An Analysis of Scripting Languages for Research in Applied Computing

Olugbenga Oluwagbemi, Adewole Adewumi, Folakemi Majekodunmi
Department of Computer and Information Sciences
Covenant University
Ota, Nigeria

Sanjay Misra
Department of Computer Engineering
Atilim University
Ankara, Turkey
smisra@atilim.edu.tr

Luis Fernandez-Sanz
Department of Computer Science
University of Alcala
Madrid, Spain

***Abstract*—There are several scripting languages that exist today. However, some are used more frequently and popular than others. This is due to certain characteristics and features that they possess. Particularly in applied computing fields like software engineering, bioinformatics and computational biology, scripting languages are gaining popularity. This paper presents a comparative study of ten popular scripting languages that are used in the above mentioned fields/area. For making comparison, we have identified the factors against which these languages are evaluated. Accordingly, based on selected criteria we determine their suitability in the fields of software engineering, bioinformatics and computational biology research. This will serve as a guide to researchers to choose the appropriate scripting language in the various fields.**

Keywords—applied computing;open source; scripting languages.

I. INTRODUCTION

Scripting languages are an important tool in present day applied computing research. There are several reasons why scripting languages are popular especially in applied computing research. Scripting languages are object-oriented in nature, easy to learn and apply, they have flexible syntax, and powerful string-handling abilities, portable, embeddable, extensible, rich sets of libraries and some of them also provide support for concurrent programming [1].

Scripting languages find applications in different applied computing areas such as software engineering, bioinformatics and computational biology. Software engineering for instance being a field that concerns itself with applying systematic and disciplined approaches to the development of quality software that meets client/user requirements has stages of software development [52]. Scripting languages come in handy in the implementation and testing phases of software development. Also, scripting languages find applications in bioinformatics – being a field that involves researching, developing and applying computational tools and approaches in order to expand the use of biological, medical, behavioral or health data [53]. This also includes acquiring, storing, organizing, archiving, analyzing and visualizing such data. Scripting languages are crucial in this regard. In addition, scripting languages play a vital role also in computational biology as this field involves mining large pools of biological data using mathematical/computational modeling techniques [53].

Some of the problems associated with developing efficient, effective and portable software are those related to improper specifications, error in the design phase, faulty implementation phase, and lack of a well tested and properly refined product in the case of software engineering [52]. The major challenge in bioinformatics and computational biology research has to do with analyzing large volume of biological data for essential information. The reason for undertaking a study on the application of scripting languages to applied computing is to address these challenges and also specify the most suitable scripting language applicable to a specific problem domain.

Research in the three fields discussed relies heavily on the use/development of tools. Scripting languages are often employed in the development of such tools. The aim of this paper therefore is to present a comparative study of ten scripting languages commonly used in academic circles with the intent of determining their suitability for three applied computing fields namely: software engineering, bioinformatics and computational biology. The rest of the paper is organized as follows: Section 2 discusses the background of scripting languages. Section 3 presents features, advantages and limitations of ten popular scripting languages as given by TIOBE – a company that assesses the quality of software. Section 4 presents a comparative study of the scripting languages based on some defined criteria. Section 5 discusses the outcomes of the comparative study and Section 6 concludes the paper.

II. Background of Scripting Languages

In this paper, the limitation is that, we are confining our study to the applications of scripting languages to software engineering, bioinformatics and computational biology –related research. We also provide insight on the various attributes/features of specific scripting languages considered in this study.

The term 'scripting language' has been defined from two perspectives namely: the pragmatic perspective and the philosophical perspective [1]. The two perspectives however agree on the fact that scripting languages are interpreted [2] possess automatic memory management and powerful operations tightly built in, rather than relying on libraries [1]. In recent times, they also possess dynamic and strong typing as seen in Python, Perl and a host of others.

Scripting languages play an indispensable role in computational and biological research as well as software engineering. Their significance cannot be underestimated. Scripting languages originated as a result of the development of the internet as a tool of communication. Rather than being compiled, scripting languages are usually being interpreted. Researchers in computational and biological sciences have several research problems, and needs. In order to solve these problems and accelerate the pace of progress in their various research domains, it is expedient that these scientists understand, solve these problems and meet these needs. Thus, the underlying motivation for the development and use of different scripting languages is the evolution of diverse problems and the complex need to work with incomplete and noisy data. Also, in software engineering particularly in the area software quality, several metrics are being proposed but there are usually no corresponding tools that can be used for measurement. In building these tools, scripting languages may come in handy. Therefore, in the next section we review ten popular scripting languages by examining their features, advantages, and limitations.

III. Features, Advantages and Limitations of Popular Scripting Languages

This section describes the features, advantages and limitations of Python, Haskell, Lua, Perl, Scala, PHP, JavaScript, Erlang, R and Ruby as ten popular scripting languages. They are popular in the sense that they rank higher in comparison to other known scripting languages in the TIOBE programming community index [54]:

A. Python

Python is a general-purpose, high-level programming language that also provides scripting capability [3]. It first appeared in 1991 and was designed by Guido van Rossum. The language was influenced by ABC, ALGOL 68, C, Haskell, Lisp, Modula-3, Perl, and Java. It has also influenced the design of other languages namely: Boo, Cobra, D, Falcon, Groovy, Ruby, and JavaScript.

B. Haskell

It is an advanced, purely-functional programming language that supports scripting capabilities [55]. It first appeared in 1990 and is an open-source product of more than twenty years of cutting-edge research which allows rapid development of robust, concise, correct software. The language was influenced by languages like: Standard ML, Lisp, and Scheme. It has in turn also influenced several other languages like: Python and Scala

C. Lua

Lua is a powerful, fast, lightweight, embeddable language that first appeared in 1993. "Lua" (pronounced LOO-ah) means "Moon" in Portuguese [56]. Roberto et al. [57] designed the language. The language was inspired by C++, CLU, Modula, Scheme and SNOBOL. It has in turn inspired languages like: Io, GameMonkey, Squirrel, Falcon and MiniD.

D. Perl

Perl is a highly capable, feature-rich programming language that first appeared in 1987 [58]. It was developed by Larry Wall and can be used in mission critical projects. The language was influenced by languages like: AWK, Smalltalk 80, Lisp, C, C++, sed, UNIX shell, and Pascal. It has in turn influenced the creation of Python, PHP, Ruby, JavaScript, and Falcon.

E. Scala

It is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It was designed by Martin Odersky and first appeared in 2003 [59]. The language was inspired by languages like Eiffel, Erlang, Haskell, Java, Lisp, Pizza, Standard ML, OCaml, Scheme and Smalltalk. It has in turn influenced the following languages namely: Fantom, Ceylon, and Kotlin.

F. PHP

It is a widely used general purpose scripting language that is especially suited for Web development and can be embedded into HTML. It was designed by Rasmus Lerdorf [60] using the C programming language and first appeared in 1995. PHP was influenced by Perl, C, C++, Java and Tcl.

G. JavaScript

JavaScript is a lightweight programming language that first appeared in 1994 and was designed by Brendan Eich [61]. The language was influenced by C, Java, Perl, Python, Scheme, Self. It has in turn influenced ActionScript, CoffeeScript, Dart, Jscript .NET, Objective-J, QML, TIScript, and TypeScript.

H. Erlang

Erlang is a programming language designed at the Ericsson Computer Science Laboratory. It first appeared in 1986. The language was influenced by Prolog and ML. It has in turn influenced F#, Clojure, Rust, Scala, Opa and Reia [62].

I. R

R is a language and environment for statistical computing and graphics. It was designed by Ihaka and Gentleman and first appeared in 1993 [63]. It was influenced by S, Scheme, and XLispStat.

J. Ruby

It is a dynamic open source programming language with a focus on simplicity and productivity. It was designed by Yukihiro Matsumoto [64] and first appeared in 1995. The language was influenced by Ada, C++, CLU, Dylan, Eiffel, Lisp, Perl, Python, and Smalltalk. It has also in turn influenced Falcon, Fancy, Groovy, loke, Mirah, Nu, and Reia.

Table 1 summarizes the features, advantages and limitations of these scripting languages.

TABLE I. SCRIPTING LANGUAGES' FEATURES, ADVANTAGES AND LIMITATIONS

Scripting Language	**Features and advantages**	**Limitations**	**References**
Python	-Object-oriented -Clear and readable syntax -Imperative -Functional -Procedural -Reflective -Strong and dynamic typing -Simple and easy to learn -Free and Open Source -Portable -Extensible -Embeddable -Extensive Libraries -Rapid development -General purpose language	-The indentation style of Python may put programmers off who have been exposed to other languages before Python. -Python 3 which is an improvement over Python 2 is different from Python 2. Not all the libraries in Python 2 currently work in Python 3	[4] [5]
Haskell	-Object-oriented -Pure functional language -Multi-platform -Derives the best features from other languages -It is innovative by constantly incorporating new language features -It is a general purpose language -Open source -Extensible -Encourages literate programming -Flexible syntax -Powerful string handling	-A few keywords are not reserved -Learning curve is high -Code refactoring is difficult to perform -Weak debugging tools	[6] [7]
Lua	-Proven and robust language -Faster when compared to other interpreted scripting languages -Portable -Embeddable -Powerful yet simple -Small in size -Free and Open source -Proto-typical	-Limited error handling support -No Unicode support -Limited pattern-matching support	[8]
Perl	-Functional -Imperative -Object-oriented -Reflective -Procedural -Generic -Portability -String processing and especially regular expression support -CPAN (Comprehensive Perl Archive Network) comes with a range of useful third party modules -Prototype based	-Slow execution -You cannot easily create a binary image ("exe") from a Perl file. -Error handling is often challenging	[9] [10] [11]
Scala	-Seamless integration with Java -It is compiled, not interpreted -Object-oriented -Functional -Extensible -Statically typed -Interoperates with Java and .NET	-New but growing steadily	[12]
PHP	-Imperative -Object-oriented -Procedural -Reflective -Weak typing -Open source	-Best suited for web applications -Insecure	[13]
JavaScript	-Prototype-based -Weakly-typed -Possesses first-class function -Multi-paradigm -Object-oriented -Imperative programming style -Functional programming style -Supports structured programming -Easy learning curve -Simplicity -Speed -Versatile and plays nicely with other languages	-Security issues -JavaScript rendering varies since different layout engines render JavaScript differently	[14] [15] [16] [17] [18] [19] [20] [21]
Erlang	-Functional programming -Supports concurrent programming -Support for distributed, fault-tolerant, soft-real-time, non-stop applications -Support for hot swapping so that code can be changed	- Learning curve is high at the beginning	[22]

Scripting Language	**Features and advantages**	**Limitations**	**References**
	without stopping a system -General purpose language		
R	-Object-oriented -Imperative -Functional -Dynamic typing -Procedural -Cross-platform -Extensible -Provides a wide variety of statistical and graphical techniques -Visualization -Open source -Library support -Proto-typical	-Steep learning curve	[23]
Ruby	-Object-oriented -Multi-platform -Derives the best features from other languages -Open source -Low learning curve -Extensible -Encourages literate programming -Flexible syntax -Rich set of libraries -Powerful string handling	-Slower compared to other scripting languages like PHP	[24] [25]

IV. Comparative Study of Popular Scripting Languages

In the previous section, we explored the features, advantages and limitations of ten scripting languages. After exhaustive survey, we identified the following attributes against which the scripting languages will be evaluated and compared.

TABLE II. COMPARISON OF THE SCRIPTING LANGUAGES

Language	**Comparison Criteria**				
	Applicability	***Ranking /Popularity***	***Prominent Users***	***Learning Curve***	***Length of existence***
Python	General Purpose	8	Google, NASA, Yahoo, Red Hat	Low	21years
Haskell	General Purpose	32	Alcatel-Lucent, AT&T, Bank of America Merril Lynch, Ericsson AB, Facebook, Google	High	22years
Lua	General Purpose	18	Firefox Web browser, MediaWiki	Low	19years
Perl	General Purpose	9	Amazon.com	Low	25years
Scala	General Purpose	33	Twitter, LinkedIn	Low	9years
PHP	Web Applications	6	Facebook	Low	17years
JavaScript	Web Applications	10	Apache Cordova	Low	18years
Erlang	General Purpose	31	Facebook, Ericsson	High	26years
R	Statistical Computing	26	New York Times, Google, Facebook, Mozilla, TechCrunch	High	19years
Ruby	General Purpose	11	NASA Langley Research Center, Motorola	Low	17years

- Applicability: Can the language be used in different contexts or specific contexts?
- Ranking/Popularity: What is its position in TIOBE language rankings – TIOBE is a company specialized in assessing and tracking the quality of software
- Prominent Users – Which prominent company is using the software?
- Learning Curve: Difficulty for new entrants
- Length of existence: Time span since its development.

The comparison is given in Table 2 and in Figure 1 we show the language rankings based on the popularity as given by TIOBE. It is important to note that in Figure 1, the languages with the lower rankings are better in terms of popularity.

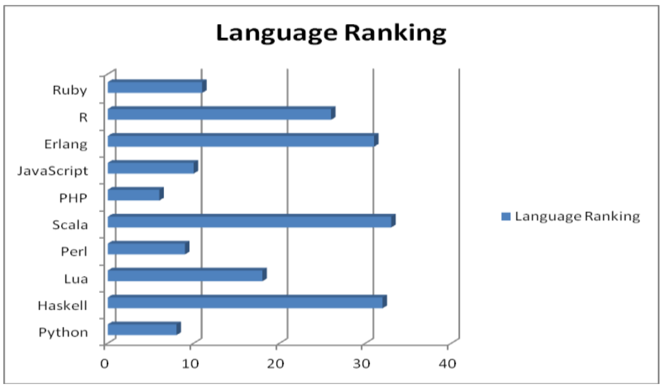

Fig. 1. Popularity ranking (lower is better)

V. Discussion

Based the findings in previous sections, in this section, we discuss the suitability of the scripting languages considered for research in the fields of software engineering, bioinformatics and computational biology.

A. Software Engineering

The central theme of software engineering research is coming up with quality software that meets user requirements and is completed on time and within a specified budget [26]. This brings four stages of the software development process to bear. First, the requirements engineering phase, and since user requirements are always evolving there is need for a language that can also adapt to the changing requirements hence the need for scripting languages. Such a scripting language must allow for the creation of prototypes as a way of gathering requirements. Among the languages considered, JavaScript, Lua, Perl, and R have this feature. The implementation or coding phase is another stage where scripting languages are needed. This is to ensure speedy completion and delivery of software. The scripting language must be one that promotes programmer productivity while also ensuring code reuse [2]. Python, Ruby, PHP, Haskell, Lua, Scala, Perl, JavaScript and Erlang all come in handy in this regard. Testing is crucial before any software can be delivered to a client hence testing is a third stage where scripting languages come into play during the software development lifecycle. With the advent of Test Driven Development [28] and its adoption by academia [29] particularly its integration into the computing curriculum [30], automated unit tests [27] can be created that define code requirements before writing the actual code. Software engineers often use testing frameworks for this and these testing frameworks are written in languages like Ruby, Python, Perl and PHP. In situations where reliability of an application cannot be compromised, Erlang should be used as it is designed for writing reliable and fault-tolerant applications.

B. Bioinformatics

Bioinformaticians constantly mine large volumes of gene data, perform DNA analysis which involves the comparative analysis of sequences, processing of plant and animal sequences with respect to infectious disease research. Some well suited scripting languages currently used in bioinformatics for this purpose include Python, Perl, and R as is currently being used. High level programming languages like C++, Java have been used in previous works to develop bioinformatics tools Some useful Bioinformatics toolkits have been developed using Java, Perl and Python programming languages [31]; Other Bioinformatics-related tools have been developed and written in diverse programming languages [32], the Bioconductor (a bioinformatics and computational biology software project), was developed using R programming language and environment [33], another bioinformatics tool was partly developed using XML and Java [34]; High-level programming language like C++ was used in developing libcov (a bioinformatics tool for manipulating protein structures) [35]; MACBenAbim, a bioinformatics and computational biology application was developed using JavaScript and HTML 5 [36]; BengaSavex, a computational biology extraction tool for identical DNA sequences was developed using C++ programming language [37]; 13CFLUX2, a high-performance bioinformatics software suite was developed using C++, Java and python add-ons[38]. Of recent, some of the several bioinformatics tools are now been developed using prominent scripting languages such as Perl, Python and JavaScript. Prova is an example of a Java-based bioinformatics tool developed using JavaScript programming language [39]; The Gaggle is another Java-based software, developed with the aim of resolving problems associated with software-database integration[40]; Kumar and Dudley jointly conducted an extensive review about various software used by bioinformaticians [41]; Fourment and Gillings [42], conducted a comparative analysis among programming languages (C, C++, C#, Java, Perl and Python), used by bioinformaticians [42]; Easyfig is a python-based application used to analyze genetic loci in bioinformatics [43]; SurreyFBA is a C++-based bioinformatics application developed for use by scientists [44]; FASIMU is also another bioinformatics application that was developed using some programming languages [45]. These tools span various application areas of bioinformatics. Researchers in this field can also explore functional scripting languages such as Haskell, Lua, Scala and Erlang, as they also hold great promise because, they have been previously used to develop useful applications in bioinformatics..

According to the results of our analysis in this paper (as shown in Figure 1), we discovered that Python had the best ranking, which was closely followed by Perl and then the R scripting language. Python must have ranked the best for developing bioinformatics tools and applications partly because it is easier to learn and gain mastery of it, than the other two scripting languages.

C. Computational Biology

The nature of work done in computational biology requires the use of scripting languages that are either functional in nature or support functional programming hence the following languages can suffice for and has been applied in some computational biology research: Python, Haskell, Perl, Scala, JavaScript, Lua, Ruby, Erlang, and R BioPython, a useful tool for Computational biologists was developed using Python programming language [46]; Bioshell, another computational biology tool, consists of a combination of scripting programs, mostly python and Perl [47]; AnnotationSketch, a computational biology tool was implemented using ANSI C with provision for bindings with scripting languages such as Ruby, Python and Lua [48]; SHOGUN is a machine learning software implemented in C++ with the capability of interfacing to MATLAB, R, Python and Octave. It is a very useful computational biology tool useful for mastering large-scale learning problems for analyzing biological sequences [49]; Fiji, a computational biology tool, was implemented using a broad range of scripting languages [50]; finally, a toxic genomic data analysis system, useful for computational biologists, was implemented using programming languages [51] [38]. Thus scripting languages find expression in the computational biology-related research.

VI. CONCLUSION

In conclusion, this paper has reviewed ten popular scripting languages according to the TIOBE language ranking. The review presented the features, advantages and limitations of each language. A comparative study is also carried out on the languages based on five criteria which include: applicability, ranking/popularity, prominent users, the learning curve and the length of existence of the language. Based on the results from the analysis we were able to deduce which language was suitable for software engineering, bioinformatics and computational biology research. We believe that the analysis will serve as a guide to researchers in the fields of software engineering, bioinformatics and computational biology who are trying to select a suitable scripting language or change from an existing language.

REFERENCES

[1] L. Prechelt, "Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++ and Java," Advances in Computers, vol. 57, pp. 205-270, 2003.

[2] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," Computer, vol. 31, pp. 23-30, 1998.

[3] J. Python, "Python programming language," USENIX Annual Technical Conference, 2007.

[4] T. A. Budd, Exploring Python, Burr Ridge, IL: McGraw-Hill, 2009.

[5] A. Downey, Think Python - How to think like a computer scientist, Needham, MA: Green Tea, 2012.

[6] C. J. Sampson, "Experience report: Haskell in the real world: writing a commercial application in a lazy functional language," ACM SIGPLAN Notices, pp. 185-190, 2009.

[7] I. Pop, "Experience Report: Haskell as a reagent," ACM SIGPLAN International Conference on Funtional Programming, 2010.

[8] R. Ierusalimschy, Programming in Lua, 2nd ed., Rio de Janeiro, Brazil: Lua, 2006.

[9] M. J. Dominus, Higher Order Perl. Burlington, MA: Morgan Kaufmann, 2005.

[10] R. L. Schwartz, B. D. Foy and T. Phoenix, Learning Perl. Sebastopol, CA: O'Reilly, 2011.

[11] Chromatic, Modern Perl. Hillsboro, OR: Onyx Neon, 2011.

[12] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman and M. Zenger, "An overview of Scala programming language," Technical Report IC/2004/64, 2004.

[13] V. Dwarampudi, S. S. Dhillon, J. Shah, N. J. Sebastian and N. Kanigicharla, "Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB.NET, AspectJ, Perl, Ruby, PHP & Scheme-a Team 11," arXiv preprint, 2010.

[14] S. Maffeis, J. C. Mitchell and A. Taly, "Language-based Isolation of Untrusted JavaScript," in 22nd IEEE Computer Security Foundations Symposium, 2009.

[15] M. Dhawan and V. Ganapathy, "Analyzing Information Flow in JavaScript-based Browser Extensions," IEEE Computer Security Applications Conference, pp. 382-391, 2009.

[16] S. Maffeis, J. C. Mitchell and A. Taly, "An Operational Semantics for JavaScript," in Programming languages and systems, Springer, pp. 307-325, 2008.

[17] L. Wilkens, "Objects with prototype-based mechanisms," Journal of Computing Sciences in Colleges, vol. 17, pp. 131-140, 2002.

[18] S. Maffeis, J. C. Mitchell and A. Taly, "Isolating JavaScript with Filters, Rewriting and Wrappers," in Computer Security, Springer, pp. 505-522, 2009.

[19] P. Thiemann, "Towards a Type System for Analyzing JavaScript Programs," in Programming Languages and Systems, Springer, pp. 408-422, 2005.

[20] C. Anderson and S. Drossopoulou, "BabyJ: From Object Based to Class Based Programming via Types," Electronic Notes in Theoretical Computer Science, vol. 82, pp. 53-81, 2003.

[21] A. E. Hassan and R. C. Holt, "Migrating Web Frameworks using Water Transformations," in 27th Annual International Computer Software and Applications Conference, pp. 296-303, 2003.

[22] F. Cesarini and S. Thompson, Erlang progrmming: a concurrent approach to software development. Sebastopol, CA: O'Reilly, 2009.

[23] W. N. Venables and D. M. Smith, "An Introduction to R," 2012.

[24] D. Thomas and D. H. Hansson, Agile Web development with Rails: a Pragmatic Guide, Raleigh, NC: The Pragmatic Programmers, 2006.

[25] B. M. Ren, J. Toman, T. S. Strickland and J. S. Foster, "The Ruby type checker," Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1565-1572, 2013.

[26] M. V. Zelkowitz, "Perspectives on software engineering," ACM Computing Surveys, vol. 10, pp. 197-216, 1978.

[27] E. G. Barriocanal, M. A. S. Urban, I. A. Cuevas and P. D. Perez, "An experience in integrating automated unit testing practices in an introductory programming course," ACM SIGCSE Bulletin, vol. 34, pp. 125-128, 2002.

[28] C. G. Jones, "Test-driven development goes to school," Journal of Computing Sciences in Colleges, vol. 20, pp. 220-231, 2004.

[29] C. Desai, D. Janzen and K. Savage, "A survey of evidence for test-driven development in academia," ACM SIGCSE Bulletin, vol. 40, pp. 97-101, 2008.

[30] C. Desai, D. S. Janzen and J. Clements, "Implications of integrating test-driven development into CS1/CS2 curricula," ACM SIGCSE Bulletin, vol. 41, pp. 148-152, 2009.

[31] H. Mangalam, "The Bio* toolkits - a brief overview," Brief Bioinform, vol. 3, pp. 296-302, 2002.

[32] H. M. Sauro, M. Hucka, A. Finney, C. Wellock, H. Bolouri, J. Doyle and H. Kitano, "Next generation simulation tools: the Systems Biology workbench and BioSPICE integration OMICS," Journal of Integrative Biology, vol. 7, no. 4, pp. 355-372, December 2003.

[33] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Homik, T. Hothom, W. Huber, S. Iacua, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tiemey, J. Y. H. Yang and J. Zhang, "Bioconductor: open software development for computational biology and bioinformatics," Genome Biology, vol. 5, no. R80, 2004.

[34] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senge, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat and P. Li, "Tavema: a tool for the composition and enactment of bioinformatics workflows," Bioinformatics, vol. 20, no. 17, pp. 3045-3054, 2004.

[35] D. Butt, A. J. Roger and C. Blouin, "libcov: A C++ bioinformatics library to manipulate protein structures, sequence alignments and phylogeny," BMC Bioinformatics, vol. 6, p. 138, 2005.

[36] O. Oluwagbemi, A. Adewumi and A. Esuruoso, "MACbenabim: A multi-platform mobile application for searching keyterms in Computational Biology and Bioinformatics," Bioinformation, vol. 8, no. 16, 2012.

[37] O. Oluwagbemi, S. Imolorhe and V. Agozie, "BengaSavex: a new computational genetic sequence extraction tool for DNA repeats," African Journal of Biotechnology, 2013.

[38] M. Weitzel, K. Noh, T. Dalman, S. Niedenfuhr, B. Stute and W. Wiechert, "13CFLUX2-high performance software suite for 13C-metabolic flux analysis," Bioinformatics, vol. 29, no. 1, pp. 143-145, 2013.

[39] A. Kozlenkov and M. Schroeder, "PROVA: Rule-based Java-scripting for a bioinformatics semantic web," Lecture Notes in Computer Science, vol. 2994, pp. 17-30, 2004.

[40] P. T. Shannon, D. J. Reiss, R. Bonneau and N. S. Baliga, "The Gaggle: an open-source software system for integrating bioinformatics software and data sources," BMC Bioinformatics, vol. 7, pp. 176, 2006.

[41] S. Kumar and J. Dudley, "Bioinformatics software for biologists in the genomic era," Bioinformatics, vol. 23, no. 14, pp. 1713-1717, 2007.

[42] M. Fourment and M. R. Gillings, "A comparison of common programming languages used in bioinformatics," BMC Bioinformatics, vol. 9, p. 82, 2008.

[43] M. J. Sullivan, N. K. Petty and S. A. Beatson, "Easyfig: a genome comparison visualizer," Bioinformatics, vol. 27, no. 7, pp. 1009-1010, 2011.

[44] A. Gevorgyan, M. E. Bushell, C. Avignone-Rossa and A. M. Kierzek, "SurreyFBA: a command line tool and graphics user interface for constraint-based modeling of genome-scale metabolic reaction networks," Bioinformatics, vol. 27, no. 3, pp. 433-434, 2011.

[45] A. Hoppe, S. Hoffmann, A. Gerasch, C. Gille and H. Holzhutter, "FASIMU: flexible software for flux-balance computation series in large metabolic networks," BMC Bioinformatics, vol. 12, pp. 28, 2011.

[46] B. Chapman and J. Chang, "Biopython: Python tools for computational biology," ACM SIGBIO Newsletter, 2000.

[47] D. Gront and A. Kolinski, "Bioshell- a package of tools for structural biology computations," Bioinformatics, vol. 22, no. 5, pp. 621-622, 2006.

[48] S. Steinbiss, G. Gremme, C. Scharfer, M. Mader and S. Kurtz, "AnnotationSketch: a genome annotation drawing library," Bioinformatics, vol. 25, no. 4, pp. 533-534, 2009.

[49] S. Sonnenburg, G. Ratsch, S. Hensche, C. Widmer, J. Behr, A. Zien, F. de Bona, A. Binder, C. Gehl and V. Franc, "The SHOGUN machine learning toolbox," The Journal of machine Learning Research archive, vol. 11, pp. 1799-1802, 2010.

[50] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak and A. Cardona, "Fiji: an open-source platform for biological-image analysis," Nature Methods, vol. 9, pp. 676-682, 2012.

[51] T. Hirai and N. Kiyosawa, "Developing a Practical Toxicogenomics Data Analysis System Utilizing Open-Source Software," Computational Toxicology Methods in Molecular Biology, vol. 930, pp. 357-374, 2013.

[52] R. S. Pressman, Software Engineering – A Practitioner's Approach, 7th ed., New York: McGraw-Hill, 2010, pp. 120-145.

[53] M. Huerta, "NIH Working Definition of Bioinformatics and Biotechnology", Accessible at: http://www.bisti.nih.gov/docs/CompuBioDef.pdf. Date accessed: 29th July, 2013

[54] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Date accessed: 29th July, 2013

[55] http://www.haskell.org/haskellwiki/Haskell. Date accessed: 29th July, 2013

[56] http://www.lua.org/about.html Date accessed: 29th July, 2013

[57] http://www.lua.org/authors.html Date accessed: 29th July, 2013

[58] http://www.perl.org/about.html Date accessed: 29th July, 2013

[59] http://www.scala-lang.org/node/241 Date accessed: 29th July, 2013

[60] R. Lerdorf and K. Tatroe, "Programming PHP", O' Reilly, pp. 5, 2002

[61] http://www.aminutewithbrendan.com/ Date accessed: 29th July, 2013.

[62] www.erlang.org/about.html Date accessed: 29 July, 2013

[63] http://en.wikipedia.org/wiki/R_%28programming_language%29 Date accessed: 29th July, 2013

[64] http://www.ruby-lang.org/en/about/ Date accessed: 29th July, 2013