Educ Inf Technol (2010) 15:255–275
DOI 10.1007/s10639-010-9126-8

Selecting software tools for IS/IT curricula

Kevin R. Parker

Published online: 20 April 2010
© Springer Science+Business Media, LLC 2010

Abstract The evaluation and selection of software tools for use in an IS or IT curriculum is difficult not only because actual industry software tools are often used but also because there is no formal approach to guide the process. How does one choose between SQL Server and MySQL, or Dreamweaver and Expression Studio? IS and IT educators must periodically go through the process of assessing the most suitable tools for their courses. Given how common such decisions are and how frequently they must be made, it is surprising to find that there is a lack of literature that deals with comparative studies of software tools for higher education. This paper proposes a set of criteria for the selection of software tools for IS and IT programs, explains how multi-criteria decision analysis can be used to weight those criteria, and details an approach for the application of those weighted criteria. The proposed approach is structured and replicable, and allows for a more thorough evaluation of the available options and a more easily supportable selection.

Keywords Software tool selection · IS/IT software tools · Software evaluation · Educational software · Multi-criteria decision analysis · Analytic hierarchy process

1 Introduction

The evaluation and selection of software for use in the classroom is seldom an easy undertaking. This has traditionally been even more difficult in an information systems (IS) or information technology (IT) curriculum in which actual industry software tools are often used. The issue of determining the most appropriate software tools for instructional purposes is not new, but as the tools keep changing and evolving, options may also change (Beise 2006).

K. R. Parker (✉)
Idaho State University, 921 S. 8th Ave., Stop 8020, Pocatello, ID 83209-8020, USA
e-mail: parkerkr@isu.edu

To paraphrase Denton and Peace (2004), IS and IT instructors are constantly trying to determine how best to provide meaningful hands-on experiences for students. Ideally, software would be readily available for students to perform homework exercises, practice and reinforce course concepts, and develop functional projects. Software tool use in a course should simulate as closely as possible an enterprise experience. However, given the complexity and expense of most enterprise-wide systems and the limited free time that most instructors have to learn, install, implement, and teach such systems, careful consideration is required to select software that will offer the most valuable educational experience to students. Both students and instructors should be able to focus on essential course concepts rather than struggling with details of a complex technical software product.

The state of information technology itself contributes to the difficulties in software evaluation and selection because there is often a large variety of candidate software packages, technology evolves rapidly and continuously, evaluation paradigms are inadequate, software packages may be available on a variety of platforms, and many individuals involved in the decision-making process lack sufficient technical backgrounds. Decisions are often based on individual agendas or superficial familiarity with a package rather than on a formal evaluation that allows software to be analyzed in a consistent, structured, and replicable manner (Huber and Giuse 1995).

There are many variables that must be considered when selecting such software. How does one go about choosing between SQL Server and MySQL, or assessing the pros and cons of Dreamweaver versus Expression Web, or deciding between Visible Analyst and ERwin, or selecting Microsoft's Project 2007 versus Rally? The list goes on and on, and IS and IT educators must periodically go through the process of selecting the most suitable tools for their courses. A formal process is needed because the selection and evaluation of software tools must be done in a consistent, quantifiable manner to be effective. By using a formal method, the justification for the selection decision is not just based on technical, intuitive, or political factors (Bandor 2006).

Given how common such decisions are and how frequently they must be made, one would expect numerous narrowly-focused studies on the topic. However, software evaluation criteria are not clearly defined and elaborated in the literature (Jadhav and Sonar 2009). A literature search revealed very few manuscripts directly related to this decision-making process. Beise (2006) confirms that there is little previous literature that deals with comparative studies of software tools for higher education. Jadhav and Sonar (2009) also observe that little work has been done on establishing a generic methodology that can be used for the selection of software packages of any type. They go on to state that there is lack of software evaluation criteria, and therefore there is need to develop a framework comprised of a software selection methodology, an evaluation technique, an evaluation criteria, and a system to help decision makers in the software selection process (Jadhav and Sonar 2009).

This apparent dearth of relevant research in software tool selection for IS and IT programs prompted related alternatives to be considered. In 2006 Parker, Ottaway, and Chao published two papers (Parker et al. 2006a, b) that proposed and then refined a structured approach for the selection of an appropriate language for programming courses, stating that it "would enable a more thorough evaluation of

the available options and a more easily supportable selection." This paper will attempt to adapt those criteria to the selection of software tools for use in an IS or IT curriculum.

2 Related work

Before reviewing previous research on software selection it first must be pointed out that "the terms evaluation, review, and selection are all used in the literature on assessment of educational software, though different writers use some or all of these words with slightly different meanings" (Squires and McDougall 1994, p. 3).

The search for literature dealing with the selection of software tools for IS/IT degree programs started with a series of ill-conceived searches that included the phrase "educational software." While such a search returns a large result set, it was soon discovered that "educational software" is generally used to refer to software for primary school children. Further, this fact is often obscured since many if not most of the "educational software" studies fail to indicate exactly what type of software they are proposing to evaluate.

2.1 Software tool selection for higher education

Once it became obvious that research focusing on the evaluation of educational software was not entirely germane to our purposes, the search turned toward "software tool selection." The resulting papers cover a wide range of topics somewhat related to software selection for higher education.

Huber and Giuse (1995) present an educational software evaluation process to facilitate timely recommendations for product acquisitions. The process is structured in such a way that both content and technical experts are involved in assessing coverage of breadth and depth of subject, clarity of presentation, quality of construction, and ease of use. The process employs a bi-level evaluation instrument, which required the construction of two evaluation forms to support each level of the assessment. The first form is concerned primarily with software content and ease of use. The second form consists of a series of open-ended comment sections concerning intended users, content, system requirements, support, usability, performance, customizability, and documentation.

Kao et al. (2000) propose a paradigm for selecting institutional software. While they never explain exactly what institutional software is, their example institution is a university. The proposed paradigm begins with finding a suitable framework for guiding the integration of the software into the institution and customizing it to suit the institution's needs. Institutional guidelines and policies relevant to selecting proper software must be taken into account, as must specific evaluation criteria. Their example demonstrates the synthesis of several frameworks into a new framework for selecting on-line courseware for an educational institution.

Both the study by Beise (2006) and that by Denton and Peace (2004) deal with database course tool selection. The software selected for use in database courses can generally be categorized as one of three types: commercial enterprise software, such as Oracle or IBM's DB2; personal database software, such as Microsoft Access; or

software available for no cost (including open source software), such as PostgreSQL or MySQL (Denton and Peace 2004). Beise (2006) derives several criteria that determine the choice of software, including cost, technical support, supporting resources, industry relevance, and accessibility. She points out that the results of her analysis are consistent with that of Denton and Peace (2004), whose criteria included cost, ease of use, security, functionality, job market appeal, and compatibility. A discussion of the individual criterion will be deferred.

Lancor (2008) describes the experiences of selecting a collaboration tool for a software engineering course that requires group work throughout the semester. Her selection process was less structured than the studies previously discussed, focusing on type (project management or collaboration tool type), pricing, installation requirements, and other common features. Her focus was more on the tool selected rather than the selection process for the tool.

2.2 Software selection for non-educational entities

After finding limited literature on the selection of software tools for higher education, it was decided to review the studies that pertain to the process by which commercial entities select software tools. This is a reasonable progression since the software used in IS and IT programs is most often professional-grade software. Fritz and Carter (1994) performed a study to assist an organization that provides advice to clients considering applications software purchases, and as such they identify, characterize, classify, and describe eleven distinct software evaluation and selection methodologies. Jadhav and Sonar (2009) provide an even more thorough review of the literature in the field of evaluation and selection of software packages.

Other studies focus on the various steps required in the selection and procurement of commercial off-the-shelf (COTS) application software for a specific business, including Requests for Proposals and Requests for Bids. Many organizations utilize component libraries or extensions, or complete COTS-based solutions such as ERP applications (Bandor 2006). A sampling of studies devoted to the COTS decision includes (Carney and Wallnau 1998; Dean and Vigder 2000; Franch and Carvallo 2003; Kontio 1996; Kontio et al. 1995; Lin et al. 2006; Torchiano et al. 2002). Most of these papers propose a new systematic process for evaluating and ranking COTS software, or discuss existing practices. Related research examines the selection of applications software or software technology, and includes (Anderson 1990; Filho et al. 2005; Obeidat 2006; Rowley 1992; van Staaden and Lubbe 2006). Many of these studies present criteria to be used in the selection process, and these will be discussed later.

Some studies are more narrowly focused, examining the selection of a particular type of software. Filho et al. (2005) focus on the selection of decision analysis software. Min (1992) looks at the purchase of logistics software. Bosilj-Vuksic et al. (2007), Davis and Williams (1994), and Nikoukaran et al. (1998) consider the process of simulation software package selection by businesses. Literature by Baki and Çakar (2005), Güngör ŞEN and Baraçli (2006), and Teltumbde (2000) provide a small sample of the studies that deal with ERP selection. Donham (2004), Hauge et al. (2009), and Polančič et al. (2004) provide examples of research in evaluating open source products based on multiple criteria of software quality. Kannan and Vinay (2008) discuss the selection of CAD/CAM systems. Lai et al.

(1999) examine the selection of multimedia authoring systems. Ngai and Chan (2005) provide a framework for the evaluation of knowledge management tools.

2.3 Educational software selection

As noted earlier, educational software research is only tangentially related to this study, but it seems advisable to make a rapid assessment of some of the studies.

Although educational software has been defined as "any software that is used in an educational context, whether or not it was specifically designed for educational use" (Squires and McDougall 1994, p.2), that is rather misleading. Educational software generally seems to "be of five basic types: tutorial, drill and practice, simulation/game, information, and management and assessment" (Ahmed 2003, p.2). Plaza et al. (2009) provide a thorough and excellent literature review regarding evaluation of educational software. Some of these studies propose criteria that seem appropriate when evaluating many types of software tool.

While checklists are commonly used in selecting educational software, several studies note serious problems with the checklist approach, including a focus on technical rather than educational issues, unknown validity of criteria, shortcomings for assessing quality and instructional efficacy, lack of tailored criteria, and a reliability on past usability evaluations that makes them inadequate for evaluating new and innovative user interfaces (Bednarik et al. 2004; Hosie et al. 2005; McDougall and Squires 1995; Squires and McDougall 1994; Squires and Preece 1996; Squires and Preece 1999; Tergan 1998).

Not everyone is as critical of checklists. Susser (2001) examines the main objections to the use of checklists one by one, explaining either that the criticism is unjustified or that it applies equally to any form of courseware evaluation. The usage of checklists brings an ease of performing, collecting, processing, and maintaining evaluations (Bednarik et al. 2004). In most cases generic checklists are adequate enough, since creating domain specific checklists is a time-consuming and expensive task that is often left undone (Gerdt et al. 2002).

Some studies go beyond pointing out inadequacies with existing evaluation approaches and propose new approaches. Most of these approaches claim to be comprehensive in scope, taking into account not only appearance and functionality, but also features for managing instruction and assessing learners (Ahmed 2003; Bednarik et al. 2004; Gerdt et al. 2002; Hosie et al. 2005; Lê and Lê 2007; Plaza et al. 2009; Squires and McDougall 1994; Squires and McDougall 1996; Squires and Preece 1996; Squires and Preece 1999; Tergan 1998; Voogt 1990).

2.4 Programming language selection

As noted earlier, none of the above studies provides exactly what is being sought: an easily adapted selection process for software tools for IS and IT curricula. However, selecting a programming language for introductory programming courses is a chore faced by both IS and IT programs, and studies dealing with that selection process propose approaches that appear to be both pertinent and adaptable to the selection of other types of software tools. Papers by Parker et al. (2006a, b) and Mannila and de Raadt (2006) seem to hold the most promise.

3 An overview of the programming language selection approach

The selection criteria proposed by Parker et al. (2006a, b) and that attributed to Mannila and de Raadt (2006) were examined. Mannila and de Raadt's intent was to compare specific introductory programming languages, and they advance a list of criteria by which that comparison can be made. An assessment of Mannila and de Raadt's criteria compared to the Parker et al. (2006a, b) criteria was made, and it was determined that the features of Mannila and de Raadt's criteria that could be adapted to the selection of software tools were subsumed by the criteria offered by Parker et al. (2006a, b).

Parker et al. (2006a, b) point out that a structured approach for selecting programming languages is needed because too often it has been handled through an informal process involving faculty evaluation, discussion, and consensus. Such an approach lacks structure and replicability. They propose an instrument and a methodology that will allow the process to be more easily repeated in the future.

A set of evaluation criteria was proposed in Parker et al. (2006a) and subsequently refined in Parker et al. (2006b). The criteria are based on over sixty studies relevant to language selection and justified by a brief review of the supporting literature in Parker et al. (2006a, b). Each of the criteria has been suggested in one or more previous studies that evaluate programming languages. Table 1 first appeared in Parker, et al. (2006b) and presents the criterion groupings.

4 Adaptations necessary to assess software tools

An examination of the criteria presented above indicates that many of them can be adapted for use in the assessment and selection of software tools. Some criteria are not applicable and will be dropped from consideration, and the remaining criteria will be regrouped. Some of the remaining criteria will be renamed and their scope will be expanded. Those being eliminated are

- debugging facilities,
- support for safe programming,
- real or customized,
- support for target application domain,
- prior experience of incoming students.

Some of the criteria groupings were also altered:

- The elimination of the criteria related to "debugging facilities" and "support for safe programming" lead to the combination of "Student-Friendly Features" and "Language Pedagogical Features" into the grouping "Pedagogical Features."
- The elimination of the criteria that refer to "real or customized" and "support for target application domain" made it necessary to eliminate the grouping "Language Design."
- Since it was the sole member of the group, the elimination of the "prior student experience" criterion made it necessary to eliminate the "Student Experience" group as well.

Table 1 Criteria for the selection of a programming language

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

- The criterion "full-featured language" was relabeled "Feature Set" to make it applicable to software tools. It was then grouped with "Software Characteristics" since that is a more logical fit.
- At that point the group labeled "Intent" contained only one criterion, so the decision was made to combine it with "Paradigm" under the heading "Course Methodology and Software Paradigm."

Note that each of the criterion adaptations is suggested elsewhere in software selection literature. Programming courses will not be included in the following discussion since those are covered extensively in Parker et al. (2006a, b).

4.1 Category 1: Software cost

The software cost category includes two criterion, financial cost and availability of an academic version.

Reasonable financial cost refers to the price to acquire the software tool. This may involve individual packages or a site license for a network version. Factors to explore include academic discounts for educational institutions, alliances in which the university or department can enroll, or the availability of a free, downloadable trial version. For example, Expression Studio is free to students if their university is part of the Microsoft Developer Network Academic Alliance. Dreamweaver, on the other hand, is available for a thirty day trial from Adobe, after which it can be purchased at an educational price of approximately $200. Software cost is included in the criteria of several software selection studies, including (Baki and Çakar 2005; Beise 2006; Bosilj-Vuksic et al. 2007; Denton and Peace 2004; Kao et al. 2000; Min 1992; Rowley 1992; Torchiano et al. 2002).

The availability of an affordable version of the software tool allows students to install the development environment on their personal machine, making it convenient for them to work on their assignments even when the computer lab is not accessible. If a student version is unavailable and the department uses a network-based version, then students may have to work on assignments in campus labs, restricted by hours of operation, availability of transportation, etc. If the academic version has a limited feature set, then the benefit to the students may not be as great, but this factor should at least be considered. Beise (2006) discusses this criterion under the heading "accessibility for students."

4.2 Category 2: Acceptance in academia

This category refers to the degree to which the software tool under consideration has been embraced by academia.

Academic acceptance refers to the popularity of a software package across academic institutions. This can be assessed by current use or projected use at other institutions. This criterion is included in Donham's (2004) discussion of community opinion.

The availability of text books is affected by many factors. The maturity of the software tool impacts the availability of textbooks, particularly when the tool is relatively new. It is often difficult to find a quality textbook for a newly released tool, but as a tool matures more texts become available. The academic acceptance of a software tool 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 tool. Beise (2006) includes textbook availability under availability of teaching and learning materials.

4.3 Category 3: Industry penetration

This category refers to the degree to which the software tool under consideration has been embraced by the professional community.

A software tool's maturity affects not only text-book availability, as noted above, but it may also impact the widespread use of a tool in both industry and academia. This has been variously referred to as pedigree, maturity, lifetime, and product maturity (Bosilj-Vuksic et al. 2007; Donham 2004; Rowley 1992; Torchiano et al. 2002).

Industry acceptance refers to the market penetration of a particular software tool in industry, i.e., the use of a tool 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 job openings that require familiarity with the tool. In software selection studies industry acceptance falls within the scope of industry relevance, pedigree, or community opinion (Beise 2006; Bosilj-Vuksic et al. 2007; Donham 2004).

Marketability refers to the employability of graduates at the local, national, and international levels. Software tool selection is often driven by demand in the workplace, i.e., what employers actually use. Denton and Peace (2004) refer to this factor as job market appeal.

4.4 Category 4: Software characteristics

The software characteristics category encompasses various software traits like system requirements, operating system limitations, the source availability model, and software feature set.

The system requirements of a software package, whether it is a student, academic, or full version, include 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. In other software selection studies the system requirements criterion has been referred to as technological aspects, technology, technical considerations, or hardware requirements (Bednarik et al. 2004; Kao et al. 2000; Rowley 1992; Torchiano et al. 2002)

Operating system dependence refers to the dependence of a tool on a particular operating system platform. For example, some software tools are dependent on the Windows operating system, while others are platform independent with environments available for a variety of operating systems. The software selection literature incorporates operating system dependence into criteria like technological aspects, compatibility, technical considerations, and software requirements (Bednarik et al. 2004; Denton and Peace 2004; Min 1992; Rowley 1992; Torchiano et al. 2002).

Open source versus proprietary refers to the entity that controls the evolution of the software tool. For example, Microsoft is responsible for additions, deletions, or modifications to SQL Server. On the opposite end of the spectrum, MySQL is an open source database (in spite of its recent acquisition by Oracle) and can be modified by any member of the open source community. Donham (2004), Hauge et al. (2009), and Polančič et al. (2004) focus exclusively on the evaluation of open source solutions.

One software characteristic that should be considered is the feature set. A full feature set versus limited feature set refers to the fact that some software tools lack certain features. That may be due to the fact that some software tools are more capable than others. For example, Microsoft Access does not support stored procedures or triggers, while those are a feature of SQL Server, Oracle, MySQL, etc. If employers expect students to know how to create stored procedures and triggers,

then the software tool selection process should take that into account and rate Access poorly for this criterion. In other cases an educational (or evaluation) version of a software tool with a limited feature set may have been developed especially for teaching or assessment purposes. For example, some modeling tools like ERwin offer evaluation versions that limit the number of entities that can be modeled. Lancor (2008) notes a problem stemming from a limitation on the number of modules, and Min (1992) lists among her criteria the size and scope of the problem to be solved by the software.

4.5 Category 5: Pedagogical features

This category considers a language's learning curve, ease-of-use features associated with the software tool, advanced features that could support subsequent courses, and whether the software tool supports collaboration.

Ease of learning concepts refers to the learning curve associated with a software tool. Bednarik et al. (2004) incorporate this into their usability parameter, while Beise (2006) calls it "ease of use/learning curve" and Huber and Giuse (1995) address it in "how easy to use."

An easy-to-use environment is critical in teaching. This includes the interface design, that is, the overall look and feel of the tool including how intuitive it is to the user (Kao et al. 2000). Denton and Peace (2004) extend ease of use to include uncomplicated software installation, minimal maintenance requirements, easy access from remote locations, and straightforward administration and use of the software by both faculty and student perspectives, because supporting the platform can require extra time and effort on the part of the individual instructor. These considerations are also addressed by (Anderson 1990; Bednarik et al. 2004; Huber and Giuse 1995; Min 1992; Rowley 1992; Torchiano et al. 2002).

Advanced features for subsequent courses is critical if multiple courses, such as Database Systems and Advanced Database Concepts, are included in a computing curriculum. If this is the case then the candidate tools must offer adequate advanced features to support an advanced course. Anderson (1990) refers to this attribute as advanced functions, while Min (1992) refers to it as sophistication.

One criterion that was not included in Parker et al. (2006a, b) is whether a software tool supports collaboration. This has become more critical in recent years, especially to employers. Lancor (2008) states that the ability for students to function effectively on teams to accomplish a common goal is critical. She goes on to point out that ABET emphasizes teamwork skills through its revised curricula criteria that includes learning outcomes that embrace the development of effective teamwork skills (ABET 2008).

4.6 Category 6: Course methodology and software paradigm

This category focuses on how the software will be used in the educational environment. This includes both the teaching goals and teaching approach that is preferred, as well as the software paradigm.

If one of the goals of a course or curriculum is to teach web-based development, the level of support for web development associated with a particular tool must be

considered. Beise (2006) refers to this as a web development interface. Technology bundles like LAMP (Linux, Apache, MySQL, PHP) come to mind when considering this category. And of course software tools like Dreamweaver, Expression Web, etc., are intended for the development of web pages and sites.

Support for teaching approach refers to how well a software package supports the teaching approach preferred by the faculty, e.g., whether the intent is to teach traditional analysis and design and database approaches or to move toward object-oriented or agile approaches. This will affect whether the course covers data flow diagrams (DFDs) and entity-relationship diagrams (ERDs), or go with the Unified Modeling Language (UML) or use case diagramming tools. Denton and Peace (2004) refer to this as functionality, explaining that the "software system chosen must provide the functionality needed to achieve the pedagogical goals of the course" (p. 404).

This leads to a discussion of software paradigm, which is inextricably linked to teaching approach. As an example of software paradigm, consider tools for systems analysis and design or database design. Such tools may support a traditional approach to design, while others may support the object-oriented paradigm, while still others may be designed to support agile development techniques. For example, a course that focuses on agile development techniques may be better served by a project management tool designed to support agile approaches, like Rally, rather than a general purpose tool like Microsoft Project. The teaching approach makes it necessary to select the paradigm that will be the focus of the course, and similarly the paradigm dictates the teaching approach.

4.7 Category 7: Support and required training

This category looks at the level of and need for documentation, training, and support for both instructors and students.

Availability of documentation and support takes into account the availability of support staff, including computer lab staff and/or network administrators, to support the teaching and administration of a software tool. Beise (2006) notes that some packages require institutional technical support, especially if installed on a campus server linked to a campus network and subject to security constraints, which can limit or prevent access from off-campus, and potentially conflict with the software configuration. This criterion should also take into account the availability of support through forums or listservs on the Internet, as well as vendor support. This is broadly addressed in the literature on software selection in (Anderson 1990; Baki et al. 2005; Beise 2006; Bosilj-Vuksic et al. 2007; Donham 2004; Huber and Giuse 1995; Kao et al. 2000; Min 1992; Rowley 1992; Torchiano et al. 2002).

Training includes not only the training required for instructors and support staff, but also the time needed to learn a software tool and the availability of qualified instructors to teach a course using that tool. For example, a few years ago it was common to have IS professors who were familiar with DFDs and ERDs, but had little or no knowledge of UML and would require training if UML software tools like Rational Rose were adopted. Anderson (1990), Beise (2006), Kao et al. (2000), and Min (1992) all mention the need for training services.

Table 2 shows the modifications made to the content of Table 1, resulting in a set of criteria for the selection of software tools for an IS/IT curriculum.

5 Practical considerations

While the selection of software tools for an IS or IT curriculum is highly subjective, a thorough set of criteria makes an objective selection process attainable. However, the process may still vary drastically at each institution due to the differences in culture, strategy, or even politics. The following steps provide a systematic approach for a general selection process.

1. Compile a list of criteria. The criteria proposed by this study can be adapted to fit the needs of most departments or programs.
2. Weight each of the criteria. Ask each evaluator to weight, specific to the department's needs, the value of importance for each criterion. If there are

Table 2 Criteria for software tool selection for IS/IT curricula

Software cost
• Reasonable financial cost
• Availability of affordable software version
Acceptance in academia
• Academic acceptance
• Availability of textbooks
Industry penetration
• Maturity
• Industry acceptance
• Marketability of graduates
Software characteristics
• System requirements
• Operating system dependence
• Open source (versus proprietary)
• Feature set
Pedagogical features
• Ease of learning basic concepts
• Easy-to-use development environment
• Advanced features for subsequent courses
• Collaboration support
Course methodology and software paradigm
• Support of Web development
• Support for teaching approach
• Software paradigm
Support and Required Training
• Availability of documentation and support
• Training required for instructors and support staff

multiple evaluators, either the evaluators can strive to reach a consensus or the weights assigned by each evaluator can be averaged.

3. Determine a list of candidate tools. The list should be comprised of software tools suggested by the faculty rather than an exhaustive list of available tools. Having sub-lists may be desirable so that a subset of candidate tools can be compared at one time to narrow down the choices, and comparing several similar tools may also be desirable.
4. Rate the software tool. Each candidate tool should be assigned a rating for each criterion. Again, with more than one evaluator a consensus should be reached or average scores could be calculated.
5. Calculate a weighted score. For each candidate tool, a weighted score can be calculated by adding together the tool score multiplied by the weight assigned to each criterion. The software tool with the highest weighted score is the optimal choice based on evaluators' assessments.

The process is fairly mechanical and can be easily adapted to fit the needs of individual departments. It may be advisable to begin the selection process with a brief introduction to the procedure. A software tool selection committee may be formed to evaluate and adapt the selection criteria and to assign a weight to each criterion for the department.

Lai et al. (1999) observe that many approaches are available to accommodate the disparate judgments of group participants in this process:

> In a common objective context where all participants share the same objectives, there are four ways for setting the priorities: consensus; vote or compromise; geometric mean of the individuals' judgment; and separate models or players. Consensus refers to the achievement of consensus of group participants in constructing the hierarchy and making judgments. If consensus cannot be obtained, the group may choose to vote or compromise on a judgment. If a consensus cannot be achieved and the group is unwilling to vote or to compromise, then a geometric mean (average) of the individuals' judgments can be calculated. If a group has significantly different objectives and cannot meet to discuss the decision, then each group member can make judgment separately, based either on separate models or players. If it is based on separate models, then each group member enters their judgment into a separate model, which will then be averaged. However, if it is based on separate players, then a combined model is set up with each 'player' evaluating those factors in their part of the combined model.

Not every faculty member in the department may have expertise in or even familiarity with all software tools to be evaluated. One solution is to request that evaluators download and experiment with an evaluation version if one is available. Another alternative would be to require each evaluator to state their confidence level on each software tool evaluated.

5.1 Structured weighting process

Assigning weighing factors to a broad set of criteria is not an easy task. 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 seven 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 software tools is needed.

Multi-criteria decision analysis (MCDA) can be applied to the software evaluation process (Jadhav and Sonar 2009). In their literature review of software selection methodologies, Güngör ŞEN and Baraçli (2006) point out that over half of methodologies that they examined use multi-criteria decision-making methods to support the final phase of the selection process. MCDA, and more specifically the Analytic Hierarchy Process (AHP), is used in the decision making process proposed by Parker et al. (2006b). Other software selection studies using MCDM or AHP include (Bosilj-Vuksic et al. 2007; Davis and Williams 1994; Kannan and Vinay 2008; Lai et al. 1999; Min 1992; Ngai and Chan 2005; Teltumbde 2000).

MCDA provides a variety of tools for structured decision making. The types of decisions for which MCDA is most appropriate are those that involve selecting from

Table 3 Comparison matrix

	Acceptance in Academia	Industry Penetration	Software Characteristics	Pedagogical Features	Course Methodology & Software Paradigm	Support & Required Training
Software Cost						
Acceptance in Academia						
Industry Penetration						
Software Characteristics						
Pedagogical Features						
Course Methodology & Software Paradigm						
Support & Required Training						

Table 4 Representative scale

Overwhelmingly more important	9:1
Very strongly more important	7:1
Strongly more important	5:1
Moderately more important	3:1
Equally important	1:1
Moderately less important	1:3 (or 1/3)
Strongly less important	1:5 (or 1/5)
Very strongly less important	1:7 (or 1/7)
Overwhelmingly less important	1:9 (or 1/9)

multiple options the single option that most closely meets a set of weighted objectives. In this case, we seek to select the software tool that most closely supports course requirements, where those requirements are expressed by the selection criteria previously defined. AHP is particularly appropriate for the type of analysis required in this research.

The AHP, developed by Saaty (1980), uses a hierarchy to structure a decision problem. It decomposes the problem into its component elements, groups the elements into homogeneous sets, and arranges them hierarchically (Teltumbde 2000). The AHP uses the hierarchical model to provide a method to assign numerical values to subjective judgments on the relative importance of each element, and then to synthesize the judgments to determine which elements have the highest priority.

The method 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.

AHP requires pairwise comparisons of selection criteria. For N selection criteria this will require $(N^2 - N)/2$ comparisons. With seven high-order attributes for the respondents to assess there are only twenty-one pairwise comparisons. This approach also reduces the possibility of overlapping criteria and provides a structured approach to assigning weights to the criteria.

The comparison matrix is shown in Table 3. The row-column intersections are where the pairwise comparisons between criteria are recorded.

The respondent is asked 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 4.

Table 5 Example weights

	Software Cost	Acceptance in Academia	Industry Penetration
Software cost		1/5	1/3
Acceptance in academia			7
Industry penetration			

Table 6 Example weights and reciprocals

	Software Cost	Acceptance in Academia	Industry Penetration
Software cost	1	1/5	1/3
Acceptance in academia	5	1	7
Industry penetration	3	1/7	1

The scale is used for evaluators' responses 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.

Working row-by-row through the first three high order selection criteria, let us suppose a respondent has indicated that academic acceptance is strongly more important (5:1) than software cost, so conversely software cost is strongly less important (1:5) than academic acceptance. Thus, a 1/5 will be entered in the intersection of software cost/academic acceptance. Next they assess industry penetration as being moderately more important (3:1) than software cost, and conversely software cost as moderately less important (1:3) than industry penetration, so a 1/3 will be entered in the intersection of software cost/industry penetration. Finally, they weight academic acceptance as very strongly more important (7:1) than industry penetration, so a 7 will be entered in the intersection of academic acceptance and industry penetration. The slightly inconsistent weighting in the respondent's scoring is not unusual and can easily be accommodated by the model. These responses are initially coded in a matrix as shown in Table 5.

Note that when comparing criterion A with B, if B is preferred to A then it is simply coded using the reciprocal. Therefore we can complete the matrix by filling in the remaining cells as seen in Table 6.

Next we convert each cell to its decimal value (Table 7).

What remains is to calculate an Eigenvector. This is accomplished by first squaring the matrix. The squared matrix appears in Table 8.

Next, each row is summed and normalized to yield the final Eigenvector. The Eigenvector holds the weights representing the relative importance of each selection criteria. Table 9 demonstrates that each row is first summed, then the row sums are totaled, and finally the row sums are divided by the total to normalize the values.

In this simple example it is apparent that academic acceptance is associated with the highest weight at 0.754, so it is the most important selection criterion, followed

Table 7 Example weights as decimal values

	Software Cost	Acceptance in Academia	Industry Penetration
Software cost	1.00	0.20	0.33
Acceptance in academia	5.00	1.00	7.00
Industry penetration	3.00	0.14	1.00

Table 8 Squared matrix

	Software Cost	Acceptance in Academia	Industry Penetration
Software cost	3.00	0.45	2.07
Acceptance in academia	31.00	3.04	15.67
Industry penetration	6.74	0.90	3.04

by industry penetration at 0.162, and the least important selection criterion, software cost, with a weight of 0.084.

5.2 Remaining steps

Recall that five steps were listed for the selection process.

1. Compile a list of criteria.
2. Weight each of the criteria.
3. Determine a list of candidate tools.
4. Rate the software tool.
5. Calculate a weighted score.

At this point we have accounted for steps 1 and 2. As noted, when determining the list of candidate tools (step 3) faculty should suggest specific tools rather than attempting to compile an exhaustive list of available tools. This may require investigation on their part or networking with colleagues at different universities.

Once the list of candidate tools has been aggregated, each of the candidate tools can be rated (step 4). Each tool should be assigned a rating for each criterion based on how well it meets or satisfies the criterion and each of its components. Referring again to the example given above—assessing database packages—if employers have expressed a preference for graduates with experience creating stored procedures and triggers, then Oracle and SQL Server would be assigned a high score for Software Characteristics while Microsoft Access would receive a poor rating for this criterion. This process is performed for each of the criterion for all of the candidate tools. A tool selection criteria survey form, like that shown in Table 10, can be used for step 4.

Once all of the candidate tools have been rated for each criterion, the weighted score can be calculated for each tool by adding together the products of the tool scores multiplied by the weights assigned to each criterion (step 5). The software

Table 9 Decomposition of matrix into eigenvector

		Sum	Normalization	Results
Software cost	3.00+0.45+2.07	5.52	5.52 / 65.91	0.084
Acceptance in academia	31.00+3.04+15.67	49.71	49.71 / 65.91	0.754
Industry penetration	6.74+0.90+3.04	10.68	10.68 / 65.91	0.162
	Total	65.91		1.000

tool with the highest weighted score is the optimal choice based on evaluators' assessments.

6 Conclusion

In practice, the choice of software tools to support a curriculum often requires a compromise. The process of appraising any software investment can become a political process as the decision touches on the diverse interests of many people and groups (Lai et al. 1999). There are economic, political, and pedagogical factors that must be considered in the decision making process. Further, software selection is not a well-defined or structured process. The fact that there are not only multiple decision makers

Table 10 Software tool selection criteria survey form

Criterion	Weight (0 to 10)
Software cost	
• Reasonable financial cost	
• Availability of affordable software version	
Acceptance in academia	
• Academic acceptance	
• Availability of textbooks	
Industry penetration	
• Maturity	
• Industry acceptance	
• Marketability of graduates	
Software characteristics	
• System requirements	
• Operating system dependence	
• Open source (versus proprietary)	
• Feature set	
Pedagogical features	
• Ease of learning basic concepts	
• Easy-to-use development environment	
• Advanced features for subsequent courses	
• Collaboration support	
Course methodology and software paradigm	
• Support of Web development	
• Support for teaching approach	
• Software paradigm	
Support and required training	
• Availability of documentation and support	
• Training required for instructors and support staff	
Weighted Score (total):	

but also multiple criteria that must be considered expands decisions from one to several dimensions, thus increasing the complexity of the selection process (Lai et al. 1999). Educators must take care that none of the criteria are neglected or sacrificed to more highly visible concerns.

The objective of this research was to propose a selection process for faculty use when selecting software tools for use in an IS or IT curriculum. This paper outlines an extensive set of evaluation criteria and then shows how the criteria can be applied in a structured manner. This approach provides a means of eliminating much of the subjectivity in the selection process. The structure and objectivity of this formal method yields the replicability missing from informal approaches.

Upon receiving feedback regarding this conceptual paper the author will reassess the proposed process and implement an online evaluation tool to streamline the weighting process and subsequent calculations. This framework will help decision makers not only in evaluation and selection of software tools for IS and IT programs, but also reduce the time and effort required for software selection.

References

ABET, Inc. (2008). Criteria for accrediting computing programs: Effective for evaluations during the 2009-2010 accreditation cycle. ABET Board of Directors. http://www.abet.org/Linked%20Documents-UPDATE/Criteria%20and%20PP/C001%2009-10%20CAC%20Criteria%2012-01-08.pdf.

Ahmed, M. I. (2003). A practical process for reviewing and selecting educational software. Technical Paper #8 Indiana University. PLATO Learning, Inc. A Practical Process for Reviewing and Selecting Educational Software.

Anderson, E. E. (1990). Choice models for the evaluation and selection of software packages. *Journal of Management Information Systems, 6*(4), 123–138.

Baki, B., & Çakar, C. (2005). Determining the ERP package-selecting criteria. *Business Process Management Journal, 11*(1), 75–86.

Bandor, M. (2006). Quantitative methods for software selection and evaluation. Technical Note (CMU/SEI-2006-TN-026), Carnegie-Mellon University. http://www.sei.cmu.edu/reports/06tn026.pdf.

Bednarik, R., Gerdt, P., Miraftabi, R., & Tukiainen, M. (2004). Development of the TUP model – Evaluating educational software. *Proceedings of the IEEE International Conference on Advanced Learning Technologie*s. 699–701.

Beise, C. (2006). Revisiting database resource choice: A framework for DBMS course tool selection. *Proceedings of the Twelfth Americas Conference on Information Systems*, Acapulco, Mexico August 4–6, 2139–2144 http://aisel.aisnet.org/amcis2006/266.

Bosilj-Vuksic, V., Ceric, V., & Hlupic, V. (2007). Criteria for the evaluation of business process simulation tools. *Interdisciplinary Journal of Information, Knowledge, and Management, 2*, 73–88. http://ijikm.org/Volume2/IJIKMv2p073-088Bosilj396.pdf

Carney, D. J., & Wallnau, K. C. (1998). A basis for evaluation of commercial software. *Information and Software Technology, 40*(14), 851–860.

Davis, L., & Williams, G. (1994). Evaluating and selecting simulation software using the analytic hierarchy process. *Integrated Manufacturing Systems, 5*(1), 23–32.

Dean, J. C., & Vigder, M. (2000). COTS software evaluation techniques. *Proceedings of the NATO Information Systems Technology Panel Symposium on Commercial Off-the-Shelf Products in Defence Applications*, Brussels, Belgium, April 3–5.

Denton, J. W., & Peace, A. G. (2004). Selection and use of MySQL in a database management course. *Journal of Information Systems Education, 14*(4), 401–407.

Donham, P. (2004). Ten rules for evaluating open source software. Point of view paper, Collaborative consulting. URL http://www.collaborative.ws/uploads/file/10%20Rules%20for%20Open%20Source%20Software.pdf.

Filho, A., Cavalcante, C. A. V., & da Costa, A. P. (2005). Multicriteria decision making on selection of decision analysis software. *Journal of Academy of Business and Economics. 5* (3), 11–16. http://www.thefreelibrary.com/Multicriteria+decision+making+on+selection+of+decision+analysis...-a0149213906

Franch, X., & Carvallo, J. P. (2003). Using quality models in software package selection. *IEEE Software, 20*(1), 34–41.

Fritz, C. A., & Carter, B. D. (1994). A classification and summary of software evaluation and selection methodologies. Computer Science Technical Report No.940823, Dept of Computer Science, Mississippi State University. http://www.google.com/url?sa=t&source=web&ct=res&cd=1&ved=0CAYQFjAA&url=http%3A%2F%2Fciteseer.ist.psu.edu%2F25945.html&ei=JnCDS57rBY6YsgPbv_jjDw&usg=AFQjCNG2xZ_nUa5N08HLjwUNyMZOr1ERtg

Gerdt, P., Miraftabi, R., & Tukiainen, M. (2002). Evaluating educational software environments. *Proceedings of the International Conference on Computers in Education*, 675–676.

Güngör ŞEN, C., & Baraçli, H. (2006). A brief literature review of enterprise software evaluation and selection methodologies: a comparison in the context of decision-making methods. *Proceedings of the 5th International Symposium on Intelligent Manufacturing Systems*, 29–31 May, pp. 874–883.

Hauge, O., Osterlie, T., Sorensen, C. F., & Gerea, M. (2009). An empirical study on selection of open source software – Preliminary results. *Emerging Trends in Free/Libre/Open Source Software Research and Development, ICSE Workshop*, Vancouver, Canada, May 18, 42-47.

Hosie, P., Schibeci, R., & Backhaus, A. (2005). A framework and checklists for evaluating online learning in higher education. *Assessment & Evaluation in Higher Education, 30*(5), 539–553.

Huber, J. T., & Giuse, N. B. (1995). Educational software evaluation process. *Journal of the American Medical Informatics Association, 2*(5), 295–296.

Jadhav, A. S., & Sonar, S. J. (2009). Evaluating and selecting software packages: a review. *Information and Software Technology, 51*(3), 555–563.

Kannan, G., & Vinay, V. P. (2008). Multi-criteria decision making for the selection of CAD/CAM system. *International Journal on Interactive Design and Manufacturing, 2*(3), 151–159.

Kao, D., Tousignant, W., & Wiebe, N. (2000). A paradigm for selecting an institutional software. *Proceedings of the Information Systems Education Conference 2000, (ISECON)*, Philadelphia, November 9–12. http://proc.isecon.org/2000/207/ISECON.2000.Kao.pdf.

Kontio, J. (1996). A Case study in applying a systematic method for COTS selection. *Proceedings of 18th International Conference on Software Engineering*, pp. 201–209.

Kontio, J., Chen, S. F., & Limperos, K. (1995). A COTS selection method and experiences of its use. *Proceedings of the 20th Annual Software Engineering Workshop*, Maryland.

Lai, V. S., Trueblood, R. P., & Wong, B. K. (1999). Software selection: a case study of the application of the analytical hierarchical process to the selection of a multimedia authoring system. *Information & Management, 36*(4), 221–232.

Lancor, L. (2008). Collaboration tools in a one-semester software engineering course: What worked? What didn't? *Journal of Computing Sciences in Colleges, 23*(5), 160–168.

Lê, Q., & Lê, T. (2007). Evaluation of educational software: Theory into practice. In J. Sigafoos & V. Green (Eds.), *Technology and teaching* (pp. 115–124). New York: Nova Science Publishers.

Lin, H., Lai, A., Ullrich, R., Kuca, M., McCllelland, K., Shaffer-Gant, J., et al. (2006). COTS software selection process. *Proceedings of the Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, 114–122.

Mannila, L., & de Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. *Proceedings of the 6th Baltic Sea Conference on Computing Education Research*, 32–37.

McDougall, A., & Squires, D. (1995). A critical examination of the checklist approach in software selection. *Journal of Educational Computing Research, 12*(3), 263–274.

Min, H. (1992). Selection of software: the analytic hierarchy process. *International Journal of Physical Distribution & Logistics Management, 22*(1), 42–52.

Ngai, E. W. T., & Chan, E. W. C. (2005). Evaluation of knowledge management tools using AHP. *Expert Systems with Applications, 29*(4), 889–899.

Nikoukaran, J., Hlupic, V., & Paul, R. J. (1998). Criteria for simulation software evaluation. Proceedings of the I998 Winter Simulation Conference, Washington, DC, 399–406.

Obeidat, M. A. (2006). Evaluation of new software technology: an empirical study. *International Management Review, 2*(3), 76–71.

Parker, K. R., Chao, J. T., Ottaway, T. A., & Chang, J. (2006). A formal language selection process for introductory programming courses. *Journal of Information Technology Education, 5*, 133–151.

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.

Plaza, I., Marcuello, J. J., Igual, R., & Arcega, F. (2009). Proposal of a quality model for educational software. *EAEEIE Annual Conference*, 1–6.

Polančič, G., Horvat, R. V., & Rozman, T. (2004). Comparative assessment of open source software using easy accessible data. *26th International Conference on Information Technology Interfaces*, June 7–10, 2004, Cavtat, Croatia.

Rowley, J. E. (1992). Evaluation of software. *Translating and the Computer, 14*, 117–126.

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

Squires, D., & McDougall, A. (1994). *Choosing and using educational software: a teachers' guide*. Philadelphia: RoutledgeFalmer, Taylor & Francis.

Squires, D., & McDougall, A. (1996). Software evaluation: a situated approach. *Journal of Computer Assisted Learning, 12*(3), 146–161.

Squires, D., & Preece, J. J. (1996). Usability and learning: evaluating the potential of educational software. *Computers in Education, 27*(1), 15–22.

Squires, D., & Preece, J. J. (1999). Predicting quality in educational software: evaluating for learning, usability and the synergy between them. *Interacting with Computers, 11* (5), 467–483. http://www.ifsm.umbc.edu/communities/Heur2.html.

Susser, B. (2001). A defense of checklists for courseware evaluation. *ReCALL, 13*(2), 261–276.

Teltumbde, A. (2000). A framework for evaluating ERP projects. *International Journal of Production Research, 38*(17), 4507–4520.

Tergan, S. O. (1998). Checklists for the evaluation of educational software: critical review and prospects. *Innovations in Education and Training International, 35*(1), 9–20.

Torchiano, M., Jaccheri, L., Sørensens, C.-F., & Wang, A. I. (2002). COTS products characterization. *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, July 15–19, 335–338.

van Staaden, P. & Lubbe, S. (2006). A case study on the selection and evaluation of software for an internet organisation. *The Electronic Journal of Business Research Methods, (4)* 1, 57–66, available online at http://www.ejbrm.com/vol4/v4-i1/Van_Staaden_Lubbe.pdf.

Voogt, J. (1990). Courseware Evaluation by Teachers: An Implementation Perspective. *Computers in Education, 14* (4), 299–307. http://doc.utwente.nl/68005/1/Voogt90courseware.pdf.