INFORMATION SYSTEMS EDUCATION JOURNAL

Volume 17, No. 5
October 2019
ISSN: 1545-679X

In this issue:

iscap
Information Systems & Computing Academic Professionals

The **Information Systems Education Journal** (ISEDJ) is a double-blind peer-reviewed academic journal published by **ISCAP** (Information Systems and Computing Academic Professionals). Publishing frequency is six times per year. The first year of publication was 2003.

ISEDJ is published online (http://isedj.org). Our sister publication, the Proceedings of EDSIGCON (http://www.edsigcon.org) features all papers, panels, workshops, and presentations from the conference.

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the EDSIGCON conference. At that point papers are divided into award papers (top 15%), other journal papers (top 30%), unsettled papers, and non-journal papers. The unsettled papers are subjected to a second round of blind peer review to establish whether they will be accepted to the journal or not. Those papers that are deemed of sufficient quality are accepted for publication in the ISEDJ journal. Currently the target acceptance rate for the journal is under 40%.

Information Systems Education Journal is pleased to be listed in the Cabell's Directory of Publishing Opportunities in Educational Technology and Library Science, in both the electronic and printed editions. Questions should be addressed to the editor at editor@isedj.org or the publisher at publisher@isedj.org. Special thanks to members of AITP-EDSIG who perform the editorial and review processes for ISEDJ.

INFORMATION SYSTEMS EDUCATION JOURNAL

Editors

2019 ISEDJ Editorial Board

Software Concepts Emphasized in Introductory Programming Textbooks

Kirby McMaster, Ret.
kmcmaster@weber.edu

Brian Rague
brague@weber.edu

School of Computing
Weber State University
Ogden, UT 84408

Samuel Sambasivam
ssambasivam@apu.edu
Department of Engineering and Computer Science
Azusa Pacific University
Azusa, CA 91702

Stuart L. Wolthuis
stuart.wolthuis@byuh.edu
Faculty of Mathematics and Computing
Brigham Young University – Hawaii
Laie, HI 96762

Abstract

In this research study, we performed a *content analysis* of selected introductory programming textbooks for three languages to examine which software development concepts are emphasized in these books. Our goal was to determine which concepts are considered to be most representative of software development based on the topics emphasized by the textbook authors. We counted how often programming words appeared in samples of C++, Java, and Python books. We discovered which concepts are consistently supported for all three languages. We also noted those concepts that are favored by just one or two languages. Our summarized results lead to several conclusions that are relevant to the choice of a language for an introductory programming course.

Keywords: Java; C++; Python; programming; CS1; CS2; content analysis;

1. INTRODUCTION

Two current questions in Computer Science are: (1) What concepts should be taught in an introductory programming course, and (2) What language should be taught in the course? Debate on these questions has continued for decades, with no clear resolution in sight (Brilliant &

Wiseman, 1996; Siegfried, Chays, & Herbert, 2008; CC 2001; CSC 2013). The two questions are related, in that various programming languages historically have been designed based on differing conceptual frameworks.

The early years of computing saw advances in programming from machine language to assembly language to higher-level languages (such as FORTRAN and COBOL). The ability to give instructions to a computer in a language closer to the problem domain is one of the greatest inventions in computing. When employees learned how to program within the work environment, little attention was paid to sound programming concepts and practices because of the coding flexibility afforded by higher-level languages.

As the next generation of higher-level languages was developed (e.g. Algol and PL/I), designers took advantage of previous experience to consider a wider range of language options. During this period, a few languages were developed specifically for teaching programming (e.g. Basic and Pascal). The availability of languages designed for a variety of purposes encouraged teachers to present programming concepts beyond simple language-specific syntax features.

Languages were developed using different computational models, including functional languages (e.g. LISP, Haskell, Scheme) and logical languages (e.g. Prolog). In the relational database world, procedural languages (e.g. relational algebra) and non-procedural languages (e.g. SQL) were considered and implemented. Structured programming concepts were promoted as best practices to develop and maintain evolving complex business applications.

Object-oriented languages C++, Java and Python evolved from C or special purpose web and scripting languages. In the current academic environment, the above three object-oriented languages are among the most popular candidates for teaching introductory programming (Guo, 2014).

The decision about which programming paradigm to teach beginning students influences the choice of introductory language. The paramount question for an effective introductory programming course remains "What concepts to teach?", followed by "Which language best supports these concepts?". The increased demand for programming courses for liberal arts students has led to the development of what are termed CS0 courses (Sooriamurthi, 2010). The preferred programming language for a CS1 or CS2 course for Computer Science majors is often different from the language taught to non-majors (Hertz, 2010).

1.1 Purpose of this Research

Many research studies have been performed in recent years on which language is best for an introductory programming course (de Raadt, Watson, & Toleman, 2002). In an effort to contribute to this discussion, our research focuses on C++, Java, and Python, which are common CS1 and CS2 languages. Rather than argue the merits of these languages for teaching programming, we performed a *content analysis* (Krippendorff, 2012) of C++, Java, and Python textbooks to determine how well they support teaching fundamental programming concepts.

Our primary assumptions are that the framework of the author is reflected by the words used frequently in the textbook, and that the framework of interest is one that is appropriate for an introductory programming course. From the author's choice of words, we can judge how well the textbook will contribute to the generally recognized objectives of an introductory programming course.

2. METHODOLOGY

This section of the paper describes the methodology used to collect word frequency data from selected C++, Java, and Python textbooks. The words we are searching for represent important concepts for an introductory programming course. In this study, we did not start with an initial list of concepts. We recorded all words we found in the books, and eliminated those that did not relate to computer programming.

2.1 Sample of Textbooks

We collected a sample of 5 C++ books, 5 Java books, and 7 Python books. We included more Python books because they tended to be shorter. We wanted our sample to include popular books in all three languages. To reduce research costs, we chose textbooks that were available on the Internet and could be downloaded as PDF files. For example, we obtained C++ books by Prata (2005) and Lafore (2002), Java books by Schildt (2007) and Wu (2010), and Python books by Lutz (2011) and Zelle (2002). Overall, we obtained a fairly representative sample of books, but some were older editions.

2.2 Convert PDF Files to Text Files

To perform word searching and counting, Adobe Reader provides a menu option to convert the contents of a PDF file into a text file. We used

Adobe Reader to create a text file for each of the textbooks in our study.

We noticed that the text file versions of the books included many character strings containing digits, punctuation, and other non-alphabetic symbols. To simplify our counting of concept words, we wrote a Python program that (after changing C++ to CPP) removed all non-letter symbols except apostrophes, and replaced them with blank characters.

We included apostrophes to allow contractions (e.g. *don't*, *g'day*) to be counted as words. We considered allowing hyphens, but they were not used consistently by the authors (e.g. *floating-point* vs. *floating point*). Our Python program also converted all letters to lower-case.

Since we were searching for words that represent programming concepts, our Python program included a function to remove most of the words on Fry's list of 100 most frequent English words (UEN, 2015). A few of Fry's top 100 words can be interpreted in a programming context (e.g. *number*, *long*), which we retained. Instead, we modified the frequent word list to include some non-programming words from Fry's second 100 words (e.g. *only*, *most*). The total number of distinct words on our common word list was 110. By screening out common words, we shrunk the number of original words by more than 40%.

In the Python program, we also added a second function to convert many plural nouns and verbs to singular form. This reduced the number of distinct words further, since only the singular forms appeared in the generated text files. Our Python program provided a filtered set of text files consisting only of letters (and apostrophes), blanks, and substantially fewer words.

2.3 Word Groups for Concepts

A single programming concept can be expressed in more than one form. For example, a noun concept can be presented in singular or plural form (e.g. variable, variables). Verbs can also be written in singular or plural form, as well as with various tenses (e.g. solve, solves, solved, solving). Often, the same concept is described by both a noun and a verb (e.g. inheritance, inherit). In some cases, synonyms representing similar ideas can be used to represent a concept (e.g. record, structure). Some concepts are written not as a single word but as a sequence of words (e.g. structured programming).

Our goal was to count how often an author referred to a programming concept, but our counting software was designed to count individual words. For this reason, we defined a word group for each concept. In this study, a word group consists of a set of nouns and verbs that represent the same concept. We occasionally included synonyms in the same word group. To get a textbook count for a concept, we summed the frequencies for each of the words in the word group.

2.4 Word Counts and Word Rates

We used a program called TextSTAT (Huning, 2007) to obtain word counts for all words in our modified text files. With TextSTAT, a "Corpus" is created to hold a list of text files to examine simultaneously. We defined a corpus for each programming language: C++, Java, and Python. We linked each corpus to the transformed textfiles for the language. The total word counts for the three languages were nearly the same, having about 900,000 words for each language. We recorded the frequencies for each word and combined them into counts for word groups.

Although total word counts were close for each language, the sets of textfiles for each language do contain different total numbers of words. The Java books have a slightly greater total word count than the Python and C++ books. To standardize the counts, we converted each word count for a concept to a *word rate*. The rate we chose was "per 100,000 words". That is, we divided the concept word count by the total number of words in the set of textfiles for the language, and then multiplied by 100,000.

For example, the 5 C++ textfiles contained a total of 868,902 words. The word count for *object* in these files is 10,264. This count is rescaled to a word rate as shown below:

word rate = (10,264/868,902) * 100,000 = 1,181.3

This indicates that the *object* concept is mentioned 1,181.3 times per 100,000 words in the C++ files. Word rates were calculated for each concept in each language.

3. ANALYSIS OF DATA

The purpose of this research is to distinguish the frequency in which programming concepts appear in textbooks for C++, Java, and Python. For every concept, we counted the number of occurrences of each word group member in the textbooks. Prior to obtaining the results presented below, our samples of textbook words were filtered by replacing non-letter characters with blanks, removing common English words, and converting plural nouns and verbs to singular form.

3.1 Word Frequency Distributions

Selected statistics for the word frequency distribution for each language are shown in Table 1 (all tables located in the appendices). The samples consisted of 868,902 C++ words, 939,851 Java words, and 902,702 Python words. Most of these words are repeated multiple times in the textbooks. For example, in the C++ sample, the maximum frequency word is *function*, which appears 18,073 times. The maximum frequency words are *class* (18,009 times) in the Java books and *python* (10,946 times) in the Python books.

The TextSTAT program uses the term *word form* to refer to a specific word string, such as *object*, that represents one word. The total number of word forms for each language are given in Table 1. Note that the Java sample has the greatest number (26,587) of word forms and also the greatest number of word forms (11,120) that appear just once.

A surprisingly large number of words have a frequency of 1. Many of these words were not actual words, but consisted of several words concatenated together into a single string. We suspect that this anomaly is due to an imperfect conversion of PDF files into text files and the extensive use of variable names in programming texts.

When we checked word counts for each of the 5 Java books separately, we observed that one of the books had a noticeably larger number of words having a frequency of 1. Since we are looking for frequent words that represent programming concepts, words that appear only once should have little effect on the word counts of interest. However, a large number of unduplicated words can slightly bias the word rates calculated from word counts. Rather than remove this Java book having the large number of distinct words, we chose to ignore all words having a frequency of 1 when performing our word rate calculations. This reduced the total word counts for C++, Java, and Python to the values shown on the bottom line of Table 1.

3.2 Word Rate Distribution

Since our focus in this paper is on frequent words in the textbooks, we need to provide a criterion for determining if a word is frequent. The actual word frequencies range from 1 up to a maximum for each language. In C++ the maximum frequency is 18,073 for *function*. Because the total word counts differ for each language, we rescaled word frequencies into *word rates* as described above. Our criterion for defining frequent words involves setting a threshold word rate for frequent words.

Table 2 describes the distributions for C++, Java, and Python in terms of word rate intervals. If a frequent word were defined to be one with a word rate above 800 (words per 100,000 words), then there would be 10 + 7 + 3 = 20 frequent words (not all distinct). These 20 frequent words are not uniform across languages. For example, the word object has word rates above 800 for C++ and Java, but not Python.

In this paper, we chose to define a *frequent* word as one with a word rate above 250. This gives us a reasonable number of words to study for each language and across languages.

Not all frequent words are *programming* words. The words *example*, *chapter*, *using*, and *same* are frequent for all three languages, but we do not interpret these words as programming concepts.

3.3 Consistently Frequent Concepts

We further define a word to be *consistently frequent* when it is frequent for all three languages. The consistently frequent programming words, together with their word rates for C++, Java, and Python, are listed in Table 3. The words are ordered by decreasing average word rate. Because these words are used frequently by authors for all three languages, they represent a measure of agreement on important programming concepts irrespective of language.

The most frequent programming word across all three languages is *class*, which is a keyword for each language (shown in **bold**) and also the most frequent Java programming word. The most frequent C++ programming word is *function*. The most frequent Python word is *python*. However, *function* and *python* are not consistently frequent. Of the 16 programming words in Table 3, the C++ word rates are highest for 7 words, 3 words have the highest rates for Java, and the remaining 6 words have the highest rates for Python.

The OOP words *class* and *object* have very high rates for C++ and Java. This suggests a substantial emphasis on OOP in the Java and C++ books. For most Table 3 words, the rates for C++ and Java are fairly similar.

The frequent word *type* has a lower word rate for Python, where data types are dynamic and are not explicitly defined. The frequent word *list* has a higher word rate in Python because (variable size) lists are used in place of (fixed size) arrays. *File* has a higher Python word rate, perhaps due to the emphasis on multimedia in some Python books.

Six of the Table 3 words (*value*, *string*, *type*, *number*, *data*, *list*) refer to data characteristics and data structures. Three of the words (*program*, *code*, *line*) represent program segments. *Name* can refer to data (e.g. variables) or program components (e.g. functions).

3.4 Language Dependent Concepts

A number of programming words are frequent in one or two languages but not the third. For example, *function* is a frequent word in C++ and Python, but not in Java. We refer to these words as *language-dependent* programming concepts. These words reflect variation between languages about words that are important. Table 4 lists 18 programming words that have a word rate range (high minus low) above 275 and at least one word rate below 150.

For example, the word *reference* has word rates of 213.4 for C++, 85.3 for Java, and 209.7 for Python. This word is not included in Table 4 because the range of word rates is below 275. The purpose of this constraint is to highlight words with language rate disparities that are meaningful.

Excluding language names *cpp* (representing C++), *java*, and *python*, the Table 4 words include 3 C++ keywords, 4 Java keywords, and 1 Python keyword. Being a keyword can have some effect on word rates, especially if the word is used in sample code (e.g. *public* in C++ and Java). The importance of some keywords (like *class*) extends throughout programming. We now direct our attention to Table 4 words that are not keywords.

In C++ books, *method* is often replaced by the two-word term *member function* to designate functions that are part of a class. This can explain the high C++ rates for *function* and *member*. C++ uses a *compiler*, while Java and Python use a run-time environment or interpreter.

In C++ and Java, an *array* is more frequent than a (linked) list. *Pointers* are common in C and C++ for indirect addressing. *Declaration* of variables is required in C++ and Java, but not in Python. *Threads* and *events* are built into the Java language, but not C++.

If the language in a programming course switches from C++ to Java, then some of the frequent C++ concepts will not be well-supported in the Java books. Similarly, if the language switch is made from Java to Python, more programming concepts will be lost.

3.5 Less Frequent Concepts

We have presented programming words that have a word rate above 250 for at least one language. In this section, we examine selected non-frequent words representing concepts from object-oriented programming, structured programming, and software engineering. We might expect a majority of these concepts to be included in the content of an introductory programming course.

Object-oriented programming concepts have appeared often in Table 3 and Table 4. The OOP words *class* and *object* have high word rates in all three languages. In Table 5A, we show word rates for 3 defining characteristics of OOP.

Encapsulation and polymorphism have low word rates for all three languages. Inheritance does get some respect from C++ authors, with a word rate above 100. Maybe there is more discussion of class hierarchies in the C++ books. Encapsulation certainly should have higher rates, since it is a critical concept in modular programming and especially for classes. Polymorphism is difficult enough to pronounce much less explain in a textbook.

Table 5B lists 10 **structured programming** concepts. The first four Table 5B words--*sequence*, *selection*, *iteration*, and *recursion*--are the formal names for classic control structures. The next two words, *branch* and *loop*, are informal terms for *selection* and *iteration*, respectively. In all three languages, *loop* is much more frequent than *iteration*, but *branch* is not a popular substitute for *selection*.

The *block* concept has been central to structured programming since the days of Algol. Word rates for *block* are near 100 for C++ and Java, but smaller for Python. Python uses indentation instead of special symbols (e.g. braces) to designate the start and end of a *block* (or paragraph). The words *argument* and *parameter* are closely related. *Argument* is a frequent word for C++, but *parameter* has word rates below 200 for all three languages.

Procedure is a forgotten term in current language textbooks, perhaps due to the residual effects of the decision by C language designers to implement only functions. This design decision persists in C++, Java, and Python for various reasons.

The 16 **software engineering** concepts in Table 5C include project stages, activities, and byproducts that do not directly involve writing code. This list includes the frequent Java word *implementation* and the frequent C++ and

Python word *error*. These words were not included in Table 4 because their range of word rates was below 200. We might expect some of these concepts to receive less emphasis in an introductory programming course.

The first four words--*analysis*, *design*, *implementation*, and *maintenance*--describe the stages of the traditional software development life cycle (SDLC). *Implementation* (which includes writing code) has word rates between 102.6 and 252.7 for all three languages. *Design* has a word rate above 100 in the Java books. *Maintenance* and *quality* are almost an afterthought in all textbooks. Based on these books, don't hire an introductory programmer to do maintenance.

Additional observations about the software development word rates include the following. In software development, *requirements* and *specifications* are usually discussed together, in response to a *problem* request from a client. One formal SDLC document that is often prepared is a Software Requirements Specification (IEEE, 1998).

The word *documentation* does not appear often in C++ books (rate just above 25), but it does in Python books (rate almost 200). What does this say about the mindset of the authors of these textbooks? From our experience, many computer programmers do not like to document their work.

The word rates for *abstraction* are very low. The term may be too general to be used frequently in introductory programming books. This thought ignores arguments presented in the article "Is Abstraction the Key to Computing?" (Kramer, 2007).

The *model* (and *modeling*) concept has rates below 100, which appears low considering that most design work requires some form of modeling for both code and data. Modeling is the realization of abstraction. In introductory courses, much of the design work is usually provided by the instructor. The students focus on writing the programs.

Algorithm has a C++ word rate of almost 160, indicating that C++ books spend a reasonable amount of time explaining the nature of algorithms. Maybe this is one reason why C++ has a reputation for being "harder" than Java and Python.

The word rates for *test* are above 100, but the rates for *debug* are near 0. One possible explanation for this difference is that *test* does not imply that the programmer made a mistake, whereas debug suggests that something needs to be fixed. On the other hand, *error* has word rates that almost qualify it as a consistently frequent word. In commercial software development organizations, initial debugging is usually performed by the developers who write the code. Formal testing is more likely to be performed by specialized test groups, especially when a suite of tests must be re-run whenever the code is changed.

As a special note, if you want to teach students about *functional decomposition* or *data decomposition*, don't use one of these books. Word rates can't get much lower than 0.3.

4. SUMMARY AND CONCLUSIONS

The choice of programming language for an introductory Computer Science course influences the concepts that will be emphasized in the course. Discussion about which concepts to teach in a first course and what language best supports these concepts continues among faculty and professional organizations. This discussion has often led to the conclusion that no language is best for all situations (CSC, 2013). Our work attempts to contribute to this dialog by revealing which programming concepts are supported in textbooks for C++, Java, and Python.

We gathered a sample of textbooks that were restricted to those available in PDF format, converted the contents into text files, and then screened the files to remove or transform unnecessary material. We counted how often words that represent programming concepts appeared in the books, and then converted the frequencies into word rates. From the transformed data, we draw several conclusions.

A word is defined to be *frequent* for a language if its word rate is at least 250 per 100,000 words in the textbooks for that language. We found 16 programming words that are frequent for all three languages. Two of the words with the highest rates are *class* and *object*, which are central concepts for object-oriented programming. This list of concepts that are supported across languages is a good start for an introductory programming course.

We next searched for words that were frequent in one or two languages, but not all three. These words highlight differences between the languages. The word *function* is very frequent in C++ and Python, but not in Java. Java prefers the term *method.* Java considers all functions (and all code) to occur within a class. C++ uses the combined term *member function* for functions within a class, but C++ (and Python) allow functions to be defined outside of a class.

With its history from C, C++ provides explicit indirect addressing using *pointers*. Java makes indirect addressing implicit through the use of *references*. C++ and Java provide fixed size *arrays* as a common data structure. Python uses variable size *lists* (without mentioning the word *linked*). C++ and Java have a *character* data type, whereas characters in Python are represented as strings of length 1. Each language provides support for the above concepts, using possibly a different name, and sometimes involving a different underlying implementation (e.g. arrays vs. lists).

Among the other concepts, Java supports *threads* and *events* for real-time programming. C++ and Java, but not Python, require a *declaration* (*name* and *type*) for variables before they can be used.in a program. For words that are frequent in two languages, many of the word rates for C++ and Java are comparable. C++ and Java books seem to provide similar support for most of the frequent programming concepts. Python provides less support.

We also examined a selection of object-oriented programming, structured programming, and software development words that did not appear on our most frequent word lists. On a word-by-word basis, many of the comparative word rates are interesting, with several results standing out. Longer technical words (e.g. *polymorphism*, *iteration*, *requirement*, and *decomposition*) tended to have lower word rates, but there are exceptions (e.g. *selection* vs. *branch*). Word rate differences for *test*, *debug*, and *error* are hard to explain. Hopefully, the extremely low rates for *abstraction*, *maintenance*, and *quality* do not persist into more advanced programming textbooks.

Finally, both C++ and Java books provide reasonable support for most of the frequent programming concepts. Python provides less support. The ultimate choice of language for an introductory programming course must be based on considerations beyond textbook coverage of important concepts.

4.1 Future Research

Planned future research activities include:

1. Replicate this study with a larger, more representative sample of textbooks.

2. Examine variation in word rates between books within the same language.

3. Perform a similar study comparing textbooks for other candidate languages for an introductory programming course (e.g. PERL, Ruby, Javascript, Ada, Scheme).

5. REFERENCES

Brilliant, S. S., & Wiseman, T. R. (1996, March). The first programming paradigm and language dilemma. In *ACM SIGCSE Bulletin* (Vol. 28, No. 1, pp. 338-342). ACM.

Curricula, C. (2001). Computer Science. *Final Report, December*, *15*, 2001.

Joint, A. C. M. (2013). IEEE-CS Task Force on Computing Curricula. *Computer science curricula*.

De Raadt, M., Watson, R., & Toleman, M. (2002). Language trends in introductory programming courses. In *Proceedings of the 2002 Informing Science+ Information Technology Education Joint Conference (InSITE 2002)* (pp. 229-337). Informing Science Institute.

Guo, P. (2014). Python is now the most popular introductory teaching language at top us universities. *BLOG@ CACM, July*, 47.

Hertz, M. (2010, March). What do CS1 and CS2 mean?: investigating differences in the early courses. In *Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 199-203). ACM.

Huning, M. (2007). TextSTAT 2.7 User's Guide. *TextSTAT, created by Gena Bennett*.

IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998). IEEE recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers.

Kramer, J. (2007). Is abstraction the key to computing?. *Communications of the ACM*, *50*(4), 36-42.

Krippendorff, K. (2012). Content Analysis: An Introduction to Its Methodology, 3rd Ed. SAGE Publications.

Lafore, R. (2002). Object-Oriented Programming in C++ (4th ed). Sams Publishing.

Lutz, M. (2011). Programming Python (4th ed). O'Reilly Media.

Prata, S. (2005). C++ Primer Plus (5th ed). Sams Publishing.

Schildt, H. (2007). Java The Complete Reference, 7th Ed. McGraw-Hill.

Siegfried, R. M., Chays, D., & Herbert, K. (2008, July). Will there ever be consensus on cs1?. In *FECS* (pp. 18-23).

Sooriamurthi, R. (2010). The essence of object orientation for CS0: concepts without code. *Journal of Computing Sciences in Colleges*, *25*(3), 67-74.

UEN (2015). High Frequency Words--Fry Instant Words. *Utah Education Network.* Retrieved Feb 21, 2017 from http://www.uen.org/

UEN (2018). k-2educator/word_lists.shtml *Utah Education Network.* Retrieved Feb 21, 2017 from http://www.uen.org/

Wu, C. T. (2010). An Introduction to Object-Oriented Programming with Java (5th ed). McGraw-Hill.

Zelle, J. (2002). Python Programming: An Introduction to Computer Science. Wartburg College Printing Services.

Appendices

Table 1: Word Frequency Distribution Summary

Statistic	**C++**	**Java**	**Python**
Textbooks	5	5	7
Authors	6	8	10
Total Words	868,902	939,851	902,702
Max Count	18,073 *function*	18,009 *class*	10,946 *python*
Min Count	1	1	1
Word Forms	17,328	26,587	21,644
Forms: count>1	11,716	15,467	14,620
Forms: count=1	5,612	11,120	7,024
PctForms:count=1	32.4%	41.8%	32.5%
*Words:count>1	863,286	928,749	895,678

* Used to calculate word rates

Table 2: Word Forms by Word Rate

Word Rate	**C++**	**Java**	**Python**
800.0+	10	7	3
400.0 - 799.9	18	15	19
200.0 - 399.9	49	40	43
100.0 - 199.9	97	121	113
50.0 - 99.9	190	218	228
25.0 - 49.9	326	325	372
* Words: count>1	863,286	928,749	895,678

* Used to calculate word rates

Table 3: Consistently Frequent Programming Concepts

(Rate > 250 for all 3 languages)

Rates for keywords are shown in **bold**

	Concept	C++ Rate	Java Rate	Python Rate	Mean
1	class	**1,929.0**	**1,939.1**	**641.9**	**1,503.33**
2	object	1,188.9	1,163.7	629.2	994.0
3	value	1,019.1	835.8	675.0	843.3
4	program	890.1	913.1	688.4	830.8
5	string	855.0	857.1	529.2	747.1
6	type	861.7	782.7	370.7	671.7
7	file	571.3	551.4	890.4	671.0
8	line	450.8	498.9	611.8	520.5
9	number	597.7	543.6	415.9	519.1
10	name	493.1	481.7	580.0	518.3
11	call	552.1	486.3	494.6	511.0
12	data	523.5	412.0	394.3	443.3
13	list	302.1	358.1	568.2	409.5
14	code	374.7	310.5	433.0	372.7
15	element	443.9	254.6	288.6	329.0
16	input	267.0	251.6	296.0	271.5

Table 4: Language-Dependent Concepts
at least 1 rate < 150, and range > 275
Rates for keywords are shown in **bold**

	Concept	C++ Rate	Java Rate	Python Rate	Range
1	function	2,093.5	58.4	696.8	2,035.1
2	python	3.5	0.1	1,222.1	1,222.0
3	cpp	1,192.2	0.0	0.2	1,192.2
4	java	11.8	1,072.5	61.0	1,060.7
5	member	719.8	119.7	24.5	695.4
6	operator	**776.6**	146.9	133.2	643.4
7	array	641.4	486.7	34.4	607.0
8	public	**197.7**	**621.0**	38.9	582.1
9	pointer	551.0	17.3	11.3	539.8
10	module	10.5	5.7	461.2	455.5
11	thread	0.8	414.9	210.0	414.1
12	constructor	395.8	268.8	49.1	346.7
13	event	8.2	336.7	132.7	328.5
14	declaration	333.0	213.2	19.1	313.9
15	static	**163.1**	**329.4**	17.9	311.5
16	compiler	300.6	72.5	8.0	292.6
17	import	1.3	**185.3**	**291.5**	290.2
18	interface	64.9	**341.0**	161.7	276.1

Table 5A: Object-Oriented Programming Concepts

	OOP Concepts	C++ Rate	Java Rate	Python Rate	Mean
1	encapsulation	6.3	5.4	5.9	5.9
2	inheritance	129.4	45.1	29.9	68.1
3	polymorphism	28.7	17.7	6.1	17.5

Table 5B: Structured Programming Concepts

	StructProg Concepts	C++ Rate	Java Rate	Python Rate	Mean
1	sequence	98.0	97.7	121.8	105.8
2	selection	38.3	45.8	44.4	42.8
3	iteration	22.5	18.8	18.8	20.0
4	recursion	24.6	30.9	17.1	24.2
5	branch	3.2	6.9	4.1	4.8
6	loop	215.8	174.0	165.9	185.2
7	block	95.6	100.7	48.3	81.5
8	argument	436.8	181.8	184.6	267.7
9	parameter	154.2	179.2	116.7	150.0
10	procedure	3.6	6.4	4.8	4.9

Table 5C: Software Engineering Concepts

	Software Dev Concepts	C++ Rate	Java Rate	Python Rate	Mean
1	analysis	10.8	11.4	16.7	13.0
2	design	74.6	112.1	45.7	77.5
3	implementation	147.3	252.7	102.6	167.5
4	maintenance	3.2	2.3	3.7	3.1
5	problem	128.2	123.9	94.3	115.5
6	requirement	24.1	11.5	14.6	16.7
7	specification	89.9	147.7	92.3	110.0
8	abstraction	7.4	6.2	4.8	6.2
9	model	41.7	80.8	50.8	57.8
10	algorithm	159.5	77.3	68.2	101.7
11	decomposition	0.1	0.3	0.1	0.2
12	test	122.1	136.1	122.1	155.8
13	debug	12.2	5.1	8.2	8.5
14	error	242.9	198.8	214.5	218.7
15	documentation	26.6	89.6	195.9	104.0
16	quality	4.3	2.9	3.1	3.4