Will There Ever Be Consensus on CS1?

Robert M. Siegfried[1], David Chays[1], and Katherine G. Herbert[2]
[1]Department of Mathematics and Computer Science, Adelphi University, Garden City, NY 11530, USA
{siegfrie, chays}@adelphi.edu
[2]Department of Computer Science, Montclair State University, Montclair, NJ 07043, USA
herbertk@mail.montclair.edu

Abstract *- The choice of programming language, the approach by which students are taught and the software tools made available to students have been controversial issues in many ways. While there once was a consensus of some sort within the computer science education community, it is much more difficult to find common ground among those of us who teach introductory programming courses. The literature is explored and answers sought to the question of which language is optimum in teaching novice programmers, as well as the approach that ought to be used. Finally, the question of whether a consensus can be reached is addressed.*

Keywords: Introductory Programming, Programming Language, Java, Objects Early Approach.

1 Introduction

The choice of a programming language to use in a CS1 course and the software tools that are made available to students is a potentially thorny subject that has been the subject of much debate. There is also a long history of introductory programming courses changing from one programming language to another and from one compiler to another when a newer language or a newer compiler (or development environment) was perceived as better. The difficulty that students had in deciphering error messages with industry-standard compilers led to the development of WATFOR, the FORTRAN compiler that was the first widely-available compiler for student use [1]. When PL/I came out, this was also criticized, in part because of its ties to IBM and in part because of the complexity of the language [2]. The difficulties that students encountered led to the development of PL/C, another compiler designed specifically for student use [3]. The development of the Pascal programming language and its compiler by Wirth [4] led to a period of time when most students learned to program in a programming language that was generally considered ideal for teaching, albeit not for commercial use.

Yet not even Pascal escaped criticism. Both Haberman and Kernighan [5, 6] wrote about its unsuitability for larger-scale programming projects. And while many colleges and universities tried switching to C, this was not generally considered successful. Johnson considered it too large and complex for use in an introductory course [7]. Brilliant and Wiseman recognized the appeal of an industry-standard language such as C, but did not consider this enough of an advantage to outweigh its drawbacks [8].

Since the decline of Pascal, many colleges and universities have changed their introductory programming language to C++ and then to Java. The Advanced Placement exam followed a similar trend, by changing its language of instruction from Pascal to C++ and then to Java.

There has not been a comprehensive survey to show what languages are currently in use in the United States but there have been some studies that provide insight into what is being used in other countries. For many years, Richard Reid of Michigan State University surveyed over 200 colleges and universities in the United States, usually on an annual basis, and in recent years, Java has become the most commonly used first programming language, either by itself or in combination with another language [9].

The most comprehensive study was done by de Raadt et. al. and found that Java and C++ are the most common programming languages taught in an introductory programming course in Australia and that concerns about the job market were a primary consideration in choosing a programming language [10]. They also discovered that these courses were as likely to teach the imperative paradigm as the object-oriented paradigm. While there has been no similar study in the United States that demonstrates this, it is commonly assumed that Java is the most common programming language, at least in part, because employers want to hire people who know it, consequently students want to learn it, and finally colleges and universities feel pressure to teach it.

Yet there was some dissatisfaction in the results of first year programming classes. Reges [11] reported that there was a decline in student enrollment at the University of Washington, something that has been seen across the country. Blum and Cortina cite a decline of 6% in Advanced Placement Computer Science courses, despite an increase in enrollment in all other disciplines in which there are Advanced Placement exams [12].

With so many introductory courses in Java being offered, one must ask the question "Should we teach Java to beginning programmers?" And if we should be teaching Java in our introductory courses, what approach should we be using?

2 Should We Teach in Java?

Java is not necessarily an ideal language for teaching introductory programming. It is not a simple language, given the number of reserved words that it contains, its more complex syntax and the complexity that inevitably comes with any object-oriented language. Hadjerroult notes that Java is not sufficiently simple for novice programmers to learn quickly [13]. Reges notes the Computer Science Department at the University of Washington faced several problems, including a lack of basic programming skills that were reported by instructors of upper-level undergraduate courses [11]. It has been suggested that the objects early approach may be at least partially responsible for this trend. This attitude may be summed up by Elliot Koffman [14], who posted a message to the SIGCSE mailing list that said "I fear that we have reinvented the 'new math' syndrome and many of us are unaware of it."

Yet despite these misgivings, a large number of colleges and universities still teach introductory programming in Java and many of them use an objects early approach. The twenty-third Reid list showed that sixty of the 120 schools that responded to the survey that year use Java, with a few additional schools using Java in combination with another programming language [15]. Many of the textbooks covering introductory programming in Java use the objects early approach. Decker and Hirschfield rebutted most of the common reasons why instructors are reluctant to use the objects early approach [16]. While these views are supported by their experiences teaching introductory programming, they present no hard data supporting this position.

2.1 Is Java the optimal language?

The Sixth Reid List (from 1992) showed that most of the colleges and universities surveyed taught beginning programming in Pascal [17]. In the intervening years there has been no one language that the computer science education community has agreed upon to the extent that they agreed on Pascal in its heyday. There are many reasons why Java has not become the dominant introductory programming language to the extent that Pascal did a generation ago. Java is a single-paradigm language; it is impossible to program in Java without making repeated, explicit use of objects. This is problematic for many instructors who are uncomfortable with an objects early approach [18]. Additionally, in the first few years after Java's release, its lack of easy console input and its lack of a student-friendly development environment made some reluctant to introduce it; this is no longer the case, given the introduction of the Scanner class, which simplifies input reading and parsing, and the development of Integrated Development Environments (IDEs) such as BlueJ and DrJava. The difficulty that beginning students encountered using an earlier IDE, Code Warrior, was one of the complaints when Adelphi originally shifted the introductory programming course to Java [19].

2.2 Objects early or objects late?

There has been a large amount of controversy within the past few years as to whether objects should be introduced toward the beginning of the course or later in the course, typically in the second half of the semester. Bruce points out that the trend toward introducing objects earlier has grown over time as more introductory Java textbooks introduce object-oriented concepts earlier than in the past [20]. Bruce is personally in favor of an objects early approach built around pedagogic tools similar to what he and his colleagues developed at Williams College, although he recognizes that there are some problems associated with this approach, including the lack of textbooks based on it.

Bruce is not the only one pointing out problems with the objects early approach. Buck and Stucki claim that the early introduction of software design, which is typically a factor in the objects early approach, is harmful because it exposes students to issues for which they do not have the necessary cognitive skills [21]. They prefer to use an "inside/out approach", where they introduce objects by using the primitives of the programming language and simple library calls. This approach teaches students many of the advantages of objects early without the complications that many people associate with the objects early approach.

Reges is one of several people who expressed concerns about the amount of material that CS1 instructors are expected to cover in one semester. McConnell and Burhans found that textbooks have grown significantly larger over the past 40 years and devote less space to selection and repetition statements, as well as variables and arrays. Much of this resembles the "Procedures Early" approach which became popular in the late 1980s and early 1990s. The "Procedures Early" approach was also controversial, with many people, such as Pattis, pointing out its shortcomings [22].

2.3 What other languages might be suitable?

Most other programming languages used in introductory programming courses are criticized in one way or another. C++ is considered inadequate, because of its descent from C and because it is too easy to avoid the use of objects when programming in it. Brilliant and Wiseman raise the issue of whether one should or should not cover object-oriented programming when teaching C++ [23].

While King offered no direct complaints about C++, he offered many reasons why Java was superior as a first programming language [24].

Rosener advocates the use of Visual Basic in a first programming course because students pick it up easily [25] and there are many schools where Visual Basic is used in some introductory courses, but its lack of pointers (or references) and the inability to design classes of objects in it make it unsuitable in an upper level course. There is also the issue of the extent to which an instructor wishes to delve into the issue of event-driven programming, which is an important part of the visual programming paradigm.

Grandell et. al. advocate the use of Python as an introductory programming language, based on their experience teaching it to high school students [26]. Mannila and deRaadt, in their extensive analysis of various programming languages, conclude that Python is the most suitable language for beginners. While Python has benefits as an easy-to-use prototyping language, its dynamic type system and its use of exact indentation to delimit blocks can inadvertently lead students to programming errors that may be difficult to detect.

Reas and Fry developed the programming language Processing for use in graphics arts [27]. The syntax is similar to Java, but it is an interpreted language and requires little of the syntactic framework commonly found in a Java program. It has the advantage of being able to motivate students who wish to create images on the screen, but it lacks many of the features expected in a general purpose language.

Advocates of the TeachScheme! approach claim that the simpler syntax makes it ideal for teaching beginners to design algorithms and provides a good stepping stone to Java and other languages. But there is no published data based on objective studies using control populations to support this. The only study involving the use of Scheme as a segue into the study of another programming language was done by Wick and Stevenson [28], who found that students who learned Scheme prior to learning Prolog did not have greater proficiency in Prolog compared to the students who studied just Prolog. The only discernible benefit that they could identify was that these students learned a second programming language in another paradigm.

3 What Features Make A Language Ideal For Beginners?

The most detailed study of the design of programming language for programming novices and the cognitive issues that must be addressed was done by Linda McIver [29]. McIver wrote that a programming language should be useful, usable, human-centered (i.e., designed around the programmer's needs), task- or domain- oriented, consistent (both with what programmers already know and within the language's own constructs) and robust. However, she found that this was usually not the case; programming languages usually are easy to translate, hard to use (especially by beginners), hardware-centered and oriented around a paradigm (occasionally leading to awkward features in a language).

Consequently, McIver recognized the importance of understanding how the novice perceives text written in a particular programming language if we are to evaluate the suitability of any programming language in teaching novices how to program. Green [30] identified thirteen cognitive dimensions of notation. These are, in reality, properties that notations or language may possess that will either make it easier or harder for novices to learn them. McIver found that six of these are of particular importance in evaluating programming languages for use by novice programmers:

1. "Closeness of mapping" addresses how well the notation represents the domain for which it is intended, e. g., if we are trying to describe arithmetic, how closely does our notation resemble arithmetic?

2. To be "consistent", similar semantics should be expressed in similar syntax. Therefore, an if..elseif..else construction would be considered more consistent than a switch statement.

3. "Diffuseness" refers to the verbosity of the language. COBOL would be an example of a diffuse notation.

4. "Error-prone" constructions are those that are more likely to lead to errors, or perhaps even encourage them. The use of separate pairs of brackets for different dimensions of an array might be considered error-prone.

5. "Hard mental operations" would require the programmer to prefer potentially difficult tasks in writing a program, e.g., entering all numeric constants in an unusual number base.

6. "Role expressiveness" refers to the ability of a reader to infer the usage of a feature just from its structure.

McIver examined several languages, including Java, which failed to meet the optimal case for cognitive dimensions. These results appear in Table I, accompanied by the cognitive dimensions of Pascal, C++ and Scheme, languages that have been used or are currently used in first year programming courses. An examination of this table leads to some interesting conclusions. Firstly, Pascal

TABLE I
COGNITIVE DIMENSIONS OF JAVA COMPARED WITH OTHER LANGUAGES (TAKEN FROM [25])

Dimension	Optimal	Java	Pascal	C++	Scheme
Closeness of Mapping	High	Low	Medium	Low to Medium	Low
Consistency	High	Low to Medium	Low to Medium	Low	Medium to High
Diffuseness	Medium to High	Low	Low to Medium	Low	Low
Error-proneness	Low	Medium to High	Low to Medium	High	Medium to High
Hard Mental Operations	Low	Medium to High	Low to Medium	High	Medium to High
Role Expressiveness	High	Low	Medium to High	Low to Medium	Low

remains the closest to optimum of the four languages shown. Secondly, Java is a small improvement over C++. And lastly, Scheme scores more favorably than the other languages.

Students in a CS1 class do not need to know an entire programming language; however, they do need to be able to use the basic skills associated with programming in Java and to be able to adapt to new programming challenges that will arise. The following list contains the programming skills and language features that can help focus students learning to program in an object-oriented language such as Java and help them to adapt to the next level of programming:

- Programming/problem-solving
- Adaptive programming skills
- Basic input and output
- Primitive data types
- Basic encapsulation
- Methods
- Conditional statements
- Iterative statements (loops)
- Working with collections (arrays)
- File input and output
- The anatomy of a class
- The String class
- Working with the API
- Documentation
- Discussion of advanced topics in Java

While many of these items would appear to be language-specific, a CS1 course using another language would replace a few of these items with analogous constructs.

While the reader may expect to see many of these topics, there are some that might not be expected. While it is generally agreed upon that input/output, primitive data types, encapsulation and methods, conditional and iterative statements and documentation should be taught in a CS1 class, the other topics are frequently omitted. Two of the most important skills in the above list are programming/ problem-solving skills and adaptive programming skills, which every student must learn. Frequently, a CS1 course in Java contains so much material that these two most important skills can be pushed aside. It is easy to assume that students with an understanding of mathematics that is at least on the level of algebra can adapt these problem solving skills to computing; however there is a body of evidence that strongly suggests that this is not the case. Biggs' structure of the observed learning outcome (SOLO) taxonomy discusses such issues [31, 32]. It illustrates how students will not learn a topic unless the instructor aligns course objectives to learning outcomes and demonstrates to students how to perform the necessary tasks. Since programming is a vital task for any technologist-student, it becomes necessary to give him/her the skills that allow the student to survive beyond the test.

4 Some Conclusions

Java is not an ideal language for beginners. McIver points out that Java's modular structure and requirement that every data item and method be part of a class mandate a certain minimum size for every program [25], no matter how simple it may be:

```
public class MyFirst {
  public static void main(String[] args)   {
    System.out.println
          ("This is my first Java program.");
  }
}
```

This also applies to the definition of constants, which can require as many as four reserved words:

```
public static final double PI = 3.14;
```

While a subset of Java can minimize the problems that novice programmers must face, it is very difficult to create a subset that addresses all these concerns.

The popularity of Java is partially due to the fact that it is used for many real-world applications, particularly web-related applications. Yet there are many features that make it difficult for novice programmers. However, most of the other languages that are presented as alternatives for use in a CS1 course have some aspects that make them undesirable to some faction within the computer science education community.

Arguably, the discipline may need a new teaching language that will offer the benefits that the computer science education community found in Pascal over thirty-

five years ago. But at the present, it seems that there will be great difficulty finding that consensus.

5 References

[1] P. W. Shantz, R. A. German, J. G. Mitchell, R. S. K. Shirley, and C. R. Zarnke, "WATFOR: The University of Waterloo FORTRAN IV", Communications of the ACM Vol. 10, No. 1 (January 1967), p. 41-44.

[2] R. C. Holt, "Teaching the Fatal Disease or Introductory Computer Programming Using PL/I", ACM SIGPLAN Notices, Vol.8 No. 5 (May 1973), p. 8-23.

[3] R. W. Conway and T. R. Wilcox, "Design and Implementation of a Diagnostic Compiler for PL/I", Communications of the ACM Vol. 16 No. 3 (March 1973), p. 169-179.

[4] N. Wirth, "The Programming Language Pascal", Acta Informatica Vol. 1 (1971), p. 35-63.

[5] A. N. Habermann, "Critical Comments on the Programming Language Pascal", Acta Informatica 3 (1973), p. 47-57.

[6] B. W. Kernighan, "Why Pascal is Not My Favorite Programming Language", Computing Science Technical Report No. 100, AT&T Bell Laboratories (April 1981).

[7] L. F. Johnson, "C In The First Course Considered Harmful", Communications of the ACM, v. 38. n. 5 (May 1995), pp. 99-101.

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

[9] D. Reid. The Reid List of the First Course Language for Computer Science Majors. http://www.csee.wvu.edu/ vanscoy/reid.htm, 2001.

[10] M. de Raadt, R. Watson and M. Toleman, "Introductory Programming: What's Happening Today and Will There be Any Students to Teach Tomorrow" Australian Computer Science Communications 26(5): 277 - 284.

[11] S. Reges, "Back to Basics in CS1 and CS2", ACM SIGCSE Bulletin Vol. 38 No. 1 (March 2006), p. 293-297.

[12] AP Program Summary Report 2005, available at http://apcentral.collegboard.com as quoted by Lenore Blum and Thomas J. Cortina, "CS4HS: An outreach Program for High School CS Teachers", ACM SIGCSE Bulletin, Vol. 39, Issue 1 (March 2007), p. 19-23.

[13] S. Hadjerroult, "Java as First Programming Language: A Critical Evaluation", ACM SIGCSE Bulletin Vol. 30, No. 2 (June 1998), pp.43-47.

[14] O. Astrachan, K. Bruce, E. Koffman, M. Kölling and S. Reges, "Resolved: Objects Early Has Failed", Proceedings of the thirty-sixth SIGCSE technical symposium on Computer Science Education", 2005, p.451-452. , Quoted in Reges Stuart, "Back to Basics in CS1 and CS2", ACM SIGCSE Bulletin Vol. 38 No. 1 (March 2006), p. 293-297.

[15] R. Reid, Reid First Course List 23 (February 1, 2002). Last retrieved April 11, 2008 from http://www.csee.wvu. edu/ ~vanscoy/ REID23.HTM

[16] R. Decker and S. Hirschfield, "The Top 10 Reasons Why Object Oriented Programming Can't Be Taught in CS1", ACM SIGCSE Bulletin Vol. 26, No.1 (March 1994), p. 51-55.

[17] R. Reid, Reid First Course List 6 (April 19, 1992). Last retrieved April 11, 2008 from http://www.csee.wvu.edu/ ~vanscoy/ REID06.HTM

[18] L. Mannila and M. de Raadt, "An Objective Comparison of Languages for Teaching Introductory Programming", Proceedings of the Sixth Koli Calling Conference on Computer Science Education, 2006, available at http://www.it.uu.se/research/group/upcerg/Publications/ proceedingsKoliCalling2006/research2.pdf

[19] S. Bloch, Adelphi University, private communication, April 1999.

[20] K. B. Bruce, "Controversy on how to teach CS1: a discussion on the SIGCSE-members mailing list", ACM SIGCSE Bulletin, Vol. 36, Issue 4, December 2004, p. 29-34.

[21] D. Buck and D. J. Stucki, "Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development", p. 75-79.

[22] R. E. Pattis, "The 'Procedures Early' Approach In CS1: A Heresy", ACM SIGCSE Bulletin, Vol. 25, Issue 1, March 1993, p. 122-126

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

[24] K. N. King, "The Case for Java as a First Language", Proceedings of the 35th Annual Southeast ACM Conference (April 1997), p. 124-131.

[25] W. Rosener, “Programming Language Consider-ations for Introductory Computer Courses: Consider Visual Basic”, Seventh International Integration of Academic and Technical Education Conference (Beaver Creek, Colorado, June 28-July 1, 1999), available at http://arapaho.nsuok.edu/~rosener/papers/vb/vb.html

[26] L. Grandell, M. Peltomäki, R.-J. Back and T. Salakoski, “Why Complicate Things? Introducing Programming In High School Using Python”, Proceedings of the 8th Australian conference on Computing education, Volume 52 (Hobart, Australia), 2006, p. 71 – 80.

[27] C. Reas and B. Fry, “Processing: A Learning Environment For Creating Interactive Web Graphics”, International Conference on Computer Graphics and Interactive Techniques (San Diego, CA, 2003), p. 1.

[28] M. R. Wick and D. E. Stevenson, “On Using Scheme to Introduce Prolog”, ACM SIGCSE Bulletin Vol. 38. No. 1, March 2006, p. 41-45.

[29] L. McIver, “Syntactic and Semantic Issues in Introductory Programming Education”, Ph.D. Dissertation, Monash University, 2001.

[30] T. R. G. Green, “Cognitive Dimensions of Notation and other Information Artefacts”, People and Computers V: Proceedings of Human Computer Interaction 1989 (HCI’89), Cambridge University Press, 1989.

[31] J. Biggs, “Assessing for Learning: Some dimensions underlying new approaches to educational assessment”. The Alberta Journal of Educational Research Vol. 41, No. 1 (1995), p. 1-17.

[32] J. B. Biggs and K. F. Collis, Evaluating the Quality of Learning – the SOLO Taxonomy. New York: Academic Press. xii + 245 pp.