Simplicity First: Use of Tools in Undergraduate Computer Science and Information Systems Teaching

David Naugler
and
Ken Surendran
Department of Computer Science
Southeast Missouri State University
Cape Girardeau, MO 63701

Abstract

Use of tools – either home grown or industry supported - is inevitable in teaching CS/IS courses. The authors first examine the pros and cons of using tools in Computer Science and Information Systems courses. They briefly discuss the side effects of using tools on learning. In light of these discussions, they then focus on the impact of using tools in database management, and systems analysis and design on the students' overall learning by analyzing student feedback in these courses and student performance in the capstone project course in which knowledge gained in these two are applied. Based on their observations, the authors make a few suggestions for the appropriate use of tools and conclude that more care is required in using tools in lower-level courses.

Keywords: software tools, database, analysis and design, industry partnership, tool selection

1. INTRODUCTION

Information Systems and Computer Science (IS/CS) instruction often make use of software tools to help students master fundamental concepts. Depending on the type of course the tools used could be a programming language, an application development environment, a database management system, a productivity package, or even a complex application package. Developing or selecting appropriate tools for teaching CS / IS courses is an ongoing effort among academicians. Development of the programming language Pascal is a classic example. One can find a range of tools for most courses. Some of the tools used are industrial strength products, sometimes available in 'lighter' versions. These complex visual professional tools are made available to universities on excellent terms (e.g. Microsoft, IBM, Oracle, Borland, Metrowerks, and Cincom all make many of their tools available for a nominal cost), are free (e.g. Sun's Java environment), are available with a grant (e.g. IBM's Rational Rose), or come with the textbook. Since these are professional tools it can even be argued that experience in their use adds to the students' skill set.

Also, it is not uncommon to see courses that are built around a specific tool. For example, in an effort to combine teaching of both the principles and the practices of database systems, a course was designed around Java 2 Enterprise Edition (Moore 2003).

In this paper, the authors, based on their collective experience in teaching a range of CS / IS courses, argue for the importance of using simple tools especially in lower-level courses. They examine the use of tools in two lower-level courses and apply learning concepts to highlight the negative impacts of using complex tools at this level. They discuss the consequences of inappropriate tool use in analysis and design. Based on students' feedback, they also show how tools

were used effectively in higher-level courses. Finally, they discuss a few precautions to bear in mind while selecting industrial strength tools for higher-level courses.

2. STUDENTS ARE NOT PROFESSIONALS

Humans, especially those in western cultures, are extremely visually oriented. When the innovators at Xerox Park in the 1970's (Hiltzig 1999) determined to make computers easy to use they developed the visual interface controlled primarily with a mouse that Apple Computer borrowed for the Lisa and later, with major success in 1984, for the MacIntosh, and that Microsoft then borrowed for the ubiquitous Windows interface. Even the Linux flavor of Unix provides mouse controlled visual interfaces. Widespread use of the computer by non-specialists had to wait for such an interface for the operating system and major applications. Even though such visual tools provide convenient user interfaces, they are the most problematic for educational purposes, especially in the lower-level courses.

Professional tools are designed for the professional practitioner and not with pedagogy in mind. A 747 is not an especially appropriate airplane in which to first learn to fly; a semi-truck is not an especially appropriate vehicle in which to first learn to drive. Students are not yet professionals – they do not have the subject mastery, the domain knowledge and the experience we require in a professional. Nevertheless, we often use such complex tools in both the lower-level and the upper-level courses. In the following we examine how tools are used in lower-level courses such as programming and information systems.

2.1 Learning Programming

Complicated programming environments such as Visual Studio .NET, CodeWarrior, JBuilder, and Sun ONE are fine professional tools but are overwhelming environments for the beginner or even the intermediate programming student. Students initially struggle with a programming environment more than with the language they are learning – after all, no program can be written until the environment can be used.

Holt Software produces a Java environment called *Ready* which was written after studying the way programming students use software. *Ready* is a simple visual environment for learning Java programming – for example it has no toolbar icon since it was determined that students constantly used the tooltips to determine which icon was which.

The ideal would be the use of restricted development environments whose features could gradually be made apparent. An example of this kind of environment is Dr Scheme for learning the Scheme programming language which lets the user select several different versions of Scheme including beginner, intermediate and advanced levels. More difficult language features can remain hidden and unusable until the student is ready for them.

Neither of the products mentioned above is produced by a commercial software development environment company. Indeed, it makes better economic sense for a company to provide the professional product at a low price to university than to develop a separate pedagogically sound product.

A makeshift solution is to use a reasonably simple programmers' editor such as TextPad along with command line versions of software. Such editors usually have syntax awareness of languages available, and can handle program output and error messages in various windows. Programmers' editors such as emacs, although immensely capable and extremely flexible, can be overwhelming for novices.

2.2 Learning Information Systems (Fundamentals and Practice)

It is expected that the CS and IS students are familiar with productivity tools, even though these are not taught in a core course. Market forces determine the specific products used in preparatory courses. Such tools, in particular an electronic spreadsheet (Excel) and a database management system (Access), are used in the early IS courses - such as Fundamentals and Practice under the new curriculum guideline (Gorgone 2003). In these courses a core set of features in Excel and Access are normally considered, leaving complex features for students to learn on their own as and when a need arises. These productivity tools, by design, hide many complexities and focus on

user convenience (usually with a high level of visual features).

While these tools are essential, the students may develop tendencies to misapply them. It is not uncommon students use Excel for solving a database problem until they are unable to meet some requirements.

In using Access the usual starting point is creating tables, with "common sense" replacing design. (This is so since at this stage the students have not been exposed to design.) However, they do explore the facilities within Access and learn to produce the relationship diagrams for the tables they are using in the application they have developed. As a result, after solving quite a few simple Access exercises, the students develop a false sense that database *is* Access and the use of reverse engineering is the normal system design approach. This causes difficulties in learning the concepts in the analysis and design courses and also in the higher-level database courses. To minimize the development of such erroneous conceptions, the students should initially be given the appropriate design as input and asked to develop the application. Since at this stage much learning is by example, poorly designed databases should be avoided in examples.

3. SIDE EFFECTS OF TOOLS ON LEARNING

The most widely held theory of learning is constructivism. See (Ben-Ari 1998) for a survey of constructivism in the Computer Science Education on which much of this section is based. According to constructivism, students actively construct knowledge, building on knowledge that the students already have. Such knowledge includes facts (correct or incorrect), ideas and beliefs. This theory sheds some light on the uses of tools in the lower-level courses.

3.1 Inadequate Mental Models

Ben-Ari (1998) points out that an icon is just a representation and is only as useful as the mental model the user constructs – thus a Graphical User Interface (GUI) can be neither friendly nor intuitive to a novice. WYSIWYG (what you see is what you get) is not true since what you see is a visual representation of an internal data structure. In other words, GUI's and even WYSIWYG GUI's hide what is really happening and to be used effectively require that the user has an effective mental model of what is really happening. For general CS/IS pedagogy the moral is that the model must be explicitly taught.

The real problem is not that the student has absolutely no mental model for a tool or concept – such total ignorance would be quickly noticed and probably remedied. The students almost always have some mental models, which they will apply. The problem is that inappropriate mental models can seriously impede learning.

The term *bricolage* was coined by the anthropologist Lévi-Strauss. Bricolage is defined as *construction or creation from whatever is immediately available for use; something constructed or created in this way, an assemblage of haphazard or incongruous elements* (Brown 1993). Hacking, in the sense of trying something and seeing what happens, is a manifestation of bricolage often observed in CS/IS students, particularly in programming. Although hacking is occasionally done by almost anyone when programming, and trial and error is at times a valid learning approach, bricolage in not an adequate method for developing the kinds of knowledge that must be constructed by students of CS/IS, and indeed is a sign that the student has not developed an adequate mental model.

3.2 Designing Without a Conceptual Model

According to (Norman 2002; p xiii) "The human mind is a wonderful organ of understanding – we are always trying to find meaning in the events around us. One of the greatest frustrations of all is trying to learn how to do something that seems completely arbitrary and capricious. … A good conceptual model can make the difference between successful and erroneous operation of many devices in our lives. … When the designers fail to provide a conceptual model, we will be forced to make up our own, and the ones we make up are apt to be wrong. Conceptual models are critical to good design."

Although Norman is concerned primarily with physical devices, his book is a mainstay of

the field of Human Computer Interaction (HCI), which deals largely with the design of usable software interfaces. Unfortunately many software interfaces are designed poorly. Most professional development tools have a steep learning curve and assume some very specific background, almost always including a good understanding (i.e. mental model) of the area the tool is being used for. Thus a Java programming environment assumes that the user is already a knowledgeable Java developer, and a database environment usually assumes that the user understands databases. It is small wonder that we so often see our students so frustrated learning to use "completely arbitrary and capricious" software. Even if the interfaces are well designed for their intended audience (professionals) they are not designed for novices; indeed, the two audiences may need quite different interfaces.

In the following, use of tools in higher-level courses is discussed. The importance of the theoretical concepts discussed above help us in analyzing the impact of tools in these courses.

4. TEACHING ANALYSIS AND DESIGN CONCEPTS

Analysis and design concepts are often taught primarily in a systems analysis and design or a software engineering course and to some extent in a database course. Rational Rose and Oracle Design tools are some of the tools available in teaching analysis and design.

4.1 Wrong Emphasis

The main learning objectives in this course relate to learning the analysis and design concepts and applying them in generating analysis and design models - e.g. Entity Relationship Diagrams (ERD), Data Flow Diagrams (DFD).

It is very tempting to use various graphics tools for ERDs, DFDs, and so forth. Under the object paradigm, products from Rational Software (now IBM) cater to software engineering activities. Products such as Rational Rose or Oracle Designer may be overwhelming especially compared to the concept being learned.

Most students want to use graphical tools to produce nice looking output. This puts the emphasis on a matter of secondary or even tertiary importance and encourages the students to focus on a relatively minor aspect. A related issue is the degree to which the instructor should use such tools in teaching. There is nothing wrong with displaying a finished ER diagram for a class but the teacher must also show hand drawn examples and even draw some, step-by-step, with the class's help on the board. (A calculus teacher who always effortlessly chooses the correct next step solving a complicated indefinite integral sets up a very false model for what the student should (realistically) be able to do and makes it look too easy – as a result many students seriously misestimate the effort required and become frustrated when it is so much more difficult than they were lead, by example, to believe.)

4.2 Lack of Flexibility

In HCI, the use of paper prototypes (storyboards) for interface design is strongly recommended even for the early versions shown to the user. It may take considerable effort to produce UI prototypes using various tools. Tool use should be avoided at the initial stages of requirements analysis. Some of the reasons (Snyder 2001) for the success of the paper-prototype approach are especially instructive for education and design in general. Paper prototypes involve no coding at all. Clients feel free to comment on paper prototypes and suggest changes. This allows many problems to be found quickly and at a low cost largely by focusing on the functionality of the interface itself and not on issues such as color, fonts, and graphics which are not important at this stage but which are extremely easy to focus on. With paper prototypes feedback not pertaining to functionality is avoided. Since the clients are not intimidated by a paper prototype as they may well be by a working prototype they will be more comfortable and creative in examining the design. Thus paper prototypes are very effective during the early design stage. It is easy for a group to work on a paper prototype and to change it. Coding is a much more solitary activity and it is very difficult for more than three people to feel useful at coding time. A working prototype is psychologically quite hard to change significantly.

4.3 Students' Perceptions

One of the authors recently taught a database course for a mixed group of CS and IS students. A survey at the end of the course was conducted to find out, among other things, the need for more tools and/or concepts. While a large majority wanted more of neither, 30% wanted more tools and only 5% indicated more concepts. In their response to topics that were most difficult and easiest, 75% chose normalization as the difficult topic and 70% chose SQL as the easiest topic. In answering the question on most important and least important topics, 40% chose SQL as the least important and 50% chose design as the most important. The familiarity with SQL (used by some in earlier IS courses) and a background in programming seems to be the reason for choosing SQL as the easiest and least important topic.

Beginners make mistakes in database design (Antony 2002), in particular normalization and identifying necessary relationships accurately. Hence it is quite reasonable that students find normalization difficult. However, even though they found normalization difficult and recognized that design was important, they still wanted more tools than concepts. It appears that students perceive that learning tools (which often have a short useful life) is more important than learning concepts (which have lasting value). One reason for this could be the early emphasis that is placed on learning packages such as Access without due regard for design. Besides, by just creating tables for some simple systems the students tend to believe they are capable of developing database applications.

4.4 Student Performance

The other author has been teaching Systems Analysis and Design for sometime now. In this Visio, MS Project, Excel, and Access are used. The omission of complex CASE tools such as Oracle Designer, Developer was deliberate, even though the course is taught using the procedure-centric paradigm. The main assessment for the course is a group project with four-phases: planning, requirements specification, design and prototype. The instructor serves as the client, trying to simulate a real life situation. Since the focus is on applying the concepts, there is less emphasis on the tools: for instance, the use of Visio is not mandatory (however, the students are encouraged to learn these tools on their own).

Students produced good process and data models at the analysis stage. However, the data designs were not as good as the process designs (there were significantly more design errors in the data model). Some produced data models using reverse engineering (i.e., creating the tables in Access and using the relationships facility). In this process, they also failed to appreciate the significance of the relationships and normalization.

5. CAPSTONE COURSE

Software Engineering is the capstone course. In this course, the students work in teams on client sponsored system development projects. They use Rational Rose for modeling and, in addition, depending on the project, the teams use various other tools, which include specific programming languages, database management systems, and connectivity products. Following a set process, each team develops a useable product along with all the intermediary system documentation.

The course focuses initially on the principles and techniques used in Analysis and Design under the object paradigm. A few Rational Rose lab sessions are organized for the students to familiarize themselves with the tool. The students seem to have enough confidence to learn to draw the main diagrams (use-case, sequence, class, state, package) using Rational Rose on their own for documenting the results of their analysis and design. Features like reverse engineering are demonstrated just to let them know that there are several others features in Rose, which they may not be using in the course. However, they will explore these features and use them if and when their project required them.

5.1 Paradigm Shift

A common problem observed in most of the projects which use Access as the database management system is the discontinuity between the object model (class design) and relational data model. Instead of transforming the object model into the data model, the students carryout a separate ERD model from scratch. Access is designed for end

users as a productivity tool rather than a database development tool for building applications. As a result, it does not facilitate the treatment of object-relational transformations. Indeed, even major database vendors such as Oracle have only been gradually adding object-relational and object-oriented capabilities. There are still some relational versus object issues. Thus the techniques for manually transforming the object model into a data model have to be taught either in the analysis and design, in the software engineering, or in the database course.

5.2 Learning New Tools

Some of the projects required the use of Java for implementation with Access for database management. These teams had to learn the use of connectivity tools like JDBC on their own. Further, such topics are not necessarily considered in the database management course or in a course using Java for want of time. These senior students clearly demonstrated that they were able to learn such new tools - to the necessary extent - on their own and use them in their projects. The effort they had to put in depended on the relevant knowledge they had from other optional courses.

5.3 Significance of Design

In the capstone course the students are asked to state their individual reflections in their final project report. Most of the teams seemed to have realized the importance of design and design reviews. Some of the groups (which carried out proper design reviews) were surprised to find that they had spent only 20% of the project time on actual coding and testing. All the teams used the modeling tools (UML diagrams in Rational Rose) effectively to document the requirements spec and design, even though they had only a few Rational Rose lab sessions. The course has helped them develop enough confidence to conduct analysis and design and to document the resulting specifications using Rational Rose, a modeling tool used extensively in software houses. However, they did not exploit the other features (such as reverse engineering) the tool offers.

6. PRECAUTIONS

Usually, university departments sign agreements with software vendors for using their products in their course work. The cost and conditions of use vary. Some consider such agreements between an academic department and industry as invasion by industry into the educational system. Deron Boyles (Boyles 1998) gives several examples as to how companies use schools as a platform to promote themselves. His work is confined to school-industry partnerships where the tangible benefits to companies are normally hidden. However, in industry-university partnerships, the expectations are clearly understood. In some cases, universities get product licenses that are normally very expensive (such as ERP systems). Perhaps what is important for a university in this situation is to maintain academic neutrality even when using specific products.

6.1 Academic Neutrality

Academic neutrality is apparent when the primary purpose of using an industrial product in an academic curriculum is confined to that of a tool and there is consideration of other products in the course even if they are not used in practical sessions. There should be sufficient learning elements so that the product is only one element of the whole course. This may require careful course planning to ensure the product is used primarily as a tool for meeting the course objectives.

6.2 Hidden Costs

It is not possible to foresee all possible costs in fulfilling contractual obligations. Some of the hidden costs are:

- Staff time – both when they are training and when they are organizing the project
- Unforeseen Implementation Costs
- Unforeseen Training Costs
- Support Costs

There is also the issue of staff turnover. Generally only one faculty member teaches a course using a particular tool.

These tools go through continuous changes. Some of the organizations include, as part of the agreement, the use of latest versions. Some leave it to the academic institutions to make a change request. There is also a cost for upgrading the course content and the teaching material.

6.3 Education versus Training

In using tools, there is the risk of keeping the content of the course limited to what the tools offer. It is important to ensure that there are sufficient knowledge components in the course. Students may be able to get training for today's jobs but we need to ensure they get the education for tomorrow's changing requirements. Tool based training certainly increases employment opportunities. However, a long-term career requires a broader foundation. Thus even a tool-centered course must retain the knowledge focus.

6.4 Tool Selection

Some of the other risks pertain to the selection of the tool and the intended scope of use. In many areas of CS/IS, it is hard to predict which product will be the market leader (hence the possible demand for skilled graduates). One needs to study market conditions fairly thoroughly before choosing a product for use in the curriculum. It is also possible that some industries may use the academic partnership as a short phase in their overall product life cycle strategy. They might stop supporting or go for alternative training sources once market penetration is achieved. It is important to ensure long-term support for the product through contractual agreements.

In the new IS curriculum, a lot more emphasis is placed on e-commerce and web related applications i.e. a course on E-business and another on implementation in emerging environments (Gorgone 2003). Currently there are two competing products in the web services area (J2EE and .NET). It is quite likely the market is big enough for two products since they address different non-functional requirements (Williams, 2003 and Miller, 2003): openness and integration. What the academic community selects to teach will not depend on such non-functional factors (which may be crucial to the specific business organizations) but on the convenience the products offer in the overall flow of the curriculum. The academy also has a minor say in the sense it supplies the manpower for developing and supporting the applications.

6.5 Complexity of tools

Complex tools (like some integrated Development Environments - IDEs) take considerable effort to learn. For lower-level courses tools that are easy to learn should be considered. If a complex IDE is chosen for other reasons it should be used in several of the higher-level courses so that it is worth the overhead in learning it.

6.6 Support in Learning Concepts

McCracken (1998) raised the following question concerning the use of tools: at what point do you draw the line between basic concepts (good) and new and useful tools (also good)? There is no clear-cut universal answer to this. However, what is important is to ensure the tools do not create any pedagogical problems but instead enhance the learning of concepts.

7. CONCLUSION

Use of tools in both lower-level and upper-level courses has been considered in this paper. Both instructor experience and student feedback has been used to arrive at suggestions for appropriate use of tools. The significance of building conceptual models as part of learning has also been emphasized. In this regard the use of tools in lower-level courses needs to be examined carefully and with due consideration of the knowledge required to build appropriate mental models, not just to use the particular tool but to use similar tools. The models needed to learn tools is part of what must be taught. It is not necessary or usually appropriate to teach all the features of a professional tool – a well chosen subset will serve pedagogically. It is best to use tools designed or suitable for learners; such tools when available can make it easier both for teacher and students and even increase the learning of the subject material. Bricolage needs to be recognized as a symptom of an inadequate mental model.

Senior students should be encouraged to learn tools on their own but only after they have developed adequate mental models usually by instruction in a subset of features of the tool or by experience with similar tools.

Several suggestions have been made concerning the choice of tools for use in courses, including maintaining academic neutrality, examining tool complexity, and ensuring ease of use in learning concepts.

Further, in selecting a tool, it is important to use criteria that are drawn from the philosophy of the program and apply it consistently. Evaluating the use of a tool in a course in light of student feedback is essential even if it is popular tool (Hadjerrouit 1998).

8. REFERENCES

Antony, S. R. and D Batra (2002). "CODASYS: A Consulting Tool for Novice Database Designers," The Database for Advances in Information Systems, Vol. 33, No. 1.

Ben-Ari, Mordechai (1998). "Constructivism in Computer Science Education," Inroads SIGCSE Bulletin, Vol. 30 No. 1, pp. 257-261.

Boyles, D. (1998). American Education and Corporations: The Free Market Goes to School *Garland Publishing, New York*.

Brown, Leslie (Ed) (1993). The New Shorter Oxford English Dictionary, Clarendon Press, Oxford.

Gorgon, J. T., G.B. Davis, J.S. Valacich, H. Topi, D.L. Feinstein, and H. E. Longenecker, Jr. (2003). "IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems"," The Database for Advances in Information Systems, Vol 34, No. 1.

Hadjerrouit, S. (1998). "Java as First Programming Language: A Critical Evaluation," Inroads SIGCSE Bulletin, Vol 33 No. 2, pp 43-47.

Hiltzig, Michael (1999). Dealers Of Lightning: Xerox PARC and the Dawn of the Computer Age, HarperBusiness.

McCracken, D. D. and D. J. Frailey (1998). "A conversation about Computer Science Education," Inroads SIGCSE Bulletin, Vol. 30 No. 2, pp. 36-39.

Miller, G. (2003). ".NET vs. J2EE," Communications of the ACM, Vol. 46, No. 6, pp. 64-67.

Moore, T. K. (2003). "Bringing The Enterprise into a Database Systems Course," SIGCSE Bulletin – Inroads, Vol 34 No. 1., pp. 262-265.

Norman, D. A. (2002). The Design of Everyday Things, Basic Books.

Snyder, C. (2001). "Paper prototyping." (http://www-106.ibm.com/developerworks/library/us-paper/?dwzone=usability) IBM developerWorks. (accessed 12 May 2003).

Williams, J. (2003). 'J2EE vs. .NET,' Communications of the ACM, Vol. 46, No. 6, pp 59-63.