Coverage of CS1 Programming Concepts in C++ and Java Textbooks

Kirby McMaster
Weber State University
Ogden, UT 84408
kmcmaster@weber.edu

Brian Rague
Weber State University
Odgen, UT 84408
brague@weber.edu

Samuel Sambasivam
Azusa Pacific University
Azusa, CA 91702
ssambasivam@apu.edu

Stuart Wolthuis
Brigham Young University-Hawaii
Laie, HI 96762
stuartlw@byuh.edu

Abstract--In this research study, we performed a content analysis of programming concepts in C++ and Java textbooks. Our goal was to determine which language has better textbook support for teaching a CS1 programming course to Computer Science majors. We counted how often our list of concept words appeared in samples of C++ and Java books. Our summarized results lead to several conclusions that relate to the choice of a programming language for a CS1 course.

Keywords--Java, C++, programming concepts, textbooks, content analysis.

1. INTRODUCTION

In the earliest years of computing, the choice of a first language for programmers was often decided by the work environment. Assembly language for a specific hardware system was the usual situation, but programming in a higher-level language such as FORTRAN became common over time. When Computer Science (CS) programs at universities began to develop, the choice of an introductory programming language was determined primarily by the curriculum designers, with an emphasis on the pedagogical value of the language rather than its practicality in developing real-world applications. As might be expected in the academic world, there was a diversity of opinion on what the first language should be [1].

Some early courses in CS emphasized broader computing frameworks rather than the subtleties of programming syntax [2]. The decision about which programming paradigm to teach beginning students strongly influences the choice of introductory language. The paramount question regarding the delivery of an effective introductory CS course is "What to teach?", followed immediately by "Which language best supports the concepts to be taught?".

In the 1970s and 1980s, Pascal became the language taught most often in introductory programming courses. Eventually, many schools moved to C for practical reasons, since graduates rarely used Pascal in their employment [3]. As the benefits of object-oriented programming became evident, the first language evolved to C++ and later to Java [4]. The most recent Computer Science Curriculum Guidelines (CS2013) [5] published by ACM/IEEE state that "...advances in the field have led to an even more diverse set of approaches in introductory courses."

In recent years, the increased demand for programming courses for liberal arts students has led to the development of what are termed CS0 courses aimed for non-CS majors, with Python a popular CS0 language [6]. The preferred programming language for a CS0 course is often different from the language taught to CS majors in CS1 courses. In this paper, we restrict our attention to languages for a CS1 course.

A. Purpose of this Research

Many research studies have been performed in recent years on which language is best for an introductory programming course. In an effort to contribute to this discussion, our research focuses on two languages, C++ and Java, which are currently the most common CS1 languages. Rather than evaluate the suitability of these languages for CS1 courses, we performed a content analysis [7] of C++ and Java textbooks to determine how well they cover important programming concepts such as *class* and *algorithm*.

An instructor in a programming course usually chooses a textbook to guide how she/he will organize and present the material. In an effort to customize the course to the specific needs of the students and curriculum, instructors will typically select a subset of topics from the text and often reorder the material to improve continuity. Our investigation considers textbook content in its entirety, since our primary research assumption is that the framework of the author is reflected by the words used most frequently throughout the complete text. The framework we are examining is one that is appropriate for a CS1 programming course. From the

author's choice of words, we can judge how well the textbook will support the main objectives of the programming course.

II. METHODOLOGY

This section of the paper describes the methodology used to collect word frequency data from selected C++ and Java textbooks. The words we examine represent important concepts for an introductory programming course.

A. Programming Concepts

We created a list of important programming concepts using several sources. We started with an initial list of programming terms taken from quizzes and exams we have given to CS1 students to assess their understanding of course topics. This list includes a number of structured programming and object-oriented programming (OOP) words.

We created a second list consisting of programming concepts mentioned in the Programming Fundamentals (PF) section of the Computing Curricula 2001 Computer Science (CC2001) Final Report [8]. We created a third list of concepts based on the Software Development Fundamentals (SDF) section of the Computer Science Curricula 2013 (CS2013) Final Report.

From the above sources, we formed a combined list that grew to 101 programming concepts. In our final list, we attempted to avoid keywords from specific languages, such as *float* and *while*. However, some keywords such as *class* and *operator* were difficult to omit because they have broad meaning throughout Computer Science.

B. Word Groups for Concepts

A single programming concept can be expressed by an author 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*, *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 across specification and implementation domains are used to represent a concept (e.g. *record*, *structure*). Some concepts are written not as a single word but as a word phrase (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* (lexeme) for each concept. A word group is comprised of a set of words that represent the same concept. To get a textbook count for a concept, we added the frequencies for each of the words in the word group. Note that context was not considered when calculating word group (concept) counts.

C. Sample of Textbooks

We collected samples of 10 C++ textbooks and 10 Java textbooks. We wanted our samples to include popular books in both languages. Due to a minimal budget (i.e. no budget), we chose textbooks that were available on the Internet and could be downloaded as PDF files. We obtained a reasonably diverse sample of books, but some were older editions. We later observed that the C++ and Java books in our samples were on average approximately the same size. The mean length of the C++ books was 223,663 words, whereas the mean for the Java books was 222,953 words.

D. Convert PDF files to Text Files

Textbooks in PDF file format are not convenient for performing repeated word searching and counting. Fortunately, Adobe Reader has a menu choice to convert the contents of a PDF file to a text file. We used Adobe Reader to create a text file for each of the 20 textbooks in our study.

We noticed that the text file versions of the books included many character strings that contained digits, punctuation, and other non-alphabetic symbols. To simplify our counting of concept words, we wrote a short Python program that removed all non-letter symbols and replaced them with blank characters (after changing C++ to CPP). This program also converted all letters to lower-case. We used this program to obtain a filtered set of 20 text files which consisted of only letters and blanks. Note that none of our concept list word groups contains a numeric or special character.

E. Perform Word Counts

We used a popular program called TextSTAT [9] to obtain word counts for all words on our programming concepts list. With TextSTAT, you first define a "Corpus", which will hold a list of text files to examine. We defined a corpus for each textbook and linked the corpus to the single transformed textfile representing the textbook.

To perform a word search, a separate TextSTAT screen allows the user to specify search options. At first we used the option to include all words, with the words and frequencies presented in alphabetical order. We then went through the concept list (also in alphabetical order) and recorded the frequencies for each word group. This was the most labor-intensive part of our methodology. Occasionally, we would enter a short string (e.g. *iterat*) to search for all words that contained the string (e.g. *iterate*, *iteration*, *iterator*).

III. ANALYSIS OF DATA

The number of programming concepts on our final list was 101. Alphabetically, the concepts ranged from *abstraction* to *variable*. As mentioned in the methodology section, each concept was represented by a group of one or more words. For example, the word group for the OOP concept *object* contained two words: *object* (singular) and *objects* (plural).

For every concept, we counted the number of occurrences of each word group member in the C++ or Java textbook. As an example, in the Java book by Schildt [10], the word *object* appears 1674 times, and the word *objects* appears 380 times. The total word count for the concept is 2054.

A. Convert Word Counts to Word Rates

Each textbook contains a different total number of words, so the word counts for concepts are not comparable across books. Larger books tend to have greater word counts. 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 book and multiplied by 100,000.

For example, Schildt's book mentioned above contains a total of 325,991 words. The word count for the *object* concept is 2054. This count is rescaled to a word rate as shown below:

word rate = (2054/325,991) * 100,000 = 630.1

This reflects that the *object* concept is mentioned 630.1 times per 100,000 words in Schildt's book. Word rates were calculated for each concept in each book.

B. Calculate Trimmed Means

After concept word rates were obtained in all C++ and Java textbooks, averages were calculated separately for the C++ and Java values. Because the word rates for concepts (C++ or Java) sometimes varied widely from book to book, we calculated trimmed means to diminish the effect of outliers. To provide a simple removal of outliers, our trimmed means include only the middle 8 out of 10 word rates. The largest and smallest word rates are dropped.

For example, word rates for the *object* concept in all 10 Java textbooks are:

522.4 561.7 630.1 705.7 843.3 684.9
703.5 767.2 863.5 488.4

Removing the highest rate (863.5) and lowest rate (488.4), the trimmed mean for *object* in the Java books is 677.3. Two trimmed means were calculated for each concept, one for C++ and the other for Java.

C. Distributions of Trimmed Means

Each set of books (C++ and Java) provided a sample of 101 trimmed means, measuring word rates for the 101 programming concepts. A statistical description of the C++ and Java word rate distributions is summarized in Table 1. The complete list of concepts, along with trimmed mean word rates for C++ and Java, are presented in the Appendix at the end of the paper.

Most of the statistics are nearly the same for the C++ distribution and the Java distribution. The central tendency mean for C++ is higher than for Java, but the median is lower. The dispersion measure (IQR) for C++ is slightly smaller. Overall, concept words appear more often in the C++ books.

TABLE I: Distributions of Trimmed Means

Statistic	C++	Java
Sample N	101	101
Minimum	0.24	0.67
Centile 25	26.28	21.28
Median	53.65	55.13
Centile 75	139.61	137.59
Maximum	940.03	968.16
IQR	113.33	116.31
Mean	122.04	111.51

For the C++ distribution, the concept having the *maximum* word rate is *function*, and the *minimum* word rate is for *decomposition*. For Java, the *maximum* word rate is for the concept *class*, while the *minimum* word rate is for *quality*. The C++ *median* word rate is for the middle concept *assignment*, which has rank 51. For Java, the median concept is *linked*.

The *mean* of the C++ word rates is more than twice the size of the C++ median. This indicates that the distribution is positively *skewed*, due to the presence of several high word rates (outliers). The mean of the Java word rates is slightly more than twice the median value, indicating another positively *skewed* distribution.

The variation of scores in a distribution is usually described by the *standard deviation*. However, this statistic is inflated when outliers are present. A more stable measure of variation is the interquartile range (IQR) [11], which is the difference between the 75th centile value and the 25th centile value. For C++, the 75th centile concept is *error*, and the 25th centile concept is *debug*. The corresponding concepts for Java are *list* (75th) and *style* (25th).

The word rates for programming concepts tend to be higher in the C++ books than in the Java books. Overall, 64 of the 101 concepts have a higher word rate in the C++ books. The remaining 37 concepts occur more often in the Java books. Additional details and comparisons of these two word rate distributions are presented in the following sections.

D. Most-Frequent Concepts

The fifteen programming concepts with the highest word rates for C++ and Java are listed in Table 2. Eleven of the concepts (73%) appear on both lists, but in different ranked positions. This demonstrates reasonable agreement by authors on which concepts are *most important* in both languages. Four concepts are on the C++ list only, and four others are confined to the Java list. The concepts that are not on both lists are shown in **bold**.

Among the Java concepts, the top three--*class*, *method*, and *object*--describe features of object-oriented programming (OOP). *Class* and *object*, but not *method*, also on the C++ list, but with slightly lower word rates. C++ prefers the older

term *function* over *method*, perhaps due to its roots in C. Six of the C++ concepts--*variable*, *type*, *value*, *number*, *string*, and *array*--describe data types and data structures. The Java list also contains all six of these concepts.

TABLE II: Most-Frequent Concepts
(Differences in **bold**)

C++ Concept	Rate	Java Concept	Rate
function	940.0	class	968.2
class	901.1	**method**	958.7
object	565.8	object	677.3
program	516.6	value	475.9
value	508.0	program	456.3
operation	475.8	string	411.4
type	408.3	type	398.7
number	354.5	variable	292.7
string	351.3	array	265.7
array	346.1	**system**	263.7
variable	337.0	number	261.9
file	323.4	**statement**	221.4
pointer	291.9	code	214.0
data	267.9	file	208.9
code	232.4	**thread**	179.4

E. Least-Frequent Concepts

The twenty programming concepts with the lowest word rates for C++ and Java are listed in Table 3. Fourteen of the concepts (70%) appear on both lists, but in different ranked positions. This shows moderate agreement by C++ and Java authors on concepts perceived to be less important in their programming frameworks. Concepts that appear on only one list are shown in **bold**.

The concepts that appear on both least-frequent lists include a few surprises. Some of these concepts are considered relevant by many programming instructors. Certainly *abstraction* is a key programming topic. Of the three pillars of OOP (*encapsulation*, *inheritance*, and *polymorphism*), *encapsulation* and *polymorphism*, but not *inheritance*, are on both lists. The *signature* concept, relevant to *polymorphism*, is rarely mentioned by either set of authors.

Function and *procedure* were once distinct concepts in modular programming. Perhaps due to compromises made in the design of the C language (and perpetuated in C++ and Java), the *procedure* term has been replaced with "void" functions. *Quality* and *maintainable* software seem to be held in minimal regard by both C++ and Java textbook authors.

The *pointer* concept has low word rates for Java, but not for C++. The substitute (but not equivalent) term *reference* appears fairly often in both sets of books. *Reserved* word is less popular than *keyword* in both languages, although their usage does not match technical differences between the concepts. Finally, almost no books contain the word *decomposition*, which is one of the bottom two words on both least-frequent lists. This concept embodies a core strategy in modular programming.

TABLE III: Least-Frequent Concepts
(Differences in **bold**)

C++ Concept	Rate	Java Concept	Rate
record	15.5	literal	14.8
documentation	14.6	**efficiency**	13.4
module	13.8	**identifier**	13.2
polymorphism	13.4	signature	12.9
pattern	13.3	**scope**	12.9
relation	12.7	encapsulation	12.8
component	12.6	**dynamic**	9.9
boolean	9.1	**debug**	9.0
maintainable	8.1	reserved	8.9
signature	7.1	record	8.0
literal	7.0	polymorphism	7.3
branch	6.4	maintainable	7.2
encapsulation	5.9	**pointer**	6.4
procedure	5.8	abstraction	6.3
abstraction	5.6	relation	5.2
event	4.9	procedure	4.7
reserved	4.7	branch	3.4
quality	1.9	module	1.5
thread	0.4	decomposition	0.7
decomposition	0.2	quality	0.7

F. Middle-Frequency Concepts

We have presented word rates for the top 15 and bottom 20 programming concepts, and now turn our attention to the remaining concepts with middle-level usage rates. This list is too long to fully discuss with a single table. Instead, we present 15 *structured programming* concepts in Table 4 and 12 *software engineering* concepts in Table 5.

TABLE IV: Middle-Frequency Concepts
Structured Programming

Concept	C++ Rate	Java Rate
algorithm	56.4	35.4
argument	170.2	118.7
assignment	106.7	55.1
block	50.7	54.9
condition	50.4	53.6
control	45.2	61.6
expression	124.9	108.3
iteration	96.4	26.3
local	42.2	37.2
logic	27.6	23.8
loop	131.0	112.0
nested	15.7	22.6
parameter	97.3	95.6
recursion	24.8	29.2
sequence	41.5	50.4

Structured programming topics have been an essential part of programming language courses for almost 50 years,

perhaps encouraged by the popularity of Dijkstra's [12] paper on the harmful effects of "go to" statements. Table 4 gives C++ and Java word rates for selected structured programming concepts (listed in alphabetical order) with middle-level word rates.

In Table 4, the concepts *argument*, *loop*, and *expression* have word rates above 100 for both languages. C++ books also have a word rate above 100 for *assignment*. *Nested* and *recursion* have low rates for both C++ and Java. The largest differences in word rates are for *iteration*, *argument*, and *assignment*, with C++ having the higher values. We observe that C++ has a higher word rate for *algorithm* than does Java, but the rate for each language should be higher.

Software engineering (SE) concepts receive less coverage in CS1 courses. These concepts become more important in later programming courses. In Table 5, the SE word rates for C++ range from 14.6 (for *documentation*) to 121.5 (for *user*). The SE word rates in Java books range from 21.3 (for *style*) to 129.1 (again for *user*).

TABLE V: Middle-Frequency Concepts
Software Engineering

Concept	C++ Rate	Java Rate
problem	102.7	68.9
solution	36.1	35.2
requirement	53.1	31.4
specification	39.5	54.7
model	26.8	25.3
algorithm	56.4	35.4
design	45.8	53.3
test	66.2	86.0
correctness	21.4	22.5
style	28.9	21.3
documentation	14.6	40.2
user	121.5	129.1

Concepts on the Table 5 list include *problem* and *solution*, reflecting the problem-solving focus in software engineering. The words *requirement*, *specification*, *model*, *algorithm*, *design*, and *document* are life cycle development activities. *Style* is a consideration to ensure source code is readable and maintainable. In our opinion, the low word rates for *model* and *style* are disappointing, even in an introductory programming context..

As Table 5 indicates, most of these SE concepts appear with relatively low word rates in both the C++ and Java textbooks. Six of the concepts appear more often in the C++ books, while the other six are more frequent in the Java books. The word rates for five of the concepts are nearly the same for both languages. There is no obvious single criterion for determining which language favors which SE concepts.

G. Word Rate Correlation

Beyond examining the C++ and Java word rate distributions separately, we now consider the joint distribution of the two rates. If the focus on key introductory concepts is consistent across the examined textbooks, we would expect to find a positive relationship between the C++ and Java word rates. For most programming concepts, a higher word rate in the C++ books should suggest a higher word rate in the Java books, and vice versa.

To measure the degree of linearity in the relationship, we calculated the Pearson correlation coefficient. The correlation value we obtained for our 101 pairs of scores was 0.650, which is positive but not near 1.0. We do not claim that the relationship is linear, but it should be monotonic.

An often-used statistic for monotonic relationships is the Spearman rank-order correlation [13]. Our result for the Spearman statistic was 0.727, which describes a moderately strong *increasing* relationship between C++ and Java word *ranks*. A scatter diagram of the word rate pairs, converted to ranks (where 1 is the "highest" rank and 101 is the "lowest" rank), is displayed in Fig. 1.

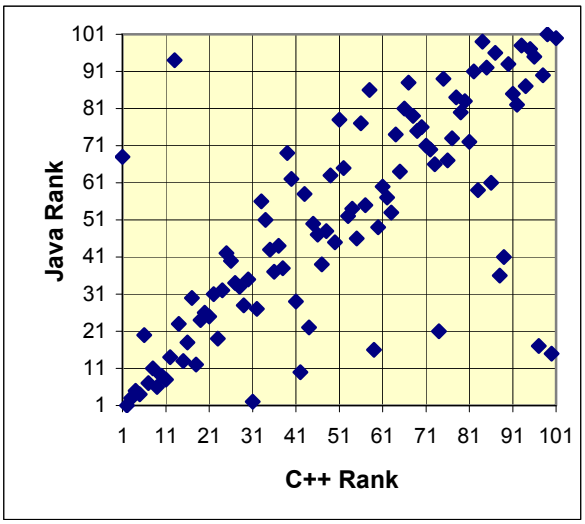

Fig. 1: C++ vs. Java Concept Ranks

In this figure, we can see that most of the pairs of ranks fall approximately along a line that runs from pair (1,1) to pair (101,101). Above the implied line, two obvious exceptions are the pairs (13,94) for *pointer* and (1,68) for *function*. In these pairs, the C++ rank is much closer to 1 than the Java rank. Below the line, the two most noticeable outliers are (100,15) for *thread* and (97,17) for *event*. These concepts have a much higher Java rank.

A list of concepts having the largest C++ vs. Java rank differences is summarized in Table 6. The choice of how large the difference should be to consider the concept as a two-dimensional outlier is subjective. In this table, we include all pairs in which the rank difference is 29 or larger in magnitude. A negative difference occurs when C++ has a better rank (lower rank number). A positive difference favors Java. Note that all but two of the concepts in Table 6 have a better Java rank.

TABLE VI: Largest Differences in Ranks
("Highest" rank is 1)

Concept	C++ Rank	Java Rank	Diff
pointer	13	94	-81
function	1	68	-67
method	31	2	29
system	42	10	32
interface	59	16	43
boolean	89	41	48
component	88	36	52
instance	74	21	53
event	97	17	80
thread	100	15	85

We stated in Table 2 that *function* is the concept having the highest word rate among C++ books. The *pointer* concept has the 13th highest C++ rate. This table indicates that both of these popular C++ concepts appear much less often in Java books. Four secondary OOP concepts--*method*, *interface*, *component*, and *instance*--are favored by Java books. Real-time *events* and *threads* are common Java features, but not C++. Also, *system* is a keyword for Java but not for C++

C++ books often use the term *member function* instead of *method* to describe a function defined inside a class. C++ does not require functions to be defined within a class. Java books prefer *method* because, in Java, all functions appear inside a class. This view holds even when a class is not used to create objects.

IV. SUMMARY AND CONCLUSIONS

The choice of programming language for introductory Computer Science courses is a strong indicator of the concepts emphasized during course instruction. Ongoing discussion about what to teach and which language best supports learning objectives for introductory programming courses continues unabated among instructors, administrators, and accreditation organizations. A definitive "best practices" approach in this area remains unresolved. Our work contributes to this debate by correlating core programming concepts with specific textbooks that promote either Java or C++ as the course language.

The primary purpose of this study was to compare how C++ and Java textbooks provide coverage of important introductory programming topics. Although each instructor usually selects a subset of topics from an individual textbook when developing a course, we performed content analysis of the entire text to achieve a more accurate and comprehensive examination of topic coverage. We developed a list of 101 programming concepts. We collected a sample of 10 C++ books and 10 Java books. Then we counted how often words that represent the programming concepts appeared in the books. From transformed word rate (per 100,000 words) data, we draw the following conclusions.

First, words that describe our 101 programming concepts have slightly higher word rates in the C++ books in our study. The word rate distribution for C++ has a mean of 122.04; for Java, the mean is 111.51.

Second, there is surprising agreement between the programming concepts mentioned most often in the C++ and Java books. Eleven of the top 15 C++ concepts are also included in the top 15 Java concepts. Highly-used concepts for both languages include *class* and *object*, each reflecting an emphasis on OOP.

Third, there is also agreement on which concepts are rarely mentioned in both sets of books. Fourteen of the bottom 20 C++ concepts are also in the list of 20 least-used Java concepts. Common neglected concepts include *encapsulation* and *polymorphism* for OOP, plus SE concepts *quality* and *maintainable*. Alas, *abstraction* is on both least-used lists.

Fourth, several concepts appear on only one of the most-frequent word lists for C++ and Java. The top 15 C++-only concepts include *function*, *operation*, *pointer*, and *data*. Top 15 Java-only concepts include *method*, *system*, *thread*, and *statement*.

Fifth, a fairly strong increasing relationship exists between concept ranks for C++ and Java, as measured by a rank-order correlation of 0.727. There are several exceptions to this relationship. *Function* and *pointer* have much higher C++ ranks. Ranks for *thread* and *event* are much higher for Java.

Sixth, based on the collected data, C++ and Java textbooks devote substantial time on practical concepts that describe how to write logically correct code. Discussion of structured programming and general software engineering concepts that deal with how to *think* like a programmer and write clear and maintainable code receives less attention. These learning goals may be less important in an introductory programming course, but they become a major focus as students progress toward a Computer Science degree.

Finally, both C++ and Java books provide reasonable support for most of the programming concepts on our list. The choice of Java or C++ (or other language) for a CS1 course should be based on considerations beyond textbook support for specific concepts. Whatever language and textbook are chosen, instructors must provide additional material to achieve their desired course objectives.

Future research activities include:

(1) Compare Java and Python textbooks to determine how well they support concepts for CS0 courses.

(2) Provide empirical support to adapt and improve our list of concepts for various programming courses.

This second activity is not a simple task, as is apparent from previous research by Hertz [14] and Tew [15].

V. APPENDIX

Concept Word Rate Trimmed Means for C++ and Java

	Concept	C++ Rate	Java Rate			Concept	C++ Rate	Java Rate
1	abstraction	5.6	6.3		51	literal	7.0	14.8
2	algorithm	56.4	35.4		52	local	42.2	37.2
3	argument	170.2	118.7		53	logic/logical	27.6	23.8
4	array	346.1	265.7		54	loop/looping	131.0	112.0
5	assignment/assign	106.7	55.1		55	maintain/maintainable	8.1	7.2
6	block	50.7	54.9		56	method	124.3	958.7
7	boolean	9.1	84.2		57	model/modeling	26.8	25.3
8	branch/branching	6.4	3.4		58	module	13.8	1.5
9	case	125.3	128.6		59	nest/nested	15.7	22.6
10	character	189.9	132.2		60	number/numeric	354.5	261.9
11	class	901.1	968.2		61	object	565.8	677.3
12	code	232.4	214.0		62	operation/operator	475.8	144.3
13	component	12.6	99.6		63	output	134.2	110.9
14	condition/conditional	50.4	53.6		64	parameter	97.3	95.6
15	constant	75.7	61.5		65	pattern	13.3	37.1
16	constructor	206.4	138.0		66	pointer	291.9	6.4
17	control	45.2	61.6		67	polymorphism	13.4	7.3
18	correct/correctness	21.4	22.5		68	problem	102.7	68.9
19	data	267.9	139.8		69	procedure	5.8	4.7
20	debug/debugging	26.3	9.0		70	process/processing	65.9	64.7
21	declaration/declare	137.5	85.1		71	program	516.6	456.3
22	decomposition/decompose	0.2	0.7		72	quality	1.9	0.7
23	definition/define	210.8	120.9		73	queue	27.7	21.1
24	design	45.8	53.3		74	record	15.5	8.0
25	development/develop	26.6	30.0		75	recursion/recursive	24.8	29.2
26	documentation/document	14.6	40.2		76	reference	104.6	96.0
27	dynamic/dynamically	33.3	9.9		77	relation/relational	12.7	5.2
28	efficient/efficiency	16.3	13.4		78	requirement/require	53.1	31.4
29	encapsulation/encapsulate	5.9	12.8		79	reserved	4.7	8.9
30	error	139.6	80.8		80	scope	45.3	12.9
31	event	4.9	155.2		81	selection	16.8	15.1
32	exception	88.6	123.4		82	sequence	41.5	50.4
33	expression	124.9	108.3		83	set	157.6	144.6
34	file	323.4	208.9		84	signature	7.1	12.9
35	floating	33.7	15.0		85	software	36.3	21.6
36	function	940.0	28.8		86	solution/solve/solving	36.1	35.2
37	identifier	18.7	13.2		87	specification/specify	39.5	54.7
38	implementation/implement	77.7	141.4		88	stack	109.3	52.3
39	index	67.9	65.5		89	statement	207.4	221.4
40	information	56.3	67.8		90	stream	48.8	67.5
41	inheritance/inherit	79.5	43.0		91	string	351.3	411.4
42	input	104.9	80.1		92	structure	92.4	35.5
43	instance	26.3	141.7		93	style	28.9	21.3
44	integer	156.8	117.5		94	system	80.4	263.7
45	interface	45.3	159.9		95	test/testing	66.2	86.0
46	iteration/iterator/iterate	96.4	26.3		96	thread	0.4	179.4
47	keyword	46.8	20.2		97	tree	32.9	18.7
48	line	220.6	149.8		98	type	408.3	398.7
49	link/linked	53.7	20.0		99	user	121.5	129.1
50	list	171.3	137.6		100	value	508.0	475.9
					101	variable	337.0	292.7

VI. REFERENCES

[1] Siegfried, Robert M., Chays, David, and Herbert, Katherine G., “Will There Ever be Consensus on CS1?” In Proceedings of FECS. 2008, 18-23 http://home.adelphi.edu/~siegfried/Consensus.pdf

[2] Brilliant, S. S., and Wiseman, T., “The First Programming Paradigm and Language Dilemma”, ACM SIGCSE Bulletin Vol. 28, No. 1 (1996), p. 338-342.

[3] de Raadt, Michael, Watson, Richard, and Toleman, Mark, “Language Trends in Introductory Programming Courses,” I nSITE, June 2002, http://proceedings.informingscience.org/IS2002Proceedings/papers/deRaa136Langu.pdf

[4] Lister, Raymond, E. A. Research perspectives on the objects-early debate. In ITiCSE proceedings (2006), pp. 146--165. (p 4, 5).

[5] Computer Science Curricula 2013, Joint Task Force on Computing Curricula, Association of Computing Machinery, IEEE Computer Society, 2013.

[6] Guo, Philip, "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities." Communications of the ACM, Blogs, 2014.

[7] Krippendorff, Klaus H., Content Analysis: An Introduction to Its Methodology, 3rd Ed. SAGE Publications, 2012.

[8] Computing Curricula 2001 Computer Science - Final Report, Joint Task Force on Computing Curricula, Association of Computing Machinery, IEEE Computer Society, 2001.

[9] Huning, M, TextSTAT 2.7 User’s Guide. TextSTAT, created by Gena Bennett, 2007.

[10] Shildt, Herbert, Java The Complete Reference, 7th Ed. McGraw-Hill, 2007.

[11] Upton, Graham, and Cook, Ian, Understanding Statistics. Oxford University Press, 1996, p.55.

[12] Dijkstra, E. W., "Go to Statements Considered Harmful." Communications of the ACM 11 (3), 147-148.

[13] Maritz, J. S., Distribution-Free Statistical Methods (2nd ed). Chapman and Hall, 1995.

[14] Hertz, Matthew, "What do 'CS1' and 'CS2' Mean? Investigating Differences in the Early Courses." SIGCSE Proceesings, Milwaukee, 2010.

[15] Tew, Allison Elliott, & Guzdial, M., Developing a Validated Assessment of Fundamental CS1 Concepts, SIGCSE Proceedings, Milwaukee, 2010.

C++ Sample Textbooks

1. Davis, Stephen Randy, C++ For Dummies (5th ed). Wiley Publishing, 2004.
2. Deitel, Paul, and Harvey Deitel, C++ How to Program (8th ed). Prentice Hall, 2012.
3. Eckel, Bruce, Thinking in C++ (Volume 1, 2nd ed). Planet PDF, 2000.
4. Enzust, Alan, and Paul Enzust, An Introduction to Design Patterns in C++ with Qt4. Prentice Hall, 2007.
5. Halterman, Richard L., Fundamentals of C++ Programming (Draft). Richard L. Halterman, 2015.
6. Lafore, Robert, Object-Oriented Programming in C++ (4th ed). Sams Publishing, 2002.
7. Liberty, Jesse, Teach Yourself C++ in 21 Days (2.d ed). Sams Publishing, 1998.
8. Oualline, Steve, Practical C++ Programming. O'Reilly, 1995.
9. Prata, Steven, C++ Primer Plus (5th ed). Sams Publishing, 2005.
10. Roberts, Eric S., and Julie Zelenski, Programming Abstractions in C++. Eric Roberts, 2003.

Java Sample Textbooks

1. Arnold, Ken, James Gosling, and David Holmes, THE Java Programming Language (4th ed). Addison Wesley Professional, 2005.
2. Deitel, Harvey, and Paul Deitel, Java How to Program (4th ed). Prentice Hall, 2002.
3. Downey, Allen B., Think Java: How to Think Like a Computer Scientist. Allen Downey, 2012.
4. Eck, David J., Introduction to Programming Using Java (Version 6.0.3). Hobart and William College, 2014 (PDF version of on-line book).
5. Lemay, Laura, and Charles L. Perkins, Teach Yourself JAVA in 21 Days. Sams.net Publishing, 1996.
6. Roberts, Eric. S., The Art and Science of Java (Preliminary Draft). Stanford University, 2006.
7. Schildt, Herbert, Java: The Complete Reference (7th ed). McGraw-Hill, 2007.
8. Sierra, Kathy, and Bert Bates, Head First Java (2nd ed). O'Reilly.
9. Stein, Lynn Andrea, Interactive Programming in Java. Lynn Andrea Stein, 1999.
10. Wu, C. Thomas, An Introduction to Object-Oriented Programming with Java (5th ed). McGraw-Hill, 2010.