Recommendations of the AP® Computer Science Ad Hoc Committee

The AP Computer Science Ad Hoc Committee

Owen Astrachan	Duke University
Robert (Corky) Cartwright	Rice University
Gail Chapman	The College Board
David Gries	University of Georgia
Cary Horstmann	San Jose State University
Richard Kick	Hinsdale Central High School
Frances Trees	Westfield Senior High School
Henry Walker	Grinnell College
Ursula Wolz	The College of New Jersey

Summary

In 1999, the College Board appointed an AP Computer Science Ad Hoc Committee to address the future of AP Computer Science (APCS). Their charge was to make recommendations to the AP Computer Science Development Committee on the future direction of AP Computer Science, including recommendations for changes in the APCS curriculum, and to explore alternative delivery languages. These recommendations were to be based, in part, on a survey of college departments conducted in summer 1999 to see what language was likely to be used in the introductory computer science courses in the next three to four years.

After four meetings, and after considering the surveys conducted in 1999 and 2000 as well as survey results published in the Journal of Computer Science Education (Stephenson and West 14(1), 14(2), April 2000), the Ad Hoc Committee recommends a change from an object-based to an object-oriented approach to programming. The fundamental difference is an inclusion of inheritance. As a result, we further recommend that the language supporting the curriculum be switched from C++ to Java, beginning with the 2003-2004 academic year.

There are philosophical and pragmatic reasons for recommending these changes. The philosophical reasons are based on three principles that should be considered in choosing a language to express the concepts found in an introductory computer science curriculum: safety, simplicity, and support for object-orientation. Pragmatic reasons for changing are based primarily on the fact that colleges and universities are adopting new approaches based on the three principles. These new approaches are more natural to express in Java than in C++ for reasons described in the appendix that follows this Executive Summary.

Surveys conducted by ETS and others indicate that an emphasis on object orientation is accepted practice in introductory computer science courses at the college level. Our observations are in agreement with the recently published results of Stephenson and West (collected over a year ago). There has been a significant shift in attitude during the past year. The Stephenson and West survey indicated that a shift was taking place toward object-oriented programming and the use of the Java language which supports that paradigm more effectively than C++. Recent survey results confirm that this shift is continuing.

Following are the issues that the committee considered in the process of reaching its decision. A full discussion of each is provided in an appendix of the report. We also present a discussion of alternatives for the examination format and provide an outline of materials needed for teacher professional development.

Reasons for Changing

- **Safety** is important at all levels of software design but especially when learning to program. Although there is some confusion about what a safe programming language is, we are using safety to mean: *Any attempt to misinterpret data is caught at compile time or generates a well-specified error at run time.* Java is a safe programming language. The Java compiler catches many inadvertent errors and the Java run time environment ensures that any attempt to misinterpet data will generate a well-defined exception.

- **A simple language** should be used to teach programming in an introductory computer science course. A simple language will be grammatically as well as semantically uncomplicated. It will be easy to learn, because the notation will express programming concepts in a straightforward way. The appendix includes a comparison of some language features that illustrate how simplicity helped guide the APCS Ad Hoc Committee's recommendation to switch to Java.

- **Object Orientation** involving encapsulation, inheritance, polymorphism, and abstraction, is an important approach in programming and program design. It is widely accepted and used in industry and is growing in popularity in the first and second college-level programming courses.

Format of the Course

It has been suggested that the APCS examinations should be language independent or allow for multiple languages. The APCS Ad Hoc Committee recommends that the exam and course rely on a single language for the expression of the concepts included in the course curriculum.

Teacher Professional Development

Appropriate and sufficient training of AP Computer Science teachers is a necessity for the success of a smooth transition to Java. In order for this to take place, basic materials and documents need to be made available in an efficient manner. These are outlined in the report and a time line is provided in the appendix with recommendations of workshops and materials that will need to be prepared.

The APCS Java Subset

The purpose of the language subset is to enable the designers of the APCS exam to formulate questions relating to the APCS syllabus. The APCS subset is not intended as a prescription for a computer science course. We expect courses to cover language features that go well beyond this subset. A draft of the subset is available online (www.cs.duke.edu/csed/adhoc) and will be finalized in spring 2001 after a three-month period of public comment.

APCS Course Content and Examination Alternatives

Various alternative approaches and formats for AP Computer Science have been suggested. Most of the suggestions can be categorized in one of the following ways.

- APCS should be language independent.
- APCS should allow multiple languages.
- APCS should utilize one language for the A examination and a second language for the AB examination.
- Versions of the exam should be offered in both C++ and Java – at least for a year or two, to facilitate the transition between languages.
- APCS should focus on discrete mathematics rather than programming concepts and practice.

Each of these options has been discussed by the AP Computer Science Ad Hoc Committee. The results are summarized below.

Language-independent APCS Examinations

It has been suggested that a language-independent examination might encourage a focus on ideas rather than language details. This approach has merit, but at least two fundamental obstacles consistently stand in the way of the adoption of this approach for APCS.

1. The need to provide a mechanism for students to receive credit and placement in colleges – the goal of the AP Program; and
2. The need for unambiguous stating of test questions and solutions.

In computer science, as in other disciplines, a language provides a mechanism for communication. In describing algorithms and solving problems, any such language must be clear and unambiguous. Virtually all colleges and universities introduce a programming language in their introductory CS1-CS2 courses. Some time ago, pseudocode was promoted at SIGCSE Symposia and elsewhere as an appropriate alternative, but this approach is not widely discussed today. Colleges introduce a specific language so that students get direct feedback from compilers and run-time environments, and so that precision of thinking can be sharpened and evaluated through testing. Some have argued that this hands-on perspective also follows from a general orientation toward practicality in the United States – U.S. culture places great value on practical objects that perform tasks and solve problems.

With this almost universal use of specific languages in CS1-CS2, it seems unlikely that colleges and universities would give credit and/or placement for an examination that did not require precise communication through a recognized programming language.

One might envision some exam questions that could be language independent, but it is much harder to envision a complete exam (including student writing of algorithms and solution designs) without specification of a language to be used.

Writing of questions and fair grading of solutions to those questions requires the use of a clear and unambiguous language. In order for pseudocode to be used in this context, it would require a precise specification, making it no better than a programming language. At a local level, an instructor might provide informal conventions concerning pseudocode, but a national examination requires careful specification.

Multiple Languages for APCS Examinations

Because various programming languages are used by colleges and universities in their introductory courses, the suggestion has been made to offer the APCS course in more than one language. Difficulties with this multi-language approach generally fall into three categories.

1. Providing guidance to high school teachers regarding curricula.
2. The logistics of exam development.
3. The acceptance of APCS grades by colleges and universities.

With a multi-language option, each high school teacher would have to select a language for use in the course or choose to teach more than one language in the same course. Very few college faculty introduce multiple languages in the same introductory course, as different syntax tends to confuse students and as switching languages involves a high overhead. College faculty probably would not consider teaching two languages in parallel to different groups in the same class.

Selecting one or more languages for APCS would require the high school teacher to anticipate what colleges students might be attending, and what languages might be used at those schools. Finally, such an approach might force teachers to bow to local pressures in choosing the language.

The AP Exam development process in all subjects includes significant testing and analysis to ensure that exams pass validity and reliability measures and that grades on multiple versions of the examination are comparable. Multiple-choice questions are administered first to college audiences, and the results are analyzed extensively. Different versions of exams (e.g., exams from successive years) contain some questions in common, so student performance on different exams can be compared. Further, great care is taken in setting final grades so that reported grades from year to year will be comparable. All of this work provides colleges and universities with a dependable measure of performance from year to year that can be used reliably in granting credit and/or placement. (More detailed information on the validity and reliability of AP examinations is available at www.collegeboard.org/ap/techman.)

While this pretesting of questions, analysis of scores, and assurance of grade comparability is vital to the AP Program, such work is time consuming. Currently, the Development Committee must work at least three years in advance in its question development and pretesting to meet essential deadlines.

In considering multiple languages, many faculty report that questions based on one language often do not translate easily to another. As an analogy, when a textbook is developed for one language, simply translating code to another language does not produce a satisfactory book for the second language. While some authors try such an approach, both students and teachers recognize major weaknesses in the translated book, and complaints abound. Because similar issues apply to the development of exams, a multi-language option for APCS would require rethinking the CS curriculum for each language option. Such factors indicate that the production of comparable exams in this context is not feasible over the long term.

Since each AP discipline must identify a course curriculum and develop an examination that will allow students to receive credit and/or placement in colleges, it also is vital to consider the likely impact of multiple-language options on the college acceptance of APCS grades. In particular, the current one-language format yields a single, well-defined examination for evaluation. Colleges and universities know exactly what the examination covers, grades are comparable from year to year, and schools can build on their experiences from past years in making placement decisions. Further, when introductory courses cover special topics that are not part of the

APCS curriculum, colleges and universities can develop special accelerated courses for AP students so that they can pick up concepts that were not covered without having to repeat material with which they will be familiar.

Many of these characteristics are lost with a multiple-language option. Some language choices support different problem-solving paradigms; all languages support different libraries; and each language has its own built-in capabilities. With different APCS options, AP students could have substantially different experiences and backgrounds – even with the same APCS grade. Such variations would significantly complicate decisions for possible placement and credit. Development of special courses for APCS students similarly would be affected. It seems likely, for example, that colleges and universities might tie their placement options to which language a student followed. Such a result would increase pressures on the high school teacher – as discussed above – to choose the "right" language alternative.

Different Languages for APCS A and APCS AB

The use of different languages for APCS A and APCS AB is sometimes proposed as a variation of the suggestion to use multiple languages. Such an approach might fit nicely with the introductory sequence at some colleges and universities where different languages are used in the first year. Although some colleges and universities successfully use two languages in CS1 and CS2, the vast majority use a single language.

At the high school level, distinctions between APCS A and AB are not as clear as between CS 1 and CS 2 in colleges. Students studying the A material may be in the same class as students studying the AB material. In some settings, students in an A course may be preparing for the AB exam on their own, as an independent study in addition to the A course. Students in high schools do not usually have the option to drop a course, so students enrolled in an AB course may opt to take the A exam if they have not mastered the AB material.

These constraints mean that the material for the APCS A examination must be a subset of the AB material. This does not allow independent languages for the two examinations. Teacher professional development is much simpler when one language is used than when two languages are used. However, many high schools have a programming course that is a prerequisite to an AP course. Such a course is not constrained to use the same language as the language of the AP course.

Multiple Languages During the Transition from C++ to Java

Although the long-term use of multiple languages for the APCS examination is not feasible, as described above, some argue that the use of both C++ and Java for one or two years might facilitate the transition. Such an approach would allow schools with adequate materials and facilities, and staffed by teachers with prior APCS experience, to make the transition to Java quickly. Such an approach also would provide more time for the language change for those schools or instructors that need it. In practice, however, this approach would not work because we expect an AP Computer Science course that uses Java will have much more emphasis on object orientation than a course delivered in C++. Thus, it would not be simply a matter of having the same AP course and exam in two languages; two entirely different AP courses and exams would be necessary.

Discrete Mathematics and AP

AP courses are intended to reflect material that is covered in the introductory college course. The AP Examination then provides a mechanism for high school students to gain college credit and/or placement for appropriate mastery of this material. Few colleges and universities teach discrete math in the first year. Instead, the vast majority teach programming and problem solving. Colleges and universities simply would not provide credit or placement for CS1-CS2 based on preparation in discrete math.

Discrete mathematics does provide important analytical tools and mathematical maturity, which subsequent computer science courses utilize. There may be much merit in having an AP discrete math course, but this topic was not part of the committee’s scope.

Teacher Professional Development

Appropriate and sufficient training of AP Computer Science teachers is a necessity for a successful transition to Java, and basic materials and documents need to be made available in an efficient manner.

- Subset of the Java language to be included in APCS:
  The APCS Ad Hoc Committee is recommending a subset of the Java language to be included in APCS. Refinement of this subset is an initial step for the smooth transition to a new language.

- APCS Course Description:
  The APCS Course Description ("acorn book") will need to be revised to place a greater emphasis on object orientation. The language-based sample questions and solutions and the information describing the APCS language philosophy, subset, and rationale will be rewritten.

- Teacher's Guide to AP Computer Science:
  This teacher's guide will need to have syllabi from various schools (high schools and colleges) for Java curricula. Much of the introductory information is still applicable with the change to Java.

- Marine Biology Case Study and Teacher's Manual:
  The Ad Hoc committee recommends that the present Marine Biology Case Study be implemented in Java emphasizing object orientation. This Case Study and its Teacher's Manual will be revised to reflect these changes.

- Additional Teacher Resources:
  Funding for the development of online resources to facilitate the transition to Java should be investigated. Such resources should include (but not be limited to):
  - PowerPoint presentations on selected Java topics and object orientation
  - Examples demonstrating correct language issues
  - Sample assignments and projects with solutions
  - Sample multiple-choice and free-response questions

To accomplish all tasks successfully and to allow for a comfortable transition to Java, a time line of activities is provided in the appendix.

Appendix

Safety

The programming language used in introductory computer science courses should promote best practices while permitting students to solve real problems using a computer. A well-accepted best practice is safety. Although there is some confusion about what a safe programming language, is we are using safety to mean:

Any attempt to misinterpret data is caught at compile time or generates a well-specified error at run time.

Safety is important at all levels of software design but especially when learning to program. Beginning students should encounter repeatable errors when they are developing and debugging programs and should be prevented from making errors that arise from inadvertently misinterpreting data. At a more advanced level, a safe programming language protects against malicious attacks based on intentionally misinterpreting data.

Java is a safe programming language. The Java compiler catches many inadvertent errors and the Java run time environment ensures that any attempt to misinterpet data will generate a well-defined exception. Java also supports garbage collection, freeing the programmer from the need to manage memory allocation and de-allocation. Garbage collection promotes safety. It is difficult to manually manage memory allocation correctly; incorrect management (such as returning storage to the heap that is still in use) can cause data to be misinterpreted and lead to errors.

Why Java is Safe and Why C and C++ are Not Safe

In Java, it is impossible to have an uninitialized variable. Instance variables are automatically initialized with default values and local variables must be initialized or a compilation error results. In contrast, uninitialized variables in C++ can cause errors that may be difficult to fix since the source of the uninitialized variable may be far from where the error occurs.

In C, an out-of-bounds array index accesses data/memory as though the memory is of the same type as the array. Such indexing cannot, in general, be caught at compile time and in C does not cause a well-specified error at run time. Programs with bad array indexes in C may generate expected output on one run, incorrect output on another run, and cause the computer to crash on a third run.

Array-indexing errors can be trapped in C++ using a class such as the `apvector` class. Such errors engender well-specified behavior that can be used to diagnose the errors when they occur in a program. However, in C++ it is possible to cast data or values from one type/class to another. Bad casts can generate incorrect output or be used maliciously to cause real harm. This is especially true when pointers are used.

Simplicity

The language used to teach programming in an introductory computer science course should be *simple*. That is, based on using a single programming language. This is not a prerequisite for a one-year curriculum, but it is the simplest approach. In making recommendations about changes in AP Computer Science, the ad hoc committee considered using several languages. (See "APCS Course Content and Exam Alternatives" for a discussion of this issue.)

A simple language is grammatically as well as semantically uncomplicated. It is easy to learn, because the notation expresses programming concepts in a straightforward way. Similar concepts are expressed using similar notations. There is little chance of mistaking what a notation means, and little chance of ambiguity. By and large, a small typographical error does not result in a syntactically legal program with an entirely different meaning.

Mathematicians and other scientists create notations to help solve problems. Sometimes, it is only when a suitable notation emerges to express a new concept that progress is made in a field. Mathematicians and other scientists have the luxury of being able to use whatever notation fits a problem. In a classroom, they can switch notations on the fly, using whichever is most suitable at the moment. Computer scientists teaching programming in an introductory course are hampered by the need to teach a notation that is implemented – and usually only that notation – because running and debugging programs is an important part of the educational experience. For this reason, the choice of a simple language in a programming course is extremely important.

Of course, simplicity can be taken too far. To paraphrase Einstein, the language used in an introductory programming course should be as simple as possible, but no simpler. It must support the principles and concepts that are to be taught, but with minimal syntactic and conceptual overhead. It should support the development of large as well as small programs. The same language features and design principles should be useable in the first week and the last, when the students have more experience. The language and its development environment should be simple for beginning students while providing support for the more advanced work that is covered later in the first year. The language (and its libraries) should support a wide range of applications that may not be essential in every first-year program but that provide an opportunity for pursuing interesting and advanced concepts early in the study of software engineering and computer science.

Following are some more particular requirements.

- The language should straightforward and simply support object-oriented programming, for this is the best current means of being able to scale from small to large programs.
- The language should not force students to study technical details that detract from the principal goals of the course.
- Programs should be readable. For example, most keywords should serve one purpose rather than many, and concepts should be expressed using keywords, rather than by convention, as much as possible.
- The language should allow student-friendly subsets – students who accidentally stumble outside a subset should not be penalized with incomprehensible error messages or programs that produce counterintuitive behavior.

Simplicity, Java, and C++

Nearly every introductory programming course uses a subset of a programming language, rather than the full language. A well-defined subset can make the language simpler, but not every language wart can be hidden or turned into a feature. The following comparison of some language features illustrates how simplicity helped guide the ad hoc committee's recommendation to switch to Java.

Avoiding cleanup chores

It is impossible to teach object-oriented concepts in C++ without discussing dynamically allocated and deallocated objects. Moreover, one must always deal with the problem of deallocating objects when they are no longer used. In contrast, Java has automatic garbage collection, which means that one doesn't even have to mention deallocating objects.

Hiding confusing implementation details

Inheritance in C++ requires the use of pointers, which forces a distinction between a variable `p` and the value `*p` to which it points – a source of great confusion for students. For example, in C++, both `p` and `*p` can be assigned to variables.

In Java, there is no equivalent of the C++ expression `*p`. In Java, p does denote a reference to the object, but no expression yields the object itself; i.e., there is no way to assign the object (in C++, `*p`) to a variable. Thus, in Java there is none of the pointer chasing with regard to objects that goes on in C++. This distinction greatly simplifies the teaching of object-oriented programming and allows a much higher-level model of execution.

Avoiding complex choices

C++ has three modes of parameter passing: by-value, by-reference, and by-const-reference. Students must understand each of these modes and must decide in each situation which mode to use. In contrast, Java has one mode of parameter passing: by-value.

In C++, stack-allocated objects and dynamically allocated objects have different semantics. The former are copied as values, because each object has a copy constructor, an assignment operator, and a destructor. This makes using composite objects syntactically similar to using basic types. But implementing objects is difficult, and the difference between the two kinds of objects is a source of confusion.

In the first code fragment below, `p` and `q` are unequal after execution of the code. In the second code fragment, the objects referenced by `p` and `q` are equal, because they are dynamically referenced.

```
{
   // C++ code follows
    Thing p(...);
    Thinq q(...);
    p = q;          // p and q are equal after this statement
    p.changeMe();   // p and q are not equal after this statement
}
// p and q are "destructed"

{
    // C++ code follows
    Thing * p = new Thing(...);
    Thing * q = new Thing(...);
    p = q;          // potential memory leak
    p->changeMe();  // p and q are equal after this statement
}
// memory allocated dynamically isn't accessible and isn't reclaimed
```

Some texts [e.g., Reiss, A Practical Introduction to Software Design with C++] advocate using only pointers to avoid the confusion and problems associated with using statically allocated objects. This approach is probably not feasible in introductory courses.

In Java, all objects are dynamically allocated, and non-primitive type variables are always pointers. Assignment always copies values, never objects. This distinction is subtle but profound when programming using polymorphism, since polymorphism in Java and C++ requires dynamic allocation.

```
{
    // Java code follows
    Thing p = new Thing(....);
    Thing q = new Thing(....);
    p = q;
    p.changeMe();   // p and q are equal after this statement
}
```

Avoiding syntactic struggle

Inheritance with polymorphism is possible in both C++ and Java, but syntactic problems with implementing inheritance in C++ makes it hard for beginning students to learn and use.

In Java, the word `extends` is used to express inheritance. In C++, the word `public` preceded by a colon is used in a class interface.

```
  class SubThing : public Thing {....};
```

C++ then confuses the issue by using `public` for a second, totally different, reason: the public section of a class is identified by the word `public` followed by a colon.

```
class Thing
{
   public:
     Thing();
   ...
};
```

In Java, an abstract class is indicated using the word `abstract` in the class definition.

```
public abstract class Thing
{...}
```

C++ has no such keyword; instead, an abstract class is a class that contains at least one pure virtual function. C++ pure virtual functions are both syntactically and semantically complex (must be overridden, but can have an implementation) compared to the Java counterpart of abstract functions.

Relying on reasonable defaults

When a programming language provides a default choice for a mechanism, the choice should reflect generality and flexibility, rather than efficiency or historical accident.

In C++, the default for inheritance is private inheritance, which is the wrong default.

```
class SubThing : Thing // C++, private inheritance
```

In object-oriented programming, a `SubThing` should be a subtype of a `Thing`; it should not just privately inherit `Thing` for implementation reasons. Thus, one must supply the keyword `public`:

```
class SubThing : public Thing // C++, public inheritance
```

In Java, by default, all methods are polymorphic/extendable in subclasses. In contrast, C++ methods cannot be polymorphic/extendable unless the word `virtual` is used.

In Java, objects are implicitly accessed through references. In C++, pointers must be explicitly chosen.

Libraries

Ideally the goals of a course will be realizable using the core language and library rather than third-party software solutions. Such solutions, like `apstring` and `apvector` in the current AP Computer Science course, can make the language simpler and safer to use, but they incur expense in development, maintenance, and distribution.

Java has standard classes for strings and dynamic arrays. In addition, the standard Java library contains a large number of useful classes for graphics, user interfaces, network programming, database access, and other applications. First-year students need not master these classes, but a cursory knowledge of some of these classes is enormously helpful in building interesting examples, supplementary labs, semester projects, and so on. The Java standard library supplies this functionality portably on many platforms. In C++, an instructor would need to procure non-standard and non-portable third-party solutions.

Language subsets

Stepping even slightly outside a C++ subset (such as the AP C++ subset) can create situations that are truly frustrating both for students and instructors. In some cases, the results are inscrutable error messages. In other cases, linker errors whose sources are difficult to determine can be frustrating. Perhaps worse, students can write code that compiles without error messages but that executes mysteriously, such as the classic

```
if (x = 0) // oops...was supposed to be x == 0
```

In Java, these situations are much less common, because Java is so much simpler than C++. There are many fewer possibilities for ambiguity – for example, the implicit conversion of a numeric value to a Boolean value does not exist in Java. Thus, subsetting Java primarily entails subsetting the library rather than the language.

Object Orientation

Object orientation, involving encapsulation, inheritance, polymorphism, and abstraction, is an important approach in programming and program design. It is widely accepted and used in industry and is growing in popularity in the first and second college-level programming courses. It facilitates the reuse of program components and the management of program complexity, allowing large and complex programs to be written more effectively and efficiently and with more confidence in their correctness than with the more traditional purely procedural approach.

Object orientation is possible in C++, but is both syntactically and semantically more difficult than in Java. For example, in C++ methods are not polymorphic by default, but require adding the key word `virtual` to the method. Because C++ is not garbage collected, a well-designed class with one virtual function should have a virtual destructor as well. This makes it difficult to write simple inheritance hierarchies in C++. Creating abstract classes is cumbersome in C++ and multiple inheritance from non-abstract classes is difficult to do properly.

In contrast, Java methods are polymorphic by default and implementing abstract classes is simple. Multiple implementation inheritance is not possible in Java, but Java interfaces are an elegant way to implement a useful form of multiple inheritance.

Encapsulation

Encapsulation is the principle method for separating the interface of an abstraction from its implementation. For example, programmers using double variables and expressions do not need to know about how the IEEE standards for 64-bit, double-precision floating point values are implemented. Encapsulation is primarily achieved through information hiding. In object-based programming, encapsulation with classes is realized with public behavior but a private implementation. Encapsulation is a useful tool and a cornerstone of the current APCS course description, but by itself it does not provide enough flexibility and power in structuring large programs.

Inheritance

Inheritance is the primary mechanism for thinking about object orientation and is extremely important for the reuse of program components. When used in conjunction with good design strategies, inheritance makes it possible to extend existing code with minimal modifications. This helps ensure reliability and correctness.

For example, consider a program that simulates creatures living in an ocean. For simplicity, we assume that creatures behave only by swimming and eating. If we model our simulation on this abstraction we might initially have two kinds of creatures: those that eat plants and those that eat fish. Creatures might swim in the deep ocean (e.g., squid) or on the surface (e.g., jellyfish). By using an inheritance hierarchy of creatures, we don't need to anticipate every form of eating and swimming in advance. We can write the simulation framework to generic swimming and eating behavior and provide new subclasses to the existing framework if we encounter a new kind of creature; the framework does not change when new classes are added. In addition, behavior and/or state such as a creature's location, which is common to all creatures, does not need to be duplicated in each creature class. This helps simplify maintenance because code is not duplicated in every creature class.

Inheritance and Java

Java has two notions of inheritance: one in which some behavior and state may be inherited by subclasses and one in which no implementation of either behavior or state is inherited, only the names of functions are inherited. These correspond in Java to *class extension* and *interface implementation*, respectively. A simple example of class extension and inheritance is illustrated by the default `toString` method inherited since every class extends the class `Object`. Subclasses often override this to provide a class-specific string used when an object is output, but the default behavior identifies the class as well as a form of object identifier. Perhaps the simplest example of an interface is `Comparable`. Classes that implement this interface must supply a method `compareTo` that determines a total order on instances of the class. This permits generic classes in Java libraries to search and sort `Comparable` objects.

In Java, a class can extend only one superclass, but implement any number of interfaces. This prevents problems that can arise with multiple inheritance.

Polymorphism

Polymorphism makes it possible to call methods on an object without knowing which of the classes in an inheritance hierarchy the object belongs to. Polymorphism is the primary mechanism used with inheritance to develop extensible and reusable code.

In the earlier example about creatures in an ocean, the method call `creature.swim()` will work regardless of what class `creature` belongs to because `swim` is a polymorphic function. This means the actual implementation of `swim` that is executed is determined at run time based on the class of `creature`. A subclass can reuse the swim function of its parent class without changing it, replace the swim function with its own swimming behavior, or augment the parent class swimming behavior with additional behavior.

Time Line for Teacher Professional Development

Appropriate and sufficient training of AP Computer Science teachers is a necessity for a successful transition to Java, and basic materials and documents need to be made available in an efficient manner.

- Subset of the Java language to be included in APCS:
  The APCS Ad Hoc Committee is recommending a subset of the Java language to be included in APCS. Refinement of this subset is an initial step for the smooth transition to a new language.

- APCS Course Description:
  The APCS Course Description ("acorn book") will need to be revised to place a greater emphasis on object orientation. The language-based sample questions and solutions and the information describing the APCS language philosophy, subset, and rationale will be rewritten.

- Teacher's Guide to AP Computer Science:
  This teacher's guide will need to have syllabi from various schools (high schools and colleges) for Java curricula. Much of the introductory information is still applicable with the change to Java.

- Marine Biology Case Study and Teacher's Manual:
  The Ad Hoc committee recommends that the present Marine Biology Case Study be implemented in Java emphasizing object orientation. This Case Study and its Teacher's Manual will be revised to reflect these changes.

- Additional Teacher Resources:
  Funding for the development of on-line resources to facilitate the transition to Java should be investigated. Such resources should include (but not be limited to):
  - PowerPoint presentations on selected Java topics and object orientation
  - Examples demonstrating correct language issues
  - Sample assignments and projects with solutions
  - Sample multiple-choice and free-response questions

To accomplish all tasks successfully and to allow for a comfortable transition to Java, the following timeline of activities is recommended.

September 2000:

Before November 10, 2000, a brief description of a SIGCSE "Birds-of-a-Feather" session (i.e., round table discussion) will be written and submitted to the appropriate contact person (see next item for specifics). The goal of the SIGCSE presentation is to solicit colleges/universities to work with AP to provide graduate courses for teachers.

October 2000:

"A well defined group of individuals" will explore methods to develop graduate credit courses (both online and on campus) to teach APCS teachers the revised curriculum implementing Java and emphasizing object orientation. This will be discussed at SIGCSE 2001 in a "Birds-of-a-Feather" session.

A core group of College Board AP Computer Science Workshop Consultants (6-8) will be formed to develop presentation materials for APCS workshops. These materials will guarantee that accurate information about the Java transition is disseminated and that key topics of the APCS Java curriculum are introduced in an appropriate manner.

January 2001:

The philosophy and rationale for the APCS change to Java is complete.

February 2001:

A two- to three-day meeting of the core APCS workshop consultants will take place.
This group will prepare a 20- to 30-minute presentation for the College Board APCS workshops. The presentation (P#1) will provide teachers with necessary information and materials required to formally request any financial or administrative support that may be needed for the APCS change to Java. It will include information on major changes.

This presentation will also be designed to share the language subset, philosophy and rationale. The presentation will be in PowerPoint format (transparency form of the presentation will be available) and will be distributed to all College Board consultants. This material will be included in all College Board Computer Science workshops beginning in the summer of 2001. This will alleviate much of the teacher anxiety that may exist and will ensure that accurate material is being distributed.

The Java subset, philosophy and rationale will be available to the CS consultants preparing this presentation.

SIGCSE panel discussion will request college partnerships. This will hopefully take place on the SIGCSE day designed for high school teachers.

March 2001:

The Java subset is complete.

June 2001:

A short workshop on Object-Oriented Design will be included in a 1-day workshop for faculty consultants following the 2001 APCS Reading.

Summer 2001:

APCS workshops will include P#1.

Request for SIGCSE workshop on Object Orientation using Java submitted for SIGCSE 2002.

Fall 2001:

APCS workshops will include P#1.

Core APCS workshop consultants will meet to develop a presentation on Object Orientation using Java. (P#2)

January 2002:

Completion of the AP Course Description (Java)
Completion of the Teacher’s Guide to AP Computer Science (Java)

February 2002:

SIGCSE workshop for APCS on Object Orientation using Java is given.

Spring 2002:

The Marine Biology Case Study (MBCS) in Java with emphasis on object orientation is complete.

The Teacher's Manual for Marine Biology Case Study is complete.

Several 2-day College Board workshops on Java with emphasis on object orientation held at selected sites.

Summer 2002:

Workshops for College Board consultants (Two of these will be necessary to accommodate consultants.)

MBCS materials will be available for workshops.

APCS College Board workshops will include P#2.

Fall 2002:

Graduate credit APCS Java courses will begin (on-line and on-site). The courses will stress the APCS curricula as defined in the College Board documents previously discussed.

College Board Computer Science workshops will include more detailed information on the language change.

APCS College Board Workshops will include P#2.

Spring 2003:

Several two-day College Board workshops on Java with emphasis on object orientation held at selected sites.

Additional online resources are available for download and use in the classroom.

Summer, 2003:

Professional night or one-day workshop following AP Reading will include Java course information.

AP Summer Institutes will focus on the revised AP Computer Science Course (Java) and will include reference to available online resources.

Fall 2003:

APCS course taught in Java with an emphasis on object orientation.

May 2004:

First APCS Java examination is given.

AP Computer Science Curriculum Survey for Colleges and Universities

The purpose of the AP Computer Science curriculum survey was to gather information and opinions to help improve the Computer Science courses of the College Board's Advanced Placement Program and to make it consistent with the way introductory college and university courses in computer science are evolving.

The first phase was a language survey to determine the current and future trends in programming language use in the first two courses in the college computer science curriculum. The information gathered from this first phase was analyzed by a College Board Ad Hoc Committee to make recommendations to the AP Computer Science Development Committee for the language to be used in testing the concepts of the AP Computer Science curriculum.

The second phase was a more detailed curriculum that was distributed at the AP Computer Science Reading in June 2000, and also made available on the web during July and August 2000. The results of the second phase were analyzed by the AP Computer Science Development Committee as a basis for any changes that need to be made in the AP Computer Science curriculum.

Results of AP Computer Science Language Survey August 1999

Information for the AP Computer Science language survey was collected by sending e-mail messages to the chairs of Computer Science departments directing them to fill out a form that was available on the World Wide Web. A password was required for individuals to access the survey form. There were 100 responses out of approximately 280 departments that were polled. Of these 100 responses, just under half were from the top 100 AP receiving colleges, and three quarters were from the top 200 AP receiving colleges. The results are summarized in the tables below.

Questions for the CS1 course	**C++**	**C++ or Java**	**Java**	**Other**	**Unknown**
Programming language in current use	49		34	17	
Programming language in three years	18	14	47	15	6
Programming language in five years	8	11	38	8	35

Comments: Many respondents felt that predicting five years into the future was not feasible based on the rapid change of technology within the Computer Science field.

Questions for the CS2 course	**C++**	**C++ or Java**	**Java**	**Other**	**Unknown**
Programming language in current use	57		37	6	
Programming language in three years	24	16	51	3	6
Programming language in five years	13	11	40	2	34

Comments: Many respondents felt that predicting five years into the future was not feasible based on the rapid change of technology within the Computer Science field.

Results of AP Computer Science Curriculum Survey Summer 2000

Versions of this survey were distributed at the AP Computer Science Reading in June 2000 and on the web during July and August 2000. For the web version of the survey, solicitations were sent to the AP Computer Science e-mail list and to the ACM SIGCSE members e-mail list.

1. What language is used in the CS1 course at your college?
   - C++ – 41
   - Java – 22
   - Scheme – 5
   - Other – 3 (C, Ada)

2. What language is used in the CS2 course at your college?
   - C++ – 38
   - Java – 31
   - Other – 4 (Scheme, Ada)

3. Do you expect the language to change in the next 3 years? If yes, what language will you use?

Yes	19	(16 to Java, 3 to Scheme)
No	41	(15 staying with C++, 23 already use Java in at least one course, 2 use Scheme, 1-Ada)
Maybe	8	(6 considering Java)

4. Does your school give credit or placement for APCS A courses?

Credit:	43
Placement:	32

5. Does your school give credit or placement for APCS AB courses?

Credit:	45
Placement:	32

6. Would your APCS credit/placement policy be affected if the language used in the APCS course were to change to Java?

Yes	24
No	37
Probably Not	7

Comments:

The language for APCS affected the decisions of those who responded Yes.
Of those responding No, 2 do not currently grant credit or placement.

Introductory Computer Science Topics

Please indicate in which course the following topics are covered in your computer science curriculum.

Category	Topics	CS1	CS2	Not in CS1/2
Object-Oriented Design				
	Encapsulation	47	47	
	Abstraction	53	47	
	Inheritance	21	46	9
	Use of library classes	49	42	3
Data Structures				
	1-D arrays	64	11	
	2-D matrices	49	41	1
	Trees	9	60	5
	Linked Lists	14	64	1
	Stacks/Queues	12	63	1
	Heaps	5	40	26
	Priority Queues	5	42	24
	Sets/Maps	9	30	32
Algorithms				
	Searching	56	50	1
	Sorting $O(n^2)$ $O(n \log n)$ Sorting comparisons	44 17 3 6	57 13 19 19	3 2 2 1
	Big-O analysis	12	53	3
Other Topics				
	Recursion	34	54	
	Assertions	24	35	17
	Invariants	21	44	18

Are there other topics that you cover in your CS1/2 courses that are not part of the APCS curriculum?

Applicative programming, higher order functions, interface design, testing, multiparadigm approach, exceptions, breadth first topics, data representation, various design methodologies, graph theory, intro to proof of correctness, software engineering principles, hashing, storage management, GUI based programming, iterators and collection views, some multithreading.

Additional Comments

Colleges may be more anxious to move to Java if it is known that the College Board is expecting to move in that direction. Let’s try to get above the language issues if possible. Design test to be as independent of language as possible.

I think you need to strongly consider the impact of these decisions on teachers in states that do not certify computer science teachers. OOP may be accessible for someone with a strong background in CS, but for the re-trained math/physics teacher, it is another story. This move could endanger not only the number of AP programs, but also the quality!