Journal of Information Technology Education Volume 5, 2006

A Formal Language Selection Process for Introductory Programming Courses

Kevin R. Parker
Idaho State University
Pocatello, Idaho, USA

parkerkr@isu.edu

Joseph T. Chao
Bowling Green State University
Bowling Green, Ohio, USA

jchao@bgnet.bgsu.edu

Thomas A. Ottaway
Idaho State University
Pocatello, Idaho, USA

ottathom@isu.edu

Jane Chang
Bowling Green State University
Bowling Green, Ohio, USA

changj@cba.bgsu.edu

Executive Summary

The selection of a programming language for introductory courses has long been an informal process involving faculty evaluation, discussion, and consensus. As the number of faculty, students, and language options grows, this process becomes increasingly unwieldy. As it stands, the process currently lacks structure and replicability. Establishing a structured approach to the selection of a programming language would enable a more thorough evaluation of the available options and a more easily supportable selection. Developing and documenting an instrument and a methodology for language selection will allow the process to be more easily repeated in the future.

The objectives of this research are to: i) identify criteria for faculty use when selecting a computer programming language for an introductory course in computer programming; ii) develop an instrument that facilitates the assignment of weights to each of those selection criterion to determine their relative importance in the selection process, and; iii) allow various computer programming languages to be scored according to those selection criteria. A set of criteria for the selection of a programming language for introductory courses proposed in a previous paper is briefly reviewed here, with each criterion accompanied with a definition and justification. Readers are referred to the source paper for a complete discussion and literature review.

In order to test the validity of these criteria a pilot study was conducted. That study revealed that the number of languages being evaluated by a respondent should be limited, and better guidance in the form of criterion explanation and rating guidance are necessary. Further, some users found the number of criteria daunting, and some of those criteria overlap, causing a language to be evaluated multiple times on what should be a single criterion. At the same a few additional criteria were proposed by study participants.

As a result of these findings, instrument refinements were made. Evaluators are now restricted to assessing only languages with which they are quite familiar in order to address not only the issue of quantity, but also inadequate familiar-

Material published as part of this journal, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact Publisher@InformingScience.org to request redistribution permission.

ity. In addition, the selection criteria were analyzed and those with commonalities were grouped together, and a few additions were made to the subcategories as proposed by the study participants. The most significant change is the use of Multi-criteria decision analysis, specifically the Analytic Hierarchy Process (AHP), to provide structure to the weighting process. These techniques are explained, and their suitability to this process is investigated. An online instrument based on AHP that includes a clarified description of each criterion is being refined to assist in the administration of future tests.

This set of criteria, as well as the instrument designed around it, are designed to be extensible, allowing revision of both the criteria and the process as new programming paradigms and languages are introduced and old ones fall out of favor. It is hoped that this formal method will yield the structure and repeatability missing from existing approaches.

Keywords: Programming language selection, introduction to programming, teaching programming, programming language evaluation, multicriteria decision analysis, analytic hierarchy process.

Introduction

A cursory glance through back issues of computer-related journals makes it apparent that discussions about the introductory programming language course and the language appropriate for that course have been numerous and on-going (Smolarski, 2003). The selection of a programming language for instructional purposes is often viewed as a tedious chore because there is no well-established approach for performing the evaluation. However, the choice of a programming language has serious education repercussions (Schneider, 1978). Dijkstra (1972, p. 864) stated that

> "…the tools we are trying to use and the language or notation we are using to express or record our thoughts are the major factors determining what we can think or express at all! The analysis of the influence that programming languages have on the thinking habits of their users … give[s] us a new collection of yardsticks for comparing the relative merits of various programming languages."

The informal process may involve faculty discussion, with champions touting the advantages of their preferred language, and an eventual consensus, or at least surrender. Because the process must be repeated every three or four years it would be preferable to develop a structured approach to make the process more systematic.

The goal of this study is to develop and refine an instrument to facilitate the selection of a programming language and to make the process more uniform and easily replicated. The original paper (Parker, Ottaway, & Chao, 2006) proposed an objective selection process. A pilot study was conducted to test the viability of the process. The following steps outline the proposed approach that guided the pilot study.

1. Compile a list of language selection criteria.
2. Weight each of the criteria. Ask each evaluator to weight, specific to the department's needs, the value of importance for each criterion.
3. Determine a list of candidate languages. The list should be comprised of languages nominated by the faculty rather than a complete list of available languages.
4. Evaluate the language. Each candidate language should be assigned a rating for each criterion.

5. Calculate weighted score. For each candidate language, a weighted score can be calculated by adding together the language score multiplied by the weight assigned to each criterion. The language with the highest weighted score is the optimal choice, given the evaluators' assessments.

This paper first reviews a set of criteria designed for use in the selection of an appropriate programming language for introductory courses. It then describes the structure of the pilot study and resulting findings. Finally, the paper advances refinements in the process that should circumvent the problems that were discovered in our original proposal.

Criteria

Table 1: Selection Criteria

Reasonable Financial Cost
Availability of Student/Academic Version
Academic Acceptance
Availability of Textbooks
Stage in Life Cycle
Industry Acceptance
Marketability (Regional and National)
System Requirements of Student/Academic/Full Version
Operating System Dependence
Proprietary/Open Source
Development Environment
Debugging Facilities
Ease of Learning Fundamental Concepts
Support for Secure Code
Advanced Features for Subsequent Programming Courses
Scripting or Full-Featured Language
Support of Web Development
Supports Target Application Domain
Teaching Approach Support
Object-Oriented Support
Availability of Support
Instructor and Staff Training
Anticipated Experience Level for Incoming Students

A previous paper proposed criteria for the selection of a programming language for introductory courses. The criteria were derived by perusing over sixty papers relevant to language selection and justified by a brief review of the supporting literature in (Parker et al., 2006). Each of the criteria in Table 1 has been used in one or more previous studies that evaluate programming languages.

A complete literature review and justification for each of the criterion can be found in (Parker et al., 2006), but a brief explanation of each follows.

Reasonable Financial Cost

This criterion refers to the price to acquire the programming language or the development environment. This may involve individual packages or a site license for a network version. There may be an academic discount for educational institutions, there may be an alliance in which the university or department can enroll, or there may even be a free, downloadable version.

Availability of Student/Academic Version

The availability of a student version or academic version allows students to install the development environment on their personal machine, making it more convenient for them to work on their assignments when the computer lab is not accessible. If a student version is unavailable and the department uses a network-based version, then students may be forced to work on their assignments in campus labs, restricted by hours of operation, availability of transportation, etc. If the academic version is stripped down, then the benefit to the students may not be as great, but this factor should at least be considered.

Academic Acceptance

Academic acceptance refers to the popularity of a language at other academic institutions. This can be assessed by current use or projected use at other institutions. For example, the increasing popularity of object-oriented programming and the recent decision by the College Board to move the Advanced Placement Computer Science program to Java have led to an increasing number of universities, colleges, and secondary schools adopting Java as the programming language for their introductory programming courses (Roberts, 2004).

Availability of Textbooks

The availability of text books is affected by many factors. The life cycle stage of the language impacts the availability of textbooks, particularly when the language is relatively new. It is often difficult to find a quality textbook for a newly released language, but as a language matures more become available. The academic acceptance of a language also plays a large role in the availability of textbooks because a larger potential market exists for a text that deals with a more widely used language. Finally, textbook availability may also be affected by the teaching approach used. For example, functions-first, objects-first, or objects-early are all approaches used to teach object-oriented languages, but few recent texts present the material from a functions-first perspective. Availability of reference books should also be taken into account (Lee & Stroud, 1996).

Stage in Life Cycle

A programming language's stage in the programming language life cycle affects not only textbook availability, as noted above, but it may also impact the widespread use of a language in both industry and academia. Universities may prefer a language that is still in its earlier stages, rather than one like FORTRAN that is in its declining years. Not to be confused with the program development life cycle, the programming language life cycle, as described by Sharp (2002), is based on the natural principles of growth, maturation and decay. The processes of natural advantage and evolution operate in the world of programming languages in the same way that they operate in the biological domain, but in the case of languages the main forces are efficiency of expression versus profitable adoption.

Industry Acceptance

Industry acceptance refers to the market penetration (Riehle, 2003) of a particular language in industry, i.e., the use of a language in business and industry. Also referred to as industrial relevance, it can be assessed based on current and projected usage, as well as the number of current and projected positions. King (1992) notes that many language decisions are made on the basis of current popularity or the likelihood of future popularity, but Howland (1997) objects that too many languages are chosen simply because of their current popularity rather than for sound pedagogical reasons.

Marketability (Regional and National)

Marketability refers to the employability of graduates. This may refer to regional or national/international marketability, based on the placement of a program's graduates. Language selection is often driven by demand in the workplace, i.e., what employers want. Not only are marketable skills important in future employability, but students are more enthusiastic when studying a language they feel will increase their employability (de Raadt, Watson, & Toleman, 2003). Emigh (2001) points out a caveat that must be considered when assessing both industry acceptance and marketability. Generally, four to five years pass between when a student begins a program of study and when he or she attains a position requiring programming skills. Even if a curriculum teaches a newer programming language, there is no guarantee that employers will still be looking for that language when the student enters the work force.

System Requirements of Student/Academic/Full Version

The system requirements of the programming language often play a role in the selection process. This includes hardware as well as operating system requirements. The amount of hard disk space needed to install the software, the operating system required, and the amount of memory to run the software all factor into the decision. Both student and lab machines must be able to meet the minimum requirements of the selected language.

Operating System Dependence

This criterion refers to the dependence of a language on a particular operating system platform, often referred to as portability. For example, any of the languages supported by the .Net framework, including Visual Basic, C++, C#, etc., depend on the Windows operating system. Other languages, such as Java, are platform independent, and development environments for Java can be found for a variety of operating systems. This may be of concern to faculty members who may or may not prefer to be bound to a specific operating system.

Proprietary/Open Source

This refers to the entity that controls the evolution of a language and its associated development environment. For example, Microsoft is responsible for additions, deletions, or modifications in any of the languages supported by the .Net framework. Sun is responsible for the ongoing evolution of the Java language. On the opposite end of the spectrum, PHP is an open source language and can be easily modified by any member of the open source community.

Development Environment

The development environment is a programmer's virtual workbench, and can improve or inhibit productivity (Jensen, 2004). Development environments range from simple text editors and command-line compilers to fully interactive and integrated development environments (IDE) (McIver, 2002). An IDE should be easy to use so that the students can concentrate on learning programming concepts rather than the environment itself (Kölling, Koch, & Rosenberg, 1995). There is evidence that well-designed programming environments assist students in learning to program (Eisenstadt & Lewis, 1992).

Debugging Facilities

While this criterion is considered part of the IDE, when assessing a programming language one should evaluate the debugging facilities that accompany the language, i.e., the existence of adequate diagnostic aids (Tharp, 1982). The Ad Hoc AP CS Committee (2000) report states that programming environments should contain extensive tools for tracing and debugging. The error diagnostics should be clear and meaningful (McIver & Conway, 1996), and the language should be robust as well as graceful in failure (Conway, 1993).

Ease of Learning Fundamental Concepts

The learning curve associated with each language or IDE differs greatly between languages. The most obvious recent example is the steep increase in the learning curve from Visual Basic 6 to

Visual Basic.Net. Basic concepts include the sequence, selection, and iteration control structures, as well as arrays, procedures, basic input/output, and file manipulation. In addition to ease of learning, the language must be characterized by concise syntax and straightforward semantics (Conway, 1993).

Coding Safety and Support for Secure Code

This criterion can be used to assess two important factors. The first considers whether the language offers features like strong type checking and array bounds checking, while avoiding features like variants and pointers in unsafe mode. Kölling et al. (1995) note that a language should have a safe, statically checked type system, no explicit pointers, and no undetectable uninitialized variables. The second factor, which is closely related to the first, is the inclusion of security-related features like Java's sandbox, which is intended to limit the memory addresses that a Java program can access.

Advanced Features for Subsequent Programming Courses

Many programs introduce basic programming language features in an introductory course and defer advanced features of the language, like multithreading, until a subsequent course. If multiple programming courses are included in a computing curriculum, one important consideration may be whether a programming language offers adequate advanced features to support an advanced programming course.

Scripting or Full-Featured Language

Programming educators must also choose between full-featured and less complex languages. Some programming instructors prefer scripting languages like Python because they offer sufficient richness to cover most of the requirements of an introductory course while reducing the complexity of the development environment and avoiding many other implementation issues. Full-featured languages, however, offer a more complete set of language features that an instructor may want to address.

Support of Web Development

Many programs consider it essential that today's students have the skills to develop web-based applications. This criterion pertains to the level of web development support that a particular language provides. This is not limited to scripting languages, discussed above, but includes web development technologies like ASP.Net that provide a high level of support for web development but at the same time utilize full-featured languages.

Supports Target Application Domain

This criterion, sometimes also referred to as "problem domain," is included to assess how well a language supports programming for specific applications (Howatt, 1995). Examples of application domain include FORTRAN's support for scientific programming, COBOL's support for business data processing, and RPG's support for report generation.

Teaching Approach Support

This criterion refers to the assessment of how well a language supports the teaching approach preferred by the faculty, i.e., whether the intent is to teach programming concepts, with the language simply being a vehicle through which those concepts are reinforced, or whether the intent is to teach the features of a particular language, such as the many user interface controls offered by Visual Basic.

Object-oriented support

This criterion assesses how well a programming language supports basic object-oriented (OO) concepts like abstraction, polymorphism, inheritance, and encapsulation. The evaluator should consider that some languages that are touted as being object-oriented are merely object-based, meaning that they fail to provide support for all of the OO features listed above. Again, if an OO language is selected, the instructor must choose between an objects-first approach and an objects-early approach.

Availability of Support

This criterion refers to the availability of support staff, including computer lab staff and/or network administrators, to support the teaching and administration of a language. The evaluator must consider the likelihood that their language questions will be answered (Cunningham, 2004) and should also take into account the availability of support through forums or listservs on the Internet, as well as vendor support (Tharp, 1982). The evaluator may want to consider the availability of other resources like teachers' guides, example programs, student workbooks, and programming assignments.

Instructor and Staff Training

This concerns the training required for instructors and support staff, the time needed to learn a language or its IDE, and the availability of qualified instructors to teach a particular language. Adopting a new language requires a willingness on the part of the university to invest in the education of its educators because instructors "must continuously enrich their qualifications, implement new training methods and techniques supplemented with practical methods and techniques supplemented with practical experience; while teaching a new language that is as new to them as it is to their class" (Emigh, 2001, p.2).

Anticipated Experience Level for Incoming Students

The final criterion is the anticipated programming experience level for incoming students. This is important because students' previous experience and training skews their understanding of new programming paradigms and languages (Traxler, 1994). If students coming into a program consistently exhibit the same traits such as previous exposure to a particular language, then it may play a role in language selection. If a program consistently sees students with uniform programming experience, it may be able to adjust its requirements and its programming language selection accordingly.

Pilot Study

Language selection is a complicated issue and for the most part is highly subjective. One major goal of utilizing a thorough list of criteria is to make the highly subjective process more objective. Since the Computer Science department at a medium-size Midwestern university is in the process of revamping the CS1 and CS2 courses, a pilot language selection process was created and a language survey was conducted during the Spring 2005 semester. The following are the steps used in the process:

1. Based on the above criteria, a Language Selection Criteria Survey Form, shown in Appendix A, was created and distributed to the faculty. A detailed description of each criterion was also provided.
2. Each evaluator was required to weight, specific to the department's needs, the importance of each criterion. The weight ranged from zero (don't care) to ten (extremely important).

When multiple evaluators worked together the weights assigned by each evaluator were averaged.

3. Based on previous decisions, evaluators were provided with a list of language candidates. The list contained seven language and platform candidates, with C++/Unix and C++/.NET considered as two distinct choices.
4. A Language Evaluation Form, shown in Appendix B, was developed and distributed to evaluators. Representative code from each candidate language was provided. This code, from the Computer Language Shootout (2003), showed how the Sieve of Erathostenes algorithm for finding prime numbers can be coded in each language. The code was provided in case an evaluator was not familiar with a particular language on the list. Evaluators were asked to assess each candidate language with regard to each of the criteria and assign a rating between zero (extremely low) and ten (extremely high).
5. For each candidate language the score assigned to each criterion was multiplied by the weight assigned to that criterion, and those products were summed to obtain an overall rating for the language. This calculation is often referred to as Multicriteria Scoring Model. That is, for each language alternative *i*, compute a weighted average score L_i as:

   $$L_i = \sum_{j=1}^{n} w_j r_{ij}$$

   where

   w_j = weight for criterion *j*

   r_{ij} = transformed score for language alternative *i* on criterion *j*

   The language with the highest weighted score was deemed to be the optimal choice based on evaluators' assessments.

The survey was administered to a group of faculty members who volunteered to participate in the pilot study because of their interest in selecting a programming language for the introductory programming courses.

Initial Findings

Only three of six volunteers completed and returned the survey in the allotted two-week period. Some of the volunteers who did not complete the survey commented that the survey was too difficult and time consuming to complete. Some of the volunteers brought up issues concerning the survey, as summarized below.

1. Too many languages were included on the language evaluation form. The number of languages on the list made the survey more difficult and time consuming.
2. Many of the faculty members were not familiar with all languages listed. If an evaluator rated an unfamiliar language then that rating was suspect.
3. There are too many criteria on the list and some of them seem to overlap. This overlap may result in a language being double penalized, or conversely double rewarded. For example, the criteria "Academic acceptance" and "Availability of textbooks" may be highly related to each other, causing a language to be evaluated twice on what should be a single criterion.

4. Arbitrarily assigning weighting factors to multiple criteria is a difficult task. Some form of structure for the process should be considered.
5. The wording on the criteria provided inadequate guidance. It must be clear to every evaluator which extreme is preferable and therefore should be assigned a higher rating. For example, if a language is available only for a specific operating system, should it receive a ten for "Operating system dependence" or a zero? Similarly, should a pure scripting language receive a ten or zero for "Scripting or full-featured language?"
6. There were suggestions for the inclusion of additional criteria.

Whether they completed the survey or not, most volunteers observed that the framework made the selection process more systematic. They also noted that it made the process more open since the individual assessments were not as easily dominated by more opinionated and vocal faculty members.

Instrument Refinement

The above concerns are all valid and must be addressed. Each will be the focus of individual sections below, followed by a discussion of additional refinements to the language evaluation instrument. The refined language selection criteria survey form is shown in Appendix D, followed by the refined language evaluation form in Appendix E.

Reduction in Number of Candidate Languages

There are several possible solutions to address the problem of having too many candidate languages to evaluate. While it might be possible to first compare a subset of languages to narrow down the choices, the authors preferred to restrict evaluators to assessing only languages with which they are quite familiar. This addresses not only the issue of quantity, but also inadequate familiarity.

Compensation for Language Unfamiliarity

One solution to this problem would be to have everyone involved in the selection process meet prior to the conducting their individual assessments to discuss the various options in case some are more familiar with certain environments than others. Perhaps a more effective approach would be to require the evaluator to associate a confidence level with each of their assessments for the criteria. The intent would be to associate a low confidence level with "guesses" to reduce their impact. Ultimately, as noted in the previous section, the most direct solution for inadequate evaluator familiarity with the language candidates is to restrict evaluators to assessing only languages with which they are familiar.

Reduction in Number of Criteria

Multiple respondents in the pilot study felt there were too many criteria to assess and there was some amount of overlap in the criteria presented. For this reason the selection criteria were analyzed and those with commonalities were grouped together. The groupings appear in Table 2. This serves to both reduce the number of criteria as well as to eliminate overlap among criteria.

Table 2: Higher Order Selection Criteria

Software Cost • Reasonable financial cost for setting up the teaching environment • Availability of student/academic version
Programming Language Acceptance in Academia • Academic acceptance • Availability of textbooks
Programming Language Industry Penetration • Language's stage in life cycle • Industry acceptance • Marketability (regional & national) of graduates
Software Characteristics • System requirements of student/academic/full version • Operating system dependence • Open source (versus proprietary)
Student-Friendly Features • Easy to use development environment • Good debugging facilities
Language Pedagogical Features • Ease-of-learning basic concepts • Support for safe programming • Advanced features for subsequent programming courses
Language Intent • Full-featured language (versus scripting) • Support of Web development
Language Design • Real or Customized • Support for target application domain (such as scientific or business)
Language Paradigm • Methodology or Paradigm • Support for teaching approach (function first, object first or object early)
Language Support and Required Training • Availability of support • Training required for instructors and support staff
Student Experience • Anticipated programming experience level for incoming students

Structured Weighting Process

Some respondents in the pilot study pointed out that assigning weighing factors to many criteria is not an easy task. Results indicated that there is a potential inconsistency because an evaluator may assign a higher value to criterion X than to criterion Y, despite the fact that he or she actually believes that criterion Y is more important than X. For example, an evaluator may independently assign a five to "Reasonable Financial Cost" and a six to "Academic Acceptance," but if specifically asked about those two criteria with respect to each other the evaluator may indicate that "Reasonable Financial Cost" should weigh more than "Academic Acceptance" in the selection process, inconsistent with the previously assigned weights. To reduce such inconsistencies, a more formalized and rigorous approach to the evaluation of selection criteria and scoring of programming languages was applied.

Multicriteria decision analysis (MCDA) provides an array of tools for structured decision making. The types of decisions for which MCDA is most appropriate are those that involve selecting from multiple options the single option that most closely meets a set of weighted objectives. In this case, we seek to select the computer programming language that most closely meets the objectives, where the objectives are the selection criteria previously defined. One particular MCDA method, the Analytic Hierarchy Process (AHP), is particularly appropriate for the type of analysis required in this research.

The Analytic Hierarchy Process, developed by Saaty (1990), utilizes a series of pairwise comparisons to derive both weights and rankings of the selection criteria. The application of AHP requires that each criterion be compared with every other criterion. This is accomplished by asking a series of questions comparing the two criteria and asking the respondent to indicate their degree of preference for one criterion over the other. Typically the respondent is presented with a scale, such as the one shown in Table 3.

Table 3: Representative scale

Equally important	1
Moderately more important	3
Strongly more important	5
Very strongly more important	7
Overwhelmingly more important	9

The scale is used for their response in assessing their preference between each pair of criterion. The responses to these comparisons are then normalized, using matrix algebra, and weights that indicate the relative ranking of the importance of the selection criterion are derived. A brief demonstration of the application of AHP is presented in Appendix C.

Prior to applying AHP, several design issues must be addressed. AHP requires pairwise comparisons of selection criteria. For *N* selection criteria this will require $(N^2 - N)/2$ comparisons. As previously noted, several respondents in the pilot study felt there was some amount of overlap in the criteria presented. For this reason the selection criteria were clustered, yielding the higher order selection criteria listed in Table 2. While this yields fifty-five pairwise comparisons, there are only eleven high-order attributes for the respondents to assess. In addition to reducing the number of selection criteria for the respondent to assess, this approach also reduces the possibility of overlapping criteria and provides a structured approach to assigning weights to the criteria.

While the authors have not yet done so, AHP can be applied through the use of an online method similar to that proposed by Vihakapirom and Li (2003). This will streamline the weighting process and subsequent calculations.

Criteria Guidance

Some respondents complained that the wording of the criteria provided inadequate guidance. For example, “Operating system dependence” does not indicate whether the evaluator should assign a zero or a ten for a language that is available only for a specific operating system. Such criteria are curriculum-dependent, and must be discussed by the individuals participating in the evaluation prior to beginning their language assessment, and a preference should be determined and used as a basis for all evaluators. In addition, a clarified description of each criterion will be included in future applications of the selection criteria.

Additional Criteria

Additional criteria were suggested by some of the respondents. One suggestion is to include a criterion focusing on the availability of a central site for student work (such as provided by Unix or other servers). The authors decided that it was less a language consideration and more of a server availability issue. Another comment suggested the addition of "Methodology or Paradigm" as a criterion. This subsumes the existing criterion "Object-oriented support" and replaced it in the modified list of criteria. Likewise, the criterion "Real or Customized" was also suggested and added to the modified criteria list.

Conclusion

In practice, the choice of a programming language for introductory courses often requires a compromise. There are economic, political, and pedagogical factors that must be considered in the decision making process. While the importance of each of these factors may depend on the specific aims and priorities of the institution, educator, or course, educators must be certain that none of the factors in the above criteria are neglected or sacrificed to more highly visible concerns (McIver & Conway, 1996).

The objectives of this research are to: i) identify criteria for faculty use when selecting a computer programming language for an introductory course in computer programming; ii) assign weights to each of those selection criterion to determine their relative importance in the selection process, and; iii) score various computer programming languages according to those selection criteria. As previously noted, an exhaustive set of selection criteria have been culled from an extensive, if not somewhat disjointed, body of literature. This paper first presents that set of criteria for the selection of a programming language for use in an introductory programming course. It then discusses the structure and administration of an informal pilot study that was intended to assess not only the completeness of the criteria but also to gather feedback on the proposed approach for language evaluation and selection. Using feedback from the pilot study as a basis, the paper then explains how the model was refined as a result of the pilot study.

Constructing an exhaustive set of evaluation criteria and using these criteria in a structured manner provides a means of eliminating much of the subjectivity in the selection process. In addition, the approach presented is extensible. As new programming paradigms and languages are introduced and old ones fall out of favor, the criteria and associated process may easily be revised. The objectivity and extensibility of this formal method yields the replicability missing from existing approaches.

References

Ad Hoc AP CS Committee (2000). Round 2: Potential principles governing language selection for CS1-CS2. Retrieved July 25, 2005 from http://www.cs.grinnell.edu/~walker/sigcse-ap/99-00-principles.html

Computer Language Shootout (2003). Sieve of Erathostenes. Retrieved March 12, 2005 from http://dada.perl.it/shootout/sieve_allsrc.html

Conway, D. (1993). Criteria and considerations in the selection of a first programming language. *Technical Report 93/192*, Department of Computer Science, Monash University.

Cunningham, W. (2004). Language comparison framework. *Portland Pattern Repository*, November 29. Retrieved July 14, 2005 from http://c2.com/cgi/wiki?LanguageComparisonFramework

de Raadt, M., Watson, R., & Toleman, M. (2003). Introductory programming languages at Australian universities at the beginning of the twenty first century. *Journal of Research and Practice in Information Technology, 35* (3), 163-167.

Dijkstra, E. (1972). The humble programmer. *Communications of the ACM, 15* (10), 859-866.

Eisenstadt, M., & Lewis, M.W. (1992). Errors in an interactive programming environment: Causes and cures. In M. Eisenstadt, M.T. Keane, & T. Rajan (Eds.), *Novice programming environments: Explorations in human-computer interaction and artificial intelligence* (Chapter 5). Hillsdale, NJ: Lawrence Erlbaum Associates. Retrieved July 6, 2005 from http://citeseer.ist.psu.edu/cache/papers/cs/3586/http:zSzzSzkmi.open.ac.ukzSzmarczSzpaperszSzBookCh5.pdf/errors-in-an-interactive.pdf

Emigh, K L. (2001). The impact of new programming languages on university curriculum. *Proceedings of ISECON 2001, Cincinnati, Ohio, 18*, 1146-1151. Retrieved July 10, 2005 from http://isedj.org/isecon/2001/16c/ISECON.2001.Emigh.pdf

Howatt, J. W. (1995). A project-based approach to programming language evaluation. *ACM SIGPLAN Notices, 30* (7), 37-40. Retrieved April 11, 2002 from http://academic.luther.edu/~howaja01/v/lang.pdf

Howland, J.E. (1997). It's all in the language: Yet another look at the choice of programming language for teaching computer science. *Journal of Computing in Small Colleges, 12* (4), 58-74. Retrieved June 8, 2005 from http://www.cs.trinity.edu/~jhowland/ccsc97/ccsc97/

Jensen, C. (2004). Choosing a language for .NET development. *Borland Developer Network*. Retrieved July 2, 2005 from http://bdn.borland.com/article/0,1410,31849,00.html

King, K.N. (1992). The evolution of the programming languages course. *ACM SIGCSE Bulletin*, *24* (1), 213-219.

Kölling, M., Koch, B., & Rosenberg, J. (1995). Requirements for a first year object oriented teaching language. *ACM SIGCSE Bulletin, 27* (1) 173-177.

Lee, P.A., & Stroud, R.J. (1996). C++ as an introductory programming language. In M. Woodman (Ed.), *Programming Language Choice: Practice and Experience* (pp. 63-82). London: International Thomson Computer Press. Retrieved June 8, 2005 from http://www.cs.ncl.ac.uk/old/publications/books/apprentice/InstructorsManual/C++_Choice.html

McIver, L. (2002). Evaluating languages and environments for novice programmers. *Proceedings of the Fourteenth Annual Meeting of the Psychology of Programming Interest Group*, London, UK, 100-110. Retrieved September 20, 2005 from http://www.ppig.org/papers/14th-mciver.pdf

McIver, L., & Conway, D.M. (1996). Seven deadly sins of introductory programming language design. *Proceedings of Software Engineering: Education and Practice* (309-316). Los Alamitos, CA, USA: IEEE Computing Society Press.

Parker, K.R., Ottaway, T.A., & Chao, J.T. (2006). Criteria for the selection of a programming language for introductory courses. *International Journal of Knowledge and Learning, 2* (1/2), 119-139.

Riehle, R. (2003). SEPR and programming language selection. *CrossTalk - The Journal of Defense Software Engineering, 16* (2), 13-17. Retrieved October 4, 2005 from http://www.stsc.hill.af.mil/crosstalk/2003/02/Riehle.html

Roberts, E. (2004). Resources to support the use of java in introductory computer science. *ACM SIGCSE Bulletin, 36* (1), 233-234.

Saaty, T. L. (1980). *The analytic hierarchy process*. New York: McGraw-Hill.

Schneider, G.M. (1978) The introductory programming course in computer science: Ten principles. ACM *SIGCSE Bulletin, 10* (1), 107-114.

Sharp, R. (2002). Programming language lifecycles–Where's Java At? *Software Reality.* Retrieved September 20, 2005 from http://www.softwarereality.com/programming/language_lifecycles.jsp

Smolarski, D.C. (2003). A first course in computer science: Languages and goals. *Teaching Mathematics and Computer Science, 1* (1), 137-152. Retrieved November 10, 2005 from http://math.scu.edu/~dsmolars/smolar-e.pdf

Tharp, A.L. (1982). Selecting the 'right' programming language. *ACM SIGCSE Bulletin, 14* (1), 151-155.

Traxler, J. (1994). Teaching programming languages and paradigms. *2nd All-Ireland Conference on the Teaching of Computing*, Dublin, Ireland. Retrieved November 19, 2005 from http://www.ulst.ac.uk/cticomp/traxler.html

Vihakapirom, P. &. Li, K.Y.R. (2003). A framework for distributed group multi-criteria decision support systems. *AUSWEB Conference 2003*, Queensland, Australia. Retrieved January 12, 2006 from http://ausweb.scu.edu.au/aw03/papers/li_______/paper.html

Appendix A: Language Selection Criteria Survey Form

Criteria	**Weight (0 to 10)**
Reasonable financial cost for setting up the teaching environment	
Availability of student/academic version (if cost > 0)	
Availability of textbooks	
Language's stage in life cycle	
System requirements of student/academic/full version	
Operating system dependence	
Open source (versus proprietary)	
Academic acceptance	
Industry acceptance	
Marketability (Regional & National) of graduates	
Easy-to-use development environment	
Ease of learning basic concepts	
Supports target application domain (such as scientific or business)	
Full-featured language (versus scripting)	
Supports teaching approach (functions first or objects first)	
Object-oriented support	
Good debugging facilities	
Support of Web development	
Supports safe programming	
Advanced features for subsequent programming courses	
Availability of support	
Training required for instructors and support staff	
Anticipated programming experience level for incoming students	

Appendix B: Language Evaluation Form

Criterion	**C++ (g++)**	**Java (Eclipse)**	**C++ (.Net)**	**VB (.Net)**	**C# (.Net)**	**JavaScript (Web Browser)**	**Python (Eclipse)**
Reasonable financial cost for setting up the teaching environment							
Availability of student/academic version (if cost > 0)							
Availability of texts							
Language's stage in life cycle							
System requirements of student/academic/full version							
Operating system dependence							
Open source (versus proprietary)							
Academic acceptance							
Industry acceptance							
Marketability of graduates							
Easy-to-use development environment							
Ease of learning basic concepts							
Supports target application domain (such as scientific or business)							
Scripting or full-featured language							
Supports teaching approach							
Object-oriented support							
Good debugging facilities							
Support of Web development							
Supports safe programming							
Advanced features for subsequent programming courses							
Availability of support							
Training required for instructors and support staff							
Anticipated programming experience level for incoming students							

Appendix C: AHP Example

Taking our first three high order selection criteria (in no particular order) let us suppose a respondent has indicated that academic acceptance is twice as important as software cost, academic version is three times as important as software cost, and academic acceptance is four times as important as academic version. The inconsistencies in the respondent's scoring are not unusual and can easily be accommodated by the model. These responses are initially coded in a matrix as:

	Software cost	Academic acceptance	Academic version
Software cost	1	½	3
Academic acceptance		1	4
Academic version			1

Note that when comparing criterion A with B, if B is preferred to A then it is simply coded using the reciprocal. We then complete the matrix as by filling in the remaining cells:

	Software cost	Academic acceptance	Academic version
Software cost	1	½	3
Academic acceptance	2	1	4
Academic version	⅓	¼	1

Next we convert each cell to its decimal value:

	Software cost	Academic acceptance	Academic version
Software cost	1.00	0.50	3.00
Academic acceptance	2.00	1.00	4.00
Academic version	0.33	0.25	1.00

What remains is to calculate an Eigenvector. This is accomplished by first squaring the matrix. The resulting matrix is:

	Software cost	Academic acceptance	Academic version
Software cost	3.00	1.75	8.00
Academic acceptance	5.33	3.00	14.00
Academic version	1.17	0.67	3.00

Now each row is summed and normalized to yield the final Eigenvector. The Eigenvector holds the weights representing the relative importance of each selection criteria. First each row is summed, then the row sums are totaled, and finally the row sums are divided by the total to normalize the values. The results are shown below:

		Sum	Normalization	Results
Software cost	3.00 + 1.75 + 8.00	12.75	12.75 / 39.92	0.32
Academic acceptance	5.33 + 3.00 + 14.00	22.33	22.33 / 39.92	0.56
Academic version	1.17 + 0.67 + 3.00	4.84	4.84 / 39.92	0.12
	Total	39.92		1.00

In this simple example it is apparent that academic acceptance is associated with the highest weight at 0.56, so it is the most important selection criterion, followed by software cost at 0.32, and the least important selection criterion, academic version, with a weight of 0.12.

Appendix D: Refined Language Selection Criteria Survey Form

	Language Accep-tance in Academia	Language Industry Penetration	Software Charac-teristics	Student-Friendly Features	Pedagogical Fea-tures	Language Intent	Language Design	Paradigm	Support and Re-quired Training	Student Experience
Software Cost										
Language Acceptance in Academia	▨									
Language Industry Pene-tration	▨	▨								
Software Characteristics	▨	▨	▨							
Student-Friendly Fea-tures	▨	▨	▨	▨						
Pedagogical Features	▨	▨	▨	▨	▨					
Language Intent	▨	▨	▨	▨	▨	▨				
Language Design	▨	▨	▨	▨	▨	▨	▨			
Paradigm	▨	▨	▨	▨	▨	▨	▨	▨		
Support and Required Training	▨	▨	▨	▨	▨	▨	▨	▨	▨	

Appendix E: Refined Language Evaluation Form[*]

	Language 1	Language 2	Language 3
Software Cost			
Programming Language Acceptance in Academia			
Programming Language Industry Penetration			
Software Characteristics			
Student-Friendly Features			
Language Pedagogical Features			
Language Intent			

Language Design			
Language Paradigm			
Language Support and Required Training			
Student Experience			

* Evaluators are restricted to assessing only languages with which they are quite familiar.

Biographies

Dr. **Kevin R. Parker** is a Professor of Computer Information Systems at Idaho State University, having previously held an academic appointment at Saint Louis University. He has taught both computer science and information systems courses over the course of his fifteen years in academia. Dr. Parker's research interests include e-commerce marketing, competitive intelligence, knowledge management, the Semantic Web, and information assurance. He has published in such journals as *Journal of Information Technology Education*, *Journal of Information Systems Education*, and *Communications of the AIS*. Dr. Parker's teaching interests include web development technologies, programming languages, data structures, and database management systems. Dr. Parker holds a B.A. in Computer Science from the University of Texas at Austin (1982), an M.S. in Computer Science from Texas Tech University (1991), and a Ph.D. in Management Information Systems from Texas Tech University (1995).

Dr. **Joseph T. Chao** is an Assistant Professor of Computer Science at Bowling Green State University. Dr. Chao has seven years of industry experience in software development, including three years as Director of Software Development prior to entering academia. His research focus is on software engineering with special interests in programming languages, object-oriented analysis and design, and agile software development. He has published in such journals as *International Journal of Knowledge and Learning*, *Academe: Bulletin of the American Association of University Professors*, and *Journal of Manufacturing Systems*. He has taught courses in all aspects of the software development lifecycle including programming, systems analysis and design, database systems, usability engineering, software engineering, and agile software development. Dr. Chao holds an M.S. in Operations Research from Case Western Reserve University and a Ph.D. in Industrial and Systems Engineering from The Ohio State University.

Dr. **Thomas A. Ottaway** is an Associate Professor in the Computer Information Systems Department at Idaho State University, having previously held academic appointments at Kansas State University and the University of Montana. He holds a B.S. in Computer Science from Wichita State University (1991), a M.S. in Management Information Systems from Texas Tech University (1993), and a Ph.D. in Production/Operations Management from Texas Tech University (1995). He has published in such journals as *Decision Sciences*, *International Journal of Production Research*, *Journal of Economics and Finance* and the *Journal of Financial and Economic Practice*. Dr. Ottaway's teaching interests include data and telecommunications networks as well as computer programming.

Dr. **Jane Chang** is currently an Assistant Professor of Applied Statistics and Operations Research at Bowling Green State University, having previously held a tenured position at Idaho State University. Dr. Chang's research interests include optimal experimental design, data analysis, and statistical design and analysis of microarray experiments. She has published several monographs and in such journals as *Utilitas Mathematica*, *Journal of Statistical Planning and Inference*, *Statistica Sinica*, *International Journal of Heat and Mass Transfer*, and *International Journal of Mechanical Engineering Education*. Dr Chang's teaching interests include Experimental Design, Statistics for Managerial Decision Making, Response Surface Methodology, Multivariate Analysis, Regression Analysis, Probability, Mathematical Statistics, and Business Statistics. Dr. Chang holds a B.A. in Applied Statistics from Chung-Yuan University and a Ph.D. in Statistics from The Ohio State University.