial differences to ADA programming language and, therefore, they do not present themselves as a reliable alternative to safety-critical software development nowadays.

There are, however, some exceptions: the Frequentis Project[86], is a safety-critical system that uses Python.

5.2. High Performance Applications

Dynamic Languages have been used to build high-performance systems several times. However, its use has always been associated with other languages, typically static ones.

Many tasks performed by systems, such as numeric computation, machine code generation and low-level interfacing, present no ambiguity about the requirements and the main concepts. Therefore, these application will present better performance if written in languages that have been optimized over several decades to perform such tasks. Most of these tasks also share an overwhelming concerning performance and in this field dynamic languages present highly unsatisfactory results[3].

Dynamic languages obtain most of their power through specialization in certain areas and therefore do

not present a reliable option when it comes to high-level performance systems. In many cases, a successful alternative would be to develop applications using dynamic languages combined with other languages. The result, in most cases, would be a combination of flexibility and effective performance[3][87].

5.3. Low Memory Systems

Dynamic languages, due to its features(high-level and interpreted), require more machinery to execute than languages that are either low-level or get compiled to machine code. Therefore, when it comes to small memory systems, applications developed using dynamic languages are an inappropriate choice.

Nowadays, however, systems once considered low memory, such as mobile phones, are how built with enough memory to run higher-level applications[3].

6. Fanaticism

My disenchantment with my job grows daily. The more I use Rails, the more I dislike the absolute mountain of crap that is J2EE. Certainly I want to do the best I can for Webify but there is no joy in working with J2EE anymore. [90]

Some dynamic languages, namely Ruby, have very vocal supporters that might soon start to tout it as an antidote to poverty, disease and social injustice [93]. Ruby fanboys are also infamous for not being very good at handling criticism - “Why do you need static type checking? Are you going to try to regexp match an Animal? (...) Ruby doesn’t need to be more safe. It certainly doesn’t need static type checking. What we need is to kick the idiots out, and educate those with less understanding. Doing so improves those individuals, ourselves, and our craft as a whole” [91]

Python too is sometimes associated with some rather impressive claims like the claim that using it reduces code volume in 80 to 90 percent in comparison to C++ or Java for all applications [92].

7. Conclusion

Research has shown that dynamic languages such as Python and Ruby are indeed easier to learn and read, closer to language neutral pseudo-code commonly used in computer science text books, they are more expressive and more importantly offer a higher level of abstraction.

Dynamic languages have certainly been successfully used in solving many problems in specific domains and have enjoyed an extraordinary success on the web. While they are still a rarity in some applications these are becoming fewer, specially as some of these are limited by hardware which is constantly improving and therefore becoming less of an issue.

Dynamic languages look set to succeed system languages in the same manner these succeeded assembly languages. Python's use of C extension modules in practice works much like assembly macros used from C code that were once popular in game programming.

8. Acknowledgements

The authors would like to acknowledge the assistance of André Moreira, A. J. Fernandes and Fábio Aguiar.

Appendix A:

List of All Programming Languages that can be considered Dynamic Languages
PL, Befunge, ChucK, Curl, dBASE, (dBL), ECMAScript, ActionScript, DMDScript, E4X, Io, JavaScript, JScript, Eiffel, Erlang, Forth, Groovy, HyperTalk, Lisp, Dylan, Logo, Scheme, Lua, Maude, system, Oberon, Objective, Modula-2, Objective-C, Perl, PHP, Pliant, POP-11, Poplog, PowerShell, Pike, Prolog, Python, REBOL, Revolution, Ruby, Smalltalk, Bistro, Self, Slate, Squeak, Snobol, SuperCard, SuperCollider, Tcl, XOTcl, TeX, macro, language, VBScript, Visual, Basic, 9.0, Visual, FoxPro, Windows, PowerShell, xHarbour. [4]

References

[1] James Gosling, The Java Language Environment, May 1996
[2] Paul Graham, Revenge Of The Nerds, May 2002

[3] Dynamic Languages — ready for the next challenges, by design. David Ascher, 27 July 2004
[4] http://en.wikipedia.org/wiki/Dynamic_language, last consulted in 18 November 2007
[5] http://wiki.tcl.tk/13325, 18 November 2007
[6] Programmers Shift to Dynamic Languages – Linda Dailey Paulson, IEEE Computer, February 2007
[7] http://blogs.msdn.com/hugunin/archive/2004/08/23/219186.aspx, Jim Hugunin, creator of IronPython and Jython, last consulted in 18 November 2007

[8] Python Reference Manual, Guido van Rossum, 19 September 2006
[9] John K. Ousterhout, March 1998, "Scripting: Higher Level Programming for the 21st Century", IEEE Computer

[10] http://www.python.org, last consulted in 21 November 2007
[11] JRuby, http://jruby.codehaus.org/, last consulted in 22 November 2007
[12] http://www.jython.org, last consulted in 21 November 2007
[13] Ruby.NET, http://rubydotnet.googlegroups.com/web/Home.htm, last consulted in 22 November 2007
[14] Gary Pollice, WPI Computer Science, CS544 Lect Compiler vs. Interpreter
[15] Mark Lutz and David Ascher, Learning Python 2nd Edition, Chapter 1.3
[16] Tim Lindholm and Frank Yellin, 1999, The Java Virtual Machine Specification 2nd Edition, Chapter 1.2
[18] http://www.codeplex.com/IronPython, last consulted in 21 November 2007
[17] John Aycock, 2003, A brief history of just-in-time, ACM Computing Surveys
[19] Intel ICC, http://www.intel.com/cd/software/products/asmo-na/eng/compilers/cwin/index.htm, last consulted in 21 November 2007

[20] http://www.atdot.net/yarv, last consulted in 21 November 2007
[21] http://rubini.us, last consulted in 21 November 2007
[22] Ruby 1.8.6, http://www.ruby-lang.org, last consulted in 21 November 2007
[23] Guido van Rossum, Extending and Embedding the Python Interpreter, September 2006
[24] Luca Cardelli, Type Systems, CRC Handbook of Computer Science and Engineering 2nd Edition, Ch. 97, 25 February 2004
[25] Anthony A. Aaby, 15 July 2004, Introduction to Programming Languages

[26] Erik Meijer and Peter Drayton, August 2004, Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages

[27] The Open Source Definition, Open Source Initiative, http://www.opensource.org/docs/definition.php, last consulted in 20 November 2007
[28] Eric S. Raymond, 1999, The Cathedral and the Bazaar
[29] http://en.wikipedia.org/wiki/Open_source, last consulted in 20 November 2007
[30] Bill Venners, January 2003, The Making of Python - A Conversation with Guido van Rossum
[31] http://www.perl.org/about.html, last consulted in 20 November 2007
[32] Python Core Development, http://www.python.org/dev/, last consulted in 24 November 2007
[33] Ruby Core, http://www.ruby-lang.org/en/community/ruby-core/, last consulted in 24 November 2007

[34] Brooks, Jr., F.P. The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition. Reading, MA: Addison-Wesley, 1995
[35] A.M. Kuchling, Python Advocacy HOWTO, http://www.amk.ca/python/howto/advocacy/, last consulted in 29 November 2007

[36] Frank Stajano, Python in Education: Raising a Generation of Native Speakers, http://www.cl.cam.ac.uk/~fms27/papers/python-native-speakers.html, last consulted in 25 November 2007
[37] Pai H. Chou, Algorithm Education in Python, http://www.ece.uci.edu/~chou/py02/python.html, last consulted in 28 November 2007

[38] Toby Donaldson, Python as a First Programming Language for Everyone, http://www.cs.ubc.ca/wccce/Program03/papers/Toby.html, last consulted in 28 November 2007

[39] MIT, http://web.mit.edu/, last consulted in 30 November 2007

[40] University of Cambridge, http://www.cam.ac.uk/, last consulted in 30 November 2007

[41] Peter Robinson, From ML to C via Modula-3 an approach to teaching programming, December 1994.
[42] A Note on MIT's Introductory Computing Subject, http://mitpress.mit.edu/sicp/course.html, last consulted in 25 November 2007
[43] Preparing to study Computer Science, http://www.cl.cam.ac.uk/admissions/undergraduate/preparation/, last consulted in 25 November 2007
[44] Bjarne Stroustrup, Learning Standard C++ as a New Language, The C/C++ Users Journal, May 1999

[45] General Python FAQ, http://www.python.org/doc/faq/general/, last consulted in 26 November 2007

[46] BioPython, http://biopython.org, last consulted in 29 November 2007
[47] The dir() function, http://docs.python.org/tut/node8.html#SECTION008300000000000000000, last consulted in 28 November 2007
[48] Lingyun Wang and Phil Pfeiffer, A Qualitative Analysis of the Usability of Perl, Python, and Tcl, 2002

[49] Guido van Rossum, Computer Programming for Everybody, July 1999
[50] Katja Schuerer et al., Introduction to Programming using Python - Programming Course for Biologists at the Pasteur Institute, 21 February 2007
[51] http://hacketyhack.net, last consulted in 28 November 2007
[52] Stuart Russel and Peter Norvig, Artificial Intellience: A Modern Approach, Second Edition, 2003
[53] Peter Norvig, Python for Lisp Programmers, http://www.norvig.com/python-lisp.html, last consulted in 28 November 2007
[54] CIS 391 - Artificial Intelligence, http://www.cis.upenn.edu/~cse391/, last consulted in 28 November 2007

[55] John Fox and Robert Andersen, Using the R statistical computing environment to teach social statistics courses, January 2005

[56] http://cran.r-project.org, last consulted in 29 November 2007

[57] J. H. Maindonald, Using R for Data Analysis and Graphics, October 2004
[58] Faculdade de Economia da Universidade do Porto, http://sigarra.up.pt/fep, last consulted in 29 November 2007
[59] BioPerl, http://www.bioperl.org, last consulted in 29 November 2007
[60] BioRuby, http://bioruby.org, last consulted in 29 November 2007
[61] BioBike, http://nostoc.stanford.edu/Docs/, last consulted in 29 November 2007

[62] BioConductor, http://www.bioconductor.org, last consulted in 29 November 2007
[63] BioJava, http://biojava.org, last consulted in 29 November 2007

[64] Harry Mangalam, The Bio* toolkits – a brief overview, Briefings in Bioinformatics. VOL 3. NO 3. 296–302. September 2002
[65] SAGE, http://www.sagemath.org, last consulted in 29 November 2007
[66] Molecular Modelling Toolkit, http://dirac.cnrs-orleans.fr/MMTK/, last consulted in 29 November 2007
[67] Python course in Bioinformatics at the Pasteur Institute, http://www.pasteur.fr/recherche/unites/sis/formation/python/, last consulted in 29 November 2007
[68] Jaan Kiusalaas, Numerical Methods in Engineering with Python, Cambridge University Press, 2005
[69] NumPy, http://numpy.scipy.org, last consulted in 29 November 2007
[70] SciPy, http://www.scipy.org, last consulted in 29 November 2007

[71] Zope, http://www.zope.org/, last consulted in 30 November 2007

[72] www.php.net, last consulted 29 November 2007

[73] http://www.sda-india.com/conferences/jax-india/sessions/Neal_Ford/Neal_Ford-Why_is_Everyone_so_Excited_about_Ruby_on_Rails-slides.pdf, last consulted 29 November 2007

[74] Sean Riley, Game Programming With Python, October 2003 30 November 2007

[75] Pygame, http://www.pygame.org/news.html, last consulted 27 November 2007

[76] http://www.2kgames.com/civ4/blog_03.htm, last consulted in 30 November 2007

[77] www.lua.org, last consulted in 30 November 2007

[78] www.worldofwarcraft.com, last consulted in 30 November 2007

[79] http://infarcrygame.ubi.com, last consulted in 30 November 2007

[80] www.ea.com/crysis, last consulted in 30 November 2007

[81] www.dawnofwargame.com, last consulted in 30 November 2007

[82] www.companyofheroesgame.com, last consulted in 30 November 2007

[83] http://en.wikipedia.org/wiki/Safety-critical, last consulted in 30 November 2007
[84] http://www.esemagazine.com/index.php?option=com_content&task=view&id=411&Itemid=2, last consulted in 30 November 2007
[85] http://en.wikipedia.org/wiki/Ada, last consulted in 30 November 2007

[86] Frequentis, http://www.python.org/about/success/frequentis/, last consulted in 30 November 2007

[87] James Gosling: "Java Is Under No Serious Threat From PHP, Ruby or C#",http://extentech.sys-con.com/read/193146.htm, May 12, 2007

[88] What's Wrong With Ruby, http://www.bitwisemag.com/2/What-s-Wrong-With-Ruby, last consulted in 30 November 2007
[89] Hacknot, http://www.hacknot.info, last consulted in 30 November 2007

[90] Mike Perham, Java Developer, in The State of Ruby on Rails, David Heinemeier Hansson, 2005

[91] Pat Maddox, Java People Must Be Stupid, http://evang.eli.st/blog/2007/1/22/java-people-must-be-stupid, January 22nd, 2007, last consulted in 30 November 2007
[92] Stephen Deibel in Programmers Shift to Dynamic Languages, Linda Dailey Paulson, IEEE Computer, February 2007
[93] Matthew Huntbach , What's Wrong With Ruby, http://www.bitwisemag.com/2/What-s-Wrong-With-Ruby, 16 March 2007

[94] http://www.rubyonrails.org/screencasts, last consulted in 30 November 2007
[95] http://hobocentral.net/screencasts.php, last consulted in 30 November 2007
[96] http://www.odeo.com, last consulted in 30 November 2007
[97] http://www.basecamphq.com, last consulted in 30 November 2007
[98] http://www.djangoproject.com, last consulted in 30 November 2007