Creating Open Source Lecture Materials

A Guide to Trends, Technologies, and Approaches in the Information Sciences

William H. Hsu
Kansas State University

This article surveys recent and continuing trends in software tools for preparation of open courseware, in particular audiovisual lecture materials, documentaries and tutorials, and derivative materials. It begins by presenting a catalog of tools ranging from open source wikis and custom content management systems to desktop video production. Next, it reviews techniques for preparation of lecture materials consisting of five specific learning technologies: animation of concepts and problem solutions; explanation of code; video walkthroughs of system documentation; software demonstrations; and creation of materials for instructor preparation and technology transfer. Accompanying the description of each technology and the review of its state of practice is a discussion of the goals and assessment criteria for deployed courseware that uses those tools and techniques. Holistic uses of these technologies are then analyzed via case studies in three domains: artificial intelligence, computer graphics, and enterprise information systems. An exploration of technology transfer to college and university-level instructors in the information sciences then follows. Finally, effective practices for encouraging adoption and dissemination of lecture materials are then surveyed, starting with comprehensive, well-established open courseware projects that adapt pre-existing content and continuing through recent large-scale online courses aimed at audiences of tens to hundreds of thousands.

1. Trends in Open Courseware for Information Sciences

1.1 Tools

This section provides a brief history of open educational resources (OER) for the information sciences, followed by a taxonomic survey of OER development tools.

1.1.1 Brief History

Open educational resources (OER) for the information sciences date back to the early decades of the field, beginning with the development of *PLATO (Programmed Logic for Automated Teaching Operations)*, the first computer-assisted instruction (CAI) system, at the University of Illinois. (Van Meer, 2003; PLATO History Foundation, 2011) The first version of *PLATO*, implemented on the ILLIAC I *circa* 1960, included what is now termed lessonware and was funded jointly by the U.S. Army, Navy, and Air Force. Meanwhile, by the late 1960s, video lecture consortia such as the Stanford Honors Co-op were delivering proprietary closed-circuit television content to corporate sponsors (House & Price, 2009). The 1970s brought a wave of intelligent tutoring systems (Carbonell, 1970; Sleeman & Brown, 1982; Iiyoshi & Kumar, 2008). By the 1980s, cable-access distance learning and extension courseware had begun to be distributed using precursors of open source licenses, culminating in the founding of the Free Software Foundation in 1985 and the first releases of the Berkeley Standard Distribution (BSD) License (1988), GNU General Public License (1989), Open Content License (1998), and Creative Commons License (2001). (Free Software Foundation, 2012) Abelson, a founder of the Massachusetts Institute of Technology *OpenCourseWare* (MIT OCW) initiative (Abelson, 2007; Attwood, 2009) and founding member of Creative Commons (Creative Commons Corporation, 2011), had been distributing *Structure and Interpretation of Computer Programs*, a leading introductory textbook in computer science, online. With the advent of MIT OCW, video lectures prepared for the MIT/Hewlett-Packard consortium (House & Price, 2009) as early as 1986 were made available (Abelson, 2005).

1.1.2 Technologies for Producing Open Source Software

When discussing "open source tools", professionals and students in science, technology, engineering, and mathematics (STEM) fields often refer only to *open source software* (DiBona, Ockman, & Stone, 1999; Raymond E. S., 1999; Open Source Initiative, 2006)[1] rather than the more general concept of *open content* (Wiley, 2011) as coined by David Wiley in 1998 (Wikipedia, 2012). The means of production are diverse for both forms of creative work, with free redistribution and access being the unifying characteristic. For open source software, however, the chief production technologies are software

[1] The first edition of DiBona, Ockman, & Stone (1999) is available as an e-book from http://oreilly.com/openbook/opensources/book/.

engineering tools: integrated development environments; content management systems; and version control systems, also known as “source code control systems”.

Integrated development environments (IDEs) are suites of development applications consisting of source code editors, compilers (and/or interpreters), and build/execution controls, plus optional components such as build utilities, interfaces to version control systems, visual code layout and refactoring tools, and interactive code inspection and debugging tools. (D'Anjou, Fairbrother, Kehn, Kellerman, & McCarthy, 2005; Nourie, 2005) They range from the proprietary (*e.g.*, Microsoft *Visual Studio* and Apple *Xcode*) to open source (*e.g.*, *Eclipse* and Oracle *NetBeans*). The range of available IDEs depends foremost on the programming languages to be supported and secondarily on the development platform, comprising the computer architecture, operating system, and compilers or interpreters. For ease of use, efficiency, and portability, many open source developers use simple editors, version control, and compilation tools to augment or replace full-featured IDE s when their full power is not required.

A ***content management system*** (CMS) is a collection of procedures (implemented manually or computationally) for organizing and carrying out work flow in a collaborative environment. (Depow, 2003; Mauthe & Thomas, 2004) Specific CMSes may be implemented as web services or using other software as a service (SaaS) architectures, or as standalone applications such as most wikis. Both types of CMSes occur in both proprietary and open source varieties. Schaffert *et al.* (2006) describe *semantic wikis*, which capture information on the deep relational structure between pages and provide this information to agents and services beyond mere linking. These are referred to as *semantic wikis*, after the Semantic Web, or Web 3.0. Moreover, both enterprise and public wikis may be used for distance learning and distribution of lecture materials, but in academic institutions and consortia, enterprise wikis are the more common type. The most popular enterprise wikis are the Wikimedia Foundation’s *MediaWiki*, *Tiki Wiki CMS Groupware*, and *TWiki.* (Wikipedia, 2012)

A ***version control system,*** also called a source code control system or revision control system, is a specific type of software configuration management system designed for the curation and archival of collaboratively created content, including but not limited to program source code. The dominant version control systems in use at present are the client-server systems *Subversion* (SVN), *Concurrent Version Systems* (CVS), and *Git*. Because of their predominance within the open source community, and the existence of popular hosting services such as *GitHub*, which supports *Git*, and *SourceForge*, which supports a number of collaborative version control systems, SVN, CVS, and Git have retained their preeminence in social development contexts such as authoring of open source software and open content.

1.1.3 Current and Emerging Technologies for Producing Other Open Content

Other forms of open content (Wiley, 2011) have included databases and data acquisition resources such as the *OpenMind Initiative* (Stork, 1999; Singh, et al., 2002; Chklovski & Gil, 2005), a collaborative framework for producing large data sets, domain knowledge bases, and ontologies for commonsense reasoning and machine learning. Open courseware itself is an instance of open content, often excerpted

and reused with “some rights reserved” as per the Creative Commons License (Creative Commons Corporation, 2011).

Current software tools for preparation of open courseware, especially audiovisual lecture materials, documentaries and tutorials, and derivative materials, focus on production of notes, slides, audio (traditional “podcasting”), and videos (including “webcasting”). Numerous office suites providing functionality similar to *Microsoft Office* are distributed under purportedly free software licenses. The best-known and most popular of these at present is Apache *OpenOffice* (Apache Software Foundation, 2011), originally released by Sun Microsystems and briefly by Oracle Corporation. (Wikipedia, 2012) Most office suites provide native file formats for lecture slides with animations and for non-interactive reading material, and support exporting of content to static formats such as text and PDF. True open source packages for video production include *Blender* (Blender Foundation, 2012) and *VirtualDub* (Lee, 2012), whereas some proprietary software is freeware or shareware when used under a noncommercial license. *Fraps* (Beepa, 2012), a popular video capture utility used to make recordings of software demonstrations and machinima-based animations (Lowood & Nitsche, The Machinima Reader, 2011), is one such example.

1.2 Computing and Information Science Disciplines

The list of content production tools given in the previous section is representative rather than comprehensive, but it covers a majority of basic content types by category and format. To understand the potential impact of these tools when used in tandem, a brief review of the state of the field in computing and information sciences is provided here.

Computing science incorporates theoretical computer science and its applications to STEM disciplines, comprising the field generally known as “computational science” or scientific computing, whose branches include industrial applications (technical computing), analysis of data (statistical computing), applied numerical analysis, *etc.* Meanwhile, the very broad interdisciplinary field of ***information science*** overlaps with computer science, but also includes aspects of the theory and practice of information processing, management, and retrieval that that are not purely computational, as they incorporate aspects of mathematics, cognitive science, linguistics, library science, and social sciences. Subsuming computational science and engineering *and* information sciences is the even broader academic field of ***informatics*** – a term for the “[study of the] structure, algorithms, behavior, and interactions of natural and artificial systems that store, process, access and communicate information”. (Wikipedia, 2012) This definition underscores a subtle but important distinction: the systems need not be computational, so that informatics is generally distinct from computer science and information technology.

Educational issues often reported among in the above fields include low comprehension and retention rates among undergraduate students. Loidl, Mühlbacher, and Schauer (2005) and Stephenson, Gal-Ezer, Haberman, and Verno (2005) discuss prominent unmet needs in the pedagogy of informatics and computer science, and put forth the hypothesis that student performance below expectations across

information science curricula are due to a lack of comprehensible preparatory material at the high school and early university level. desJardins and Littman (2010) documented a materials-oriented remediation plan for issues identified by instructors and students, along with data demonstrating positive student outcomes. These materials have been made publicly available (Littman, 2007; Rutgers University, 2012) and provide several of the motivating examples of production, evaluation, and dissemination techniques in Section 3 of this article.

2. Tradeoffs

Both open source tools for content production, and open content itself, represent tradeoffs when compared to closed source analogues. Advocates of open content and open source software cite lower cost, accessibility, and community quality assurance while advocates of commercial and other proprietary tools cite provider services, functional features, and ease of adoption, maintenance, and support. Understanding this tradeoff presents a challenging economics problem because of fundamental differences in the means of production, motivating rewards for labor, and underlying forms of capital involved. (Lerner & Tirole, 2002; Lerner & Tirole, 2004) This is especially true in domains such as computer graphics, where content and content development tools are conflated due to both of them being used directly by instructors and students.

As part of a 2007 interview of CEO Dean Drako of Barracuda Networks[2], *CNet* columnist Matt Asay reported on a survey conducted by Barracuda of 228 of their enterprise customers who were asked to list one or more advantages of commercial software *versus* open source software. (Asay, 2007)

2.1 Claimed Advantages for Commercial Off-The-Shelf (COTS) Tools

The top three specific advantages of commercial software over open source software cited by Barracuda Networks customers were: vendor professional services (cited by 65%), ease of adoption (47%), and automated updates (41%). These were followed by six additional specific advantages, for a total of nine: reduced IT support (35%), best product functionality (28%), security (23%), code quality (17%), intellectual property protection (7%), and price (3%). (Asay, 2007)

Comparative benefits of commercial software cited (as disadvantages of open source software) in an article first compiled in 2004 by the Canadian Internet Policy and Public Interest Clinic (CIPPIC) were: "liability for intellectual property infringement", a "guarantee of quality or fitness", and "licensing" issues. (Kerr & Bornfreund, 2007) While liability, accountability, and warranties of quality or fitness are cited as advantages by COTS proponents, some open source consultants and vendors such as GBDirect, Ltd. have noted that both proprietary and open source licenses "typically disclaim all liabilities and warranties, including such basic warranties as merchantability and fitness for purpose". (GBDirect, 2004)

[2] Barracuda Networks, Inc. is a company providing "security, networking and storage solutions based on network appliances and cloud services" ("Barracuda Networks", Wikipedia, 2012)

17% of customers surveyed by Barracuda cited “quality” as one of nine specific COTS advantages, though this survey did not list “warranty” as a separate response. (Asay, 2007)

2.2 Claimed Advantages for Open Source Tools

According to the 2007 Barracuda customer survey discussed above, the top three specific advantages of open source software over commercial software were price (cited by 80%), access to source code (57%), and community code review (41%). These were followed by six additional specific advantages, for a total of nine: bug fix turnaround (18%), security (15%), code quality (15%), best product functionality (15%), ease of adoption (10%), and intellectual property protection (5%). (Asay, 2007) Of these desiderata, the last four were also cited as advantages of commercial software in the same survey, with commercial software receiving a higher percentage of citation in these categories: 17% *vs.* 15% for code quality; 28% *vs.* 15% for best product functionality; 47% *vs.* 10% for ease of adoption; and 7% *vs.* 5% for intellectual property protection. These responses were from the same pool of customers, who were asked to list advantages of commercial software over open source software **and** *vice versa*.

Kerr and Bornfreund (2007) cite “four inherent advantages [of open source software] over proprietary software”: (1) lower cost; (2) access to source code, allowing the user community to “detect and fix programming bugs” and providing for greater customizability and freedom in scheduling updates; (3) security through transparency; and (4) reduced vendor “lock in”. Similarly, 435 respondents to the 2009 Future of Open Source survey at the InfoWorld Open Source Business Conference gave the “top four factors that make open source software attractive” as: (1) lower cost; (2) security, (3) no vendor “lock in”, and (4) better quality. (Guseva, 2009) In an earlier survey of firms, Dedrick and West (2003) reported lower cost, third-party expertise availability, risk tolerance, and “trial basis” as adoption factors. Other researchers have elaborated on the nature of “free” user-to-user assistance as a part of the open source model, a consideration that is often relevant to educators seeking accessible resources. (Bonaccorsi & Rossi Lamastra, 2003; Lakhani & Hippel, 2003; Singh, Twidale, & Rathi, 2006)

Open source developers have long cited rationales for open source as a *business* model, especially lower production costs amortized over individuals, large user communities, and reliability through transparency. (DiBona, Ockman, & Stone, 1999; Hars & Ou, 2001; Krishnamurthy, 2002; Carmichael & Honour, 2002; O'Hara & Kay, 2003; Ye & Kishida, 2003; Downes, 2007) von Krogh and Spaeth (2007) note that significant consequences of these properties are that they result in a high influx of developers, perceived market tension with proprietary software publishers, and a paradigm shift in perceptions of intellectual property, as also noted by Fitzgerald (2006).

The causal attribution of specific problems experienced by users of open source tools, whether to instability, a dearth of documentation, production quality, software maintenance, or deficiencies in content, is difficult to further attribute to economic factors of production. These include labor, materials, means, and various forms of organizational, intellectual, and social capital that are technically complex. At present, they are infeasible to quantify because of a lack of controls or baselines for comparing the production platforms of free software *versus* commercial software.

3. Techniques for Lecture Material Preparation

Next, the article continues with a review of techniques for preparation of lecture materials consisting of five specific learning technologies: animation of concepts and problem solutions; explanation of code; video walkthroughs of system documentation; software demonstrations; and creation of materials for instructor preparation and technology transfer. These technologies are presented first in terms of a motivating pedagogical principle and then in terms of means, choices, and costs associated with their implementation. Accompanying the description of each technology and the review of its state of practice is a discussion of the goals and assessment criteria for deployed courseware that uses those tools and techniques. This technical survey concludes with examples and recommendations from the relevant literature on education and outreach.

3.1 Production Techniques

Effective production of open educational content involves materials design, planning and development of a syllabus that will incorporate the content, and – due to the open nature of the content – deliberate planning for reusability by other instructors and content developers. In this section, we focus on development of syllabi for video lectures, along with the requisite preparation of materials.

Interactive materials design consists of mapping from teaching objectives to concrete lesson plans with corresponding educational media, including text, audiovisual components, and software. A fundamental requirement and common first step of materials design is to adapt and integrate text, some of which may be recaptured from hard copy and some of which may have been prepared for electronic distribution (*e.g.*, as digital textbooks or documents). Office suites such as those mentioned in Section 1.1.3 facilitate repurposing of content (Obrenovic, Starcevic, & Selic, 2004; Verbert, GaSvevic, Jovanovic, & Duval, 2005). *Content repurposing*, the adaptation of existing information from various media to serve a new use case (Hossain, Rahman, & El-Saddik, 2004; Obrenovic, Starcevic, & Selic, 2004; Duffy, 2008), can be achieved through methods that include format conversion (including screen captures), embedding of images and video, hypertext linking of documents, and porting of software. Some user interface and web components are designed for portability and reuse; these components range from widgets to frameworks and whole graphical user interfaces (GUIs), particularly in educational software toolkits. Finally, at the most basic technical level of reuse is source code, which may be provided purely for functionality (*e.g.,* as part of a component library) to support educational programming, or together with documentation, as a teaching material in its own right (*e.g.,* example code to illustrate algorithms).

Video lecture syllabus planning begins with formulating a lesson plan that incorporates source material. This provides a key goal of content repurposing in the materials design phase: to excerpt or adapt text and multimedia from previously formatted documents: PDF, *PowerPoint*, *etc.* (Poindexter & Heck, 1999) Potential value added by this step includes retention of formatting (including layout and mathematical typesetting); adaptation to an interactive medium, such as interactive animation; and subsequent capture of preprogrammed or planned interactions, such as software demonstrations and problem

solving traces, on video. (Evans & Fan, 2002; Duffy, 2008) A large segment of the videos of this nature distributed via *YouTube* are produced using popular video capture utilities, including the free utility *FRAPS* (Beepa, 2011). Finally, the integrated video lecture must be recorded and produced. Many educational videos are produced using feature-intensive third-party screen recording software such as *Tegrity* or *Camtasia* (Prabaker, Bergman, & Castelli, 2006; Gaspar & Langevin, 2007).

Finally, planning for reusability involves several considerations. First, regularly offered courses typically require a minimum frequency of turnover in order to maintain the freshness of the information, and the instructor's currency and familiarity with it. This frequency depends upon the discipline and the specific domain of the course, and in particular with the speed of new developments. A balance is needed between popularity of modules and materials across many curricula and demand for the material within one course. Second, the time constraints placed on authors of open courseware and content may necessitate triage: correcting obsolete or erroneous material in order of priority. This need is further exacerbated by the challenges inherent in preparing lectures as long as a class period: producing new versions of lectures demands recording and post-production time in addition to the time needed to update old lectures or correct errata. Third, keeping up with demand in academia or industry for timely and popular topics requires some awareness and responsiveness to student needs. These may be elicited through informal polling ranging from impromptu classroom straw polls to nonscientific open surveys on a course web site, or through more formal surveys or controlled studies. Fourth, courses may need to be tailored for different clientele, such as undergraduate and graduate students, on-campus and distance students, or traditional students and industry professionals (either group of which may be enrolled online). Depending on whom a course is being offered to, demand may arise for additional material or spin-off material for a second course. This type of deliberate reuse carries its own incentives, such as being able to create review notes for a prerequisite course, and its own risks, such as having too much redundancy or a "one size fits all" type of inflexibility in the prepared materials. Fifth, a related kind of reuse is cross-medium: most courses include material from books, written notes, lecture slides, homework, exams, recordings, videos, and open content may span or integrate more than one of these.

3.2 Learning Technologies

Several unifying themes recur across learning technologies in information technology: problem-solving traces; learning by watching (Kuniyoshi, Inaba, & Inoue, 1994), apprenticeship learning (Collins, Brown, & Newman, 1987), and learning by doing (Shank, Berman, & Macpherson, 1999); and principled integration and reuse. These are not necessarily specific to open source tools, but as the examples given in this section illustrate, they are quite prevalent among open source technology for the approaches surveyed.

Delivering this type of visualization as open content requires preparation of multimedia in formats that are portable across platforms. Commercial presentation software packages such as Microsoft *PowerPoint* provide such functionality, but to reach the widest audience, many authors now make extensive use of public video sites, particularly *YouTube*. These are frequently accessed by linking offsite

(Luo, 2010). Production of multimedia often involves a combination of video technologies. The traditional web approach of linking still accounts for a significant amount of video content delivery. Embedding of videos in course wikis, blogs, and other groupware is becoming prevalent with the advent of Web 2.0 technology, namely*, Asynchronous JavaScript and XML (AJAX)*, where XML stands for *eXtensible Markup Language*. (Lin, Chi, Chang, Cheng, & Huang, 2007; Duffy, 2008; Luo, 2009) Finally, digital compositing using video capture (Beepa, 2011) and 3-D computer-generated animation (CGA) engines such as machinima (Lowood, 2006) provide three-dimensional multimedia production technology that is used in both entertainment and education industries. (Chang, Chiu, & Hung, 2010; Lowood & Nitsche, 2011)

3.2.1 Animation of Concepts and Problem Solutions

Animating the process of solving problems in mathematics and programming predates the earliest online educational resources surveyed in Section 1.1.1: Tufte (1997) gives many examples of visual explanations involving paper pop-ups, three-dimensional mechanical displays, and other forms of non-electronic (often manually-operated) displays. With the proliferation of educational supplements specializing in solved problems, a niche has formed for computer visualization of both abstract concepts and problem solutions. (Tufte, 1990; Tufte, 1997; Tufte, 2006)

Early work on systems for computer-assisted instruction (CAI) in mathematics and foreign languages led to the first controlled evaluations (Suppes & Morningstar, 1969). Simulation-based educational software systems were identified as a key trend in CAI by the 1970s, especially in science, technology, engineering, and mathematics (STEM) disciplines. (Chambers & Sprecher, 1980; Rieber, 1990; Mayer & Anderson, The Instructive Animation: Helping Students Build Connections Between Words and Pictures in Multimedia Learning, 1992; Pane, Corbett, & John, 1996) Successes in computer-based instructional technologies led to a wide proliferation of CAI applications in the 1980s. These fell into two main branches: drill-and-practice systems and tutoring systems. Kulik and Kulik (1991) identified two other major categories of computer-based instruction systems besides CAI: *computer-managed instruction*[3], including automated grading and critiquing[4]; and *computer-enriched instruction*, comprising some simulations, data generation, and environments for exploring and experimenting with software.

By the early 1990s, algorithm visualization had emerged as its own distinct application area of CAI (Gloor, 1992; Hundhausen, Douglas, & Stasko, 2002; Naps, et al., 2002) This led to the development of recommended practices for interactive visualization that closely mirror and extend principles of information graphic design (Tufte, 1997). Lester *et al.* (1997) reported improved problem solving in a tutoring system with an "animated pedagogical agent" – an effect they attributed to multimodal

[3] Rößling *et al.* (2008) provide a representative survey of contemporary computer-managed instruction systems.
[4] Automated grading and test case generation systems for basic programming courses are also studied by Edwards (2003), who notes that they tend to elicit a cultural shift from *ad hoc* debugging styles to more principled and systematic ones.

information delivery, improvements in personalization and contextualization of advice, and improved student motivation due to the use of an anthropomorphic agent. Additional information design principles of animation in educational multimedia have also been guided by cognitive psychology. These include a cognitive design theory developed by Mayer (1999) for facilitating problem-solving transfer, wherein experience in solving one problem generalizes to others. This theory gave rise to seven principles put forth by Mayer and Moreno (2002):

1. “Present animation and narration rather than narration alone” (multimedia)
2. “Present on-screen text near rather than far from corresponding animation” (spatial contiguity)
3. “Present corresponding animation and narration simultaneously rather than successively” (temporal contiguity)
4. “Exclude extraneous words, sounds, and video” (coherence)
5. “Present animation and narration rather than animation and onscreen text” (multimodality)
6. “Present animation and narration rather than animation, narration, and on-screen text" (nonredundancy)
7. “Present words in conversational rather than formal style” (personalization)

Plaisant and Schneiderman (2005) provide the following list of principles for producing recorded demonstrations in cybereducation:

1. "Provide procedural instruction rather than conceptual information."
2. "Keep segments short" (15 - 30 seconds).
3. "Ensure that tasks are clear and simple," using scripted narration of concrete running examples.
4. "Coordinate demonstrations with textual documentation."
5. "Use spoken narration" (as opposed to textual explanations), for greater communication efficiency through multisensory integration.
6. "Be faithful to the actual user interface".
7. "Use highlighting to guide attention", combining sound and visual effects for amplified multisensory effect.
8. "Keep file sizes small" by using on-screen digital recording, optimized data formats, codecs, and compression schemes for the type of multimedia (usually audiovisual) information being delivered.
9. “Strive for universal usability” by developing demonstrations with high portability and accessibility, minimal documentation requirements, and highly intuitive user interfaces.
10. “Ensure user control” by providing ease of navigation through the material, such as by allowing users to resume demonstrations and skip previously-covered parts.

In recent years, numerous controlled studies have been conducted on the effectiveness of animation in CAI in various disciplines. While some have shown that animation of physical and biological processes can improve comprehension of basic science concepts (Thatcher, 2006), findings have been inconclusive in many disciplines, such as medical education (Ruiz, Cook, & Levinson, 2009). Holzinger, Kickmeier-

Rust, and Albert (2008) report similar variability in a survey of dynamic media for computer science education, and conclude:

> Dynamic media is only successful in facilitating learning in comparison to traditional static media such as texts or images, when they are able to (1) reduce the cognitive load, which is necessary to comprehend them, (2) serve to generate mental models of a concept and, consequently (3), offer visualizations that *correspond to a meaningful mental model.*

Ruiz *et al.* and Holzinger *et al.* report that fielded systems bear out the findings of cognitive psychologists with regards to cognitive load theory: that dynamic visualizations of complex processes, both physiological and algorithmic, can aid in comprehension, but that attentional cueing is needed to focus the viewer's attention on one or a few aspects of the visualized processes (Koning, Tabbers, Rikers, & Paas, 2007; Hasler, Kersten, & Sweller, 2007). This bears out earlier research on mental models, particularly *epistemic fidelity*, a measure of goodness of algorithm visualization that is based on the premise that graphics closely correspond to the mental model of an algorithm that a designer forms and uses. (Hundhausen, Douglas, & Stasko, 2002) Application of these criteria have led to social constructivist design principles for individualized interactive learning using tools for algorithm visualization (Lawrence, Badre, & Stasko, 1994) and ethnographic field studies of how such tools were used in undergraduate classes in computer science. (Hundhausen, 2002; Stasko, Badre, & Lewis, 1993) For examples of additional learning technologies based on social constructivism, the interested reader is referred to Jonassen, Howland, & Marra (2011).

3.2.2 Explanation of Code

A key part of computer science pedagogy centers around learning to produce code (*i.e.*, program) by reading good examples of code that accomplishes certain specified functions. (Raymond D. R., 1991) This approach has its roots in *structured programming*, a paradigm of computer programming that aims at improving the clarity, reliability, and development efficiency of programs by using functions (also known as procedures and subroutines in imperative programming and methods in object-oriented programming), iteration, and block structure. (Dijkstra, Hoare, & Dahl, 1972)

The practice of "learning by reading code" was further advanced by the introduction of *literate programming*, the synthesis of documentation paradigms "suitable for program exposition" and coding paradigms "suitable for program creation". (Knuth, 1984; Van Wyk, 1990; Ramsey, 1994) In addition to establishing a methodology for creating programs with formatted inline documentation together with a "web of abstract objects", Knuth (1984) highlighted the role of documentation in programmer training, which has since been increasingly recognized as an integral component of software education. (Sametinger, 1994) This has led to an art of computer documentation that has garnered its own constructivist theory. (Spinuzzi & Zachry, 2000; Spinuzzi, 2002)

Building upon this practice of literate programming and code review, Linn and Clancy (1992) developed case studies of programming problems and outlined a pedagogical framework for presenting decisions and the rationale for each one. They presented a case study template that combined program

specifications, example inputs, diagrams of program traces, pseudocode, analogies depicting the program's desired behavior, references to more general templates (behavioral supertypes) and those using the template as a behavioral subtype or other design pattern, and debugging notes. In later work surveying several studies of their own and other computer science education researchers, Clancy and Linn (1999) report that problem solving transfer (Mayer, 1999) by means of learning design patterns varies by student background, being very low for novices; is directly proportionate to abstract understanding, generality, and reusability; is enhanced when accompanied by syntax, use case examples, and execution traces. As with algorithm visualization, students still need to learn by doing and interacting. For example, Gaspar and Langevin (2007) note that observing the process of deriving computer programs as solutions to problems, which they call "coding with intention" (as opposed to "cut and paste-oriented programming"), tends to restore a deeper understanding of solution design and synthesis. (Starr, Manaris, & Stalvey, 2008) By contrast, they find that presentation of working code examples alone tends to result in shallower analogical thinking.

Learning to read source code and design documents, like learning to read documentation, is a crucial and foundational skill for students of information technology, including both theoretical computer science and informatics. The organization and documentation of software differs by type – Brooks (1995) classifies software as simple standalone programs, *programming products* (tested, documented, portable, and fielded applications), *programming systems* (application programmer interfaces, libraries, frameworks, modular interfaces, and middleware), and *programming systems products*. Studying solutions by dissection, using concrete examples, specifications, test cases, and formal properties, is recommended for many topics in computer science education, from elementary programming (Deek, Kimmel, & McHugh, 1998; Ben-Ari, Berglund, Booth, & Holmboe, 2004) to computational science and informatics (Stevenson, 1993) to computer security and information security (Yang, 2001). This underscores the importance of open source materials: O'Hara and Kay (2003) advocate the study of open source software in computer science education for its high achievable degree of verifiability. In an annual President's Letter to the Association for Computing Machinery (ACM), Patterson (2006) lists courses that "leverage high-quality examples of the open source movement" as the #1 category of "course I would love to take", while advocating "writing documentation for portions of open source code" as a good way to learn large systems.

3.2.3 Walkthroughs of Programming Systems and Products: Documentation and Videos

We have seen how interactive algorithm visualizations and literate programs have been systematically shown to be superior to watching recorded animations and reading source code as flat text, through studies by both educational psychologists and professional societies within information technology. A natural extension of the hypothesis that interactivity yields good mental models for learning programming and problem solving skills is that this benefit also arises from interactive user and developer documentation. Toward this end, studies have been conducted that assess the effectiveness of online documentation (Hertzum & Frøkjær, 1996) and instructional materials in general (Mehlenbacher, 2002; Zhang, Zhao, Zhou, & Nunamaker, 2004). A common conclusion was that

usability measures such as readability, ease of interpretation, adaptability to differences in student background and aptitude are all important. (Winslow, 1996)

Programming system and product walkthroughs are a variant of the traditional software walkthrough (Bias, 1991) and *cognitive walkthrough* (Wharton, Rieman, Lewis, & Polson, 1994); they typically fit the definition of a *socio-technical walkthrough* (Herrmann, Kunau, Loser, & Menold, 2004; Herrmann, 2009). Following the terminology of Corbi (1989), studies of programming products and systems can be divided into *static analysis* (reading the code) and *dynamic analysis* (running the code). Both yield important and cognitively disparate understandings of a program. A formal understanding of static analysis provides programmers with the tools needed to both analyze and synthesize solutions from specification, by articulating the process of problem decomposition and stepwise refinement down to the most basic problem solving steps. (Soloway, 1986) Dynamic analysis allows the student to think concretely, explore boundary cases, understand, and visualize the performance of programs.

Software walkthroughs are technically considered a form of peer review called *static testing*, as opposed to formal verification by static analysis. They are distinguished from software inspection in that they permit direct alterations to the programming product (whereas software inspections are made relative to a fixed specification) and do not include measurement criteria for the development process or product. (IEEE, 1998) As software walkthroughs are often conducted in person, *via* videoteleconferencing systems, or using groupware, so are programming system and product walkthroughs. Xiao, Chi, and Yang (2007) describe groupware-based software product development, particularly collaborative development using wikis.

The importance of a walkthrough, with students in the role of clients and end users, is that student feedback may be collected and used. In the classroom, this can serve to provide points of clarification (Soloway, 1986) for homework, term projects, and follow-up (independent study) projects. The process of refinement can also facilitate development of supplemental materials such as collections of exercises. (Clancy & Linn, 1999) Walkthroughs can also serve as aids in restructuring courses to balance between basic problem solving and programming tasks, and in outcomes-based assessment of such changes. (Deek, Kimmel, & McHugh, 1998) Finally, they can aid in design of programming systems components that have high variable amounts of code, such as graphical user interfaces that may be reconfigurable with no programming or may require several times the amount of code as is in the original program. (Karat, Campbell, & Fiegel, 1992; Rowley & Rhoades, 1992; Brooks, 1995)

As with case studies of programming problems and source code, the delivery format of peer review matters. The technology exists to produce videos of programs (and of programming systems or products) being used, modified, and evaluated through walkthroughs. This recapitulates the actual practice of peer review at a higher level of fidelity and interactivity than reading evaluation reports. Similar techniques, applied in basic programming education, have yielded better comprehension, basic skill acquisition, and problem solving transfer. (Deek, Kimmel, & McHugh, 1998; Prabaker, Bergman, & Castelli, 2006; Gaspar & Langevin, 2007) Finally, text documentation itself is also subject to walkthrough and inspection-based peer review. (Novick, 2000)

3.2.4 Software Demonstrations

In addition to algorithm animation, code inspection, and walkthroughs of code with specifications, documentation, and test cases, there are demonstrations of programming products. With proprietary software, these are usually shrink-wrapped commercial off-the-shelf (COTS) products (Ncube & Maiden, 2000), but with open source educational software, they are usually *local builds* – that is, compiled for (or ported to) a particular platform consisting of a computer architecture and operating system, using a specified compiler version and runtime environment. Live demonstrations usually entail choosing the most popular of these, but recorded demos may be prepared for multiple versions and self-study may involve tasks such as performing platform-specific configuration, retargeting a compiler, or even porting applications between programming languages.

Getting and using student feedback is a challenge in both traditional campus environments and distance learning environments. Telepresence and archival systems have proven useful for managing the distance e-learning environment. (Baecker, Wolf, & Rankin, 2004; Rankin, Baecker, & Wolf) In addition, tracking student outcomes can be challenging even in a capstone course – more so for a low-level course in an undergraduate curriculum. Liu (2006) presents a pedagogical application of software project demonstrations as both an assessment tool for instructors and a learning tool for student presenters and their peer reviewers.

3.2.5 Creation of Materials for Instructors and Technology Transfer

Creation of open educational resources entails selection of development and delivery platforms. These are often selected on the basis of introspective student feedback about understanding of the course material and whether they tend to promote attentional focus. (Anderson, Anderson, & Simon, 2004) Other considerations, however, are instructor usability, reusability, and documentation, which are important not only for prospective adopters of prepared materials but for team-taught courses and those that often change instructors.

Authoring systems (Kearsley, 1982) provide not only content for CAI (and by extension, e-learning), but also tie-ins with the other educational materials surveyed in this section. Both the marketing goals and technology transfer goals of authors are furthered by tools that enable them to support open educational resources. This includes technical support – namely, help systems, documentation, demos, and content management systems, as previously described in this paper. It also includes nontechnical support: responding to resource provision requests and following up. The MIT *DSpace* project (Smith, et al., 2003) is a good example of a materials repository that facilitates open access, particularly public information retrieval functions (search, indexing, and ranking). Each of these forms of support plays a useful role in course promotion and solicitation of adoption of materials by individuals at other institutions.

4. Case Studies

Holistic uses of these technologies are then analyzed via case studies in three domains: artificial intelligence, computer graphics, and enterprise information systems. These case studies give an overview of existing materials and practices pertaining to the learning technologies covered. An exploration of technology transfer to college and university-level instructors in the information sciences then follows. This chapter emphasizes courses in computer science, informatics, and computational science and engineering, discussing the presentation of materials that combine mathematical theory, algorithms, software implementations, and data.

4.1 Artificial Intelligence

Open educational materials for the upper-division undergraduate-level course *Introduction to Artificial Intelligence* (CIS 530) and the graduate level *Artificial Intelligence* (CIS 730) course at Kansas State University were developed starting in 2004. Over 30 sets of lecture slides were prepared for 42 lectures. These initially consisted primarily of materials written by the authors of the textbook (Russell & Norvig, 2010), but by 2009 nearly all of the running examples in Russell and Norvig's slides had been converted to PowerPoint animations, except for the Bayesian network examples, which were animated using the author's own software.

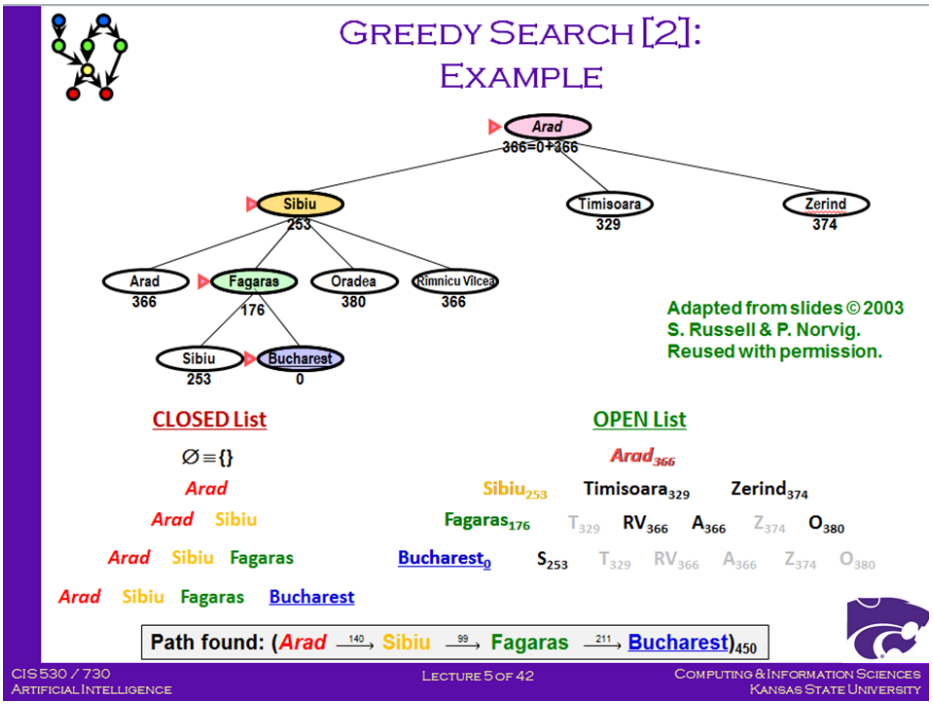

Figure 1. Algorithm animation in Kansas State University CIS 530/730 (*Artificial Intelligence*): A/A* search.

Figure 1 depicts a manual animation developed from a series of static black-and-white images in the second edition of the textbook. The graph being search is shown in a separate slide not pictured here.

Many similar concepts – intelligent agents, state space search, constraint satisfaction search, game tree search, and logical reasoning – were illustrated and usually animated in similar fashion.

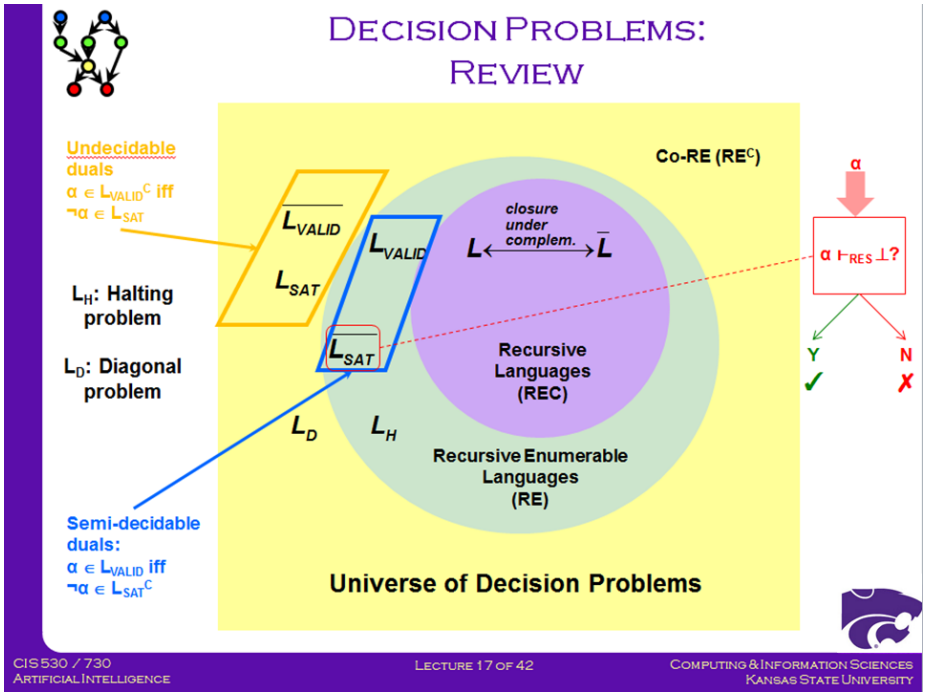

Figure 2. Concept animation in KSU CIS 530/730 (*Artificial Intelligence*): computability of formal languages.

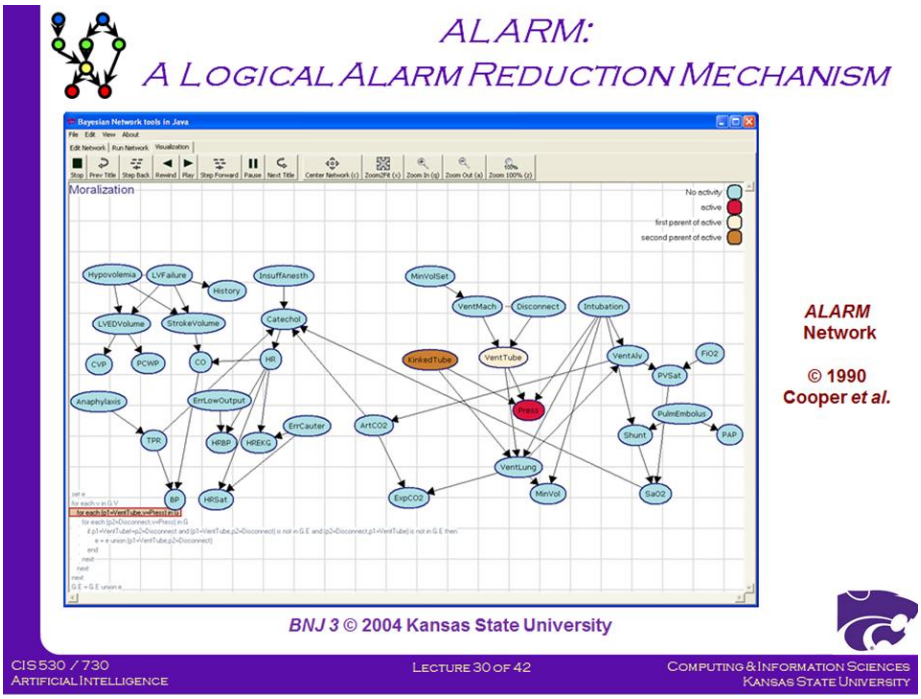

Figure 3. Algorithm animation and software demonstration in KSU CIS 530/730 (*Artificial Intelligence*): exact inference in graphical models of probability using the junction tree algorithm.

Figure 2 depicts an animation of a concept (computability or decidability of various formal languages related to first-order logic). Because most undergraduate and a few graduate students had not been previously exposed to computability theory in an automata theory course, other than an informal introduction to the halting problem, this elective course provided the first introduction to the topic. Students reported in the first years after this figure was drawn by hand that comprehension and retention were low for this concept. A significant improvement was observed after the preparation of this animation.

Figure 3 depicts an animation of the moralization step prior to triangulation and clique-finding in the junction tree algorithm for exact inference in Bayesian networks. This algorithm is known for being difficult to explain effectively in an introductory course in artificial intelligence or visualize in a lecture format, even a recorded lecture. The *Bayesian Network tools in Java* toolkit (Hsu, Guo, Joehanes, Perry, & Thornton, 2003; Hsu & Barber, 2004) was used to animate the junction tree algorithm, among others.

Both PDF and *PowerPoint* versions were created of all slides; similarly, both PDF and *Word* versions were created of all assignments. All of the lectures in this course were recorded and produced using *Camtasia*, and provided to enrolled students *K-State Online*, a content management system based upon *Axio*, and to the public using an Apache-based public mirror web site.

4.2 Computer Graphics

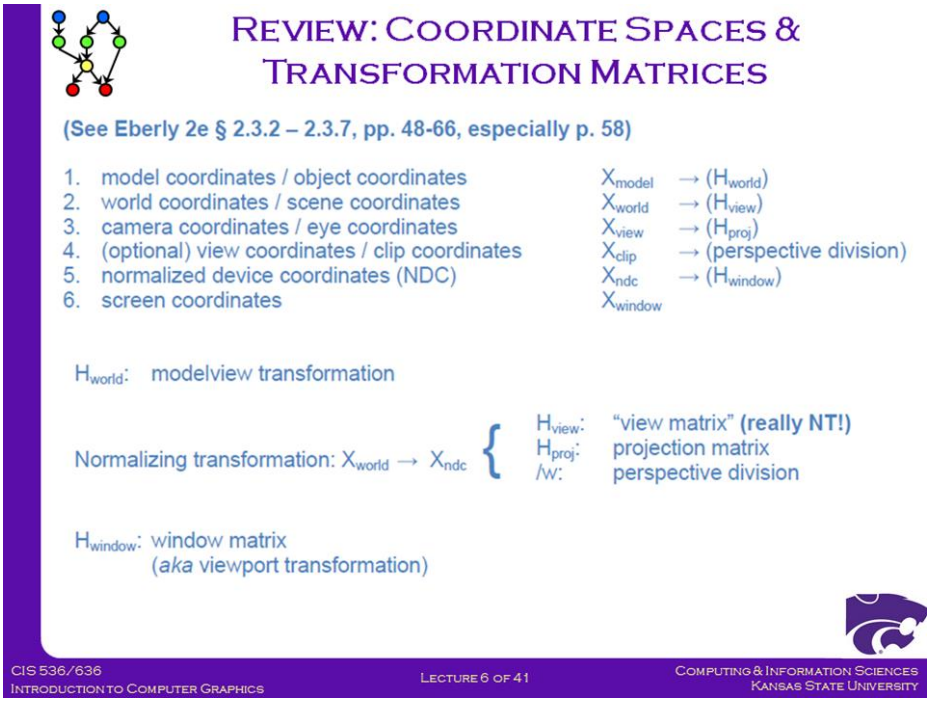

Figure 4. Explanation of concept in KSU CIS 536/636 and 736 (*Computer Graphics*): coordinate systems and transformations for 3-D perspective viewing. Based on Eberly (2006) and Foley *et al.* (1991).

Open educational materials for three computer graphics courses at Kansas State University were developed starting in 2008, and completed in 2011: the upper-division undergraduate-level course *Introduction to Computer Graphics* and the graduate level *Interactive Computer Graphics*. 35 sets of lecture slides were prepared for 42 lectures, plus slides for background refresher lectures for the undergraduate course and advanced topics for the second graduate course. These initially consisted primarily of materials written by the first author of a past textbook for the course (Foley, van Dam, Feiner, & Hughes, 1991), but by 2011 nearly all of the running examples in Van Dam's slides had been converted to PowerPoint animations, except some computer-generated animations which were cited and adapted from *YouTube*.

Figure 4 depicts a key series of concepts in 3-D rendering: the modelview transformation and cumulative transformation matrices that comprise the normalizing transformation for perspective projection. (Eberly, 2006) This slide is accompanied by a voiceover by the instructor explaining each step. It is the culmination of a series of five lectures on viewing that include several figures and animations, each the basis of a screencast. This synopsis slide is one of several that are prepared for students to use for exam review and as a long-term reference.

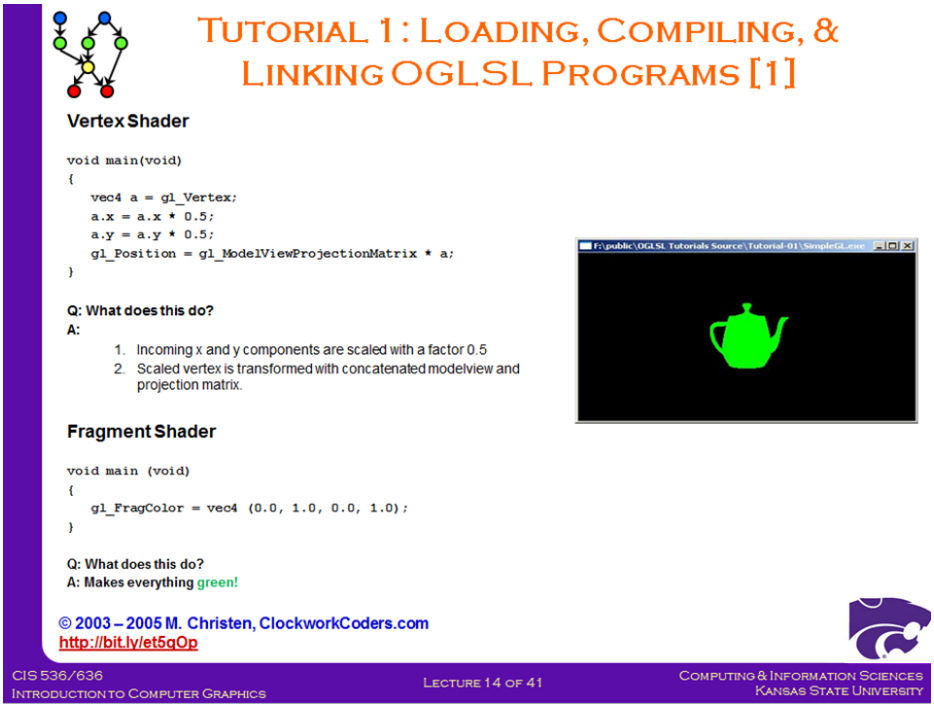

Figure 5. Explanation of code and multimedia walkthrough in KSU CIS 536/636 and 736 (*Computer Graphics*): vertex and pixel/fragment shaders in the OpenGL Shading Language.

Figure 5 depicts inline documentation explaining the function of two toy example shaders: a vertex shader that performs two simple operations and a fragment shader that changes object colors. This is an example of both code explanation (covered in Section 3.2.2) and a third-party programming system walkthrough (covered in 3.2.3).

4.3 Enterprise Information Systems

Both the *Database System Concepts* (CIS 560) and *Enterprise Information Systems* (CIS 562) courses at Kansas State University include course modules in relational database management systems (RDBMS), particularly database design approaches such as entity-relational (E-R) data modeling. One of the challenges in making RDBMS accessible and engaging to undergraduate computer science students is formulating concrete examples of schema design and refactoring without losing the abstract comprehension that students need in order to apply known algorithms for calculating functional dependencies and normalize a data model, analyze existing E-R diagrams to understand what the dependencies entail, synthesize a new model that meets specified criteria, and evaluate alternative designs. (Starr, Manaris, & Stalvey, 2008) Some variability among these concrete examples is necessary: too little and students tend to overfit their mental models to the examples encountered; too much and it becomes difficult to grasp their common features. This leads to the challenge of encouraging students to "design with intention" in RDBMS design and development – a problem analogous to that faced by Gaspar and Langevin (2007) in getting students to "code with intention".

Toward this end, a general-purpose data generator environment was developed in PHP for use in both courses. This data generator takes instances of abstract data types as input. These represent distributions over random variables that are attributes within some relational schema. For example, in an E-R diagram consisting of entities Subscriber and Magazine, and the relationship Subscribes-To, the Gender and Age attributes of Subscriber directly influence Magazine.Genre.

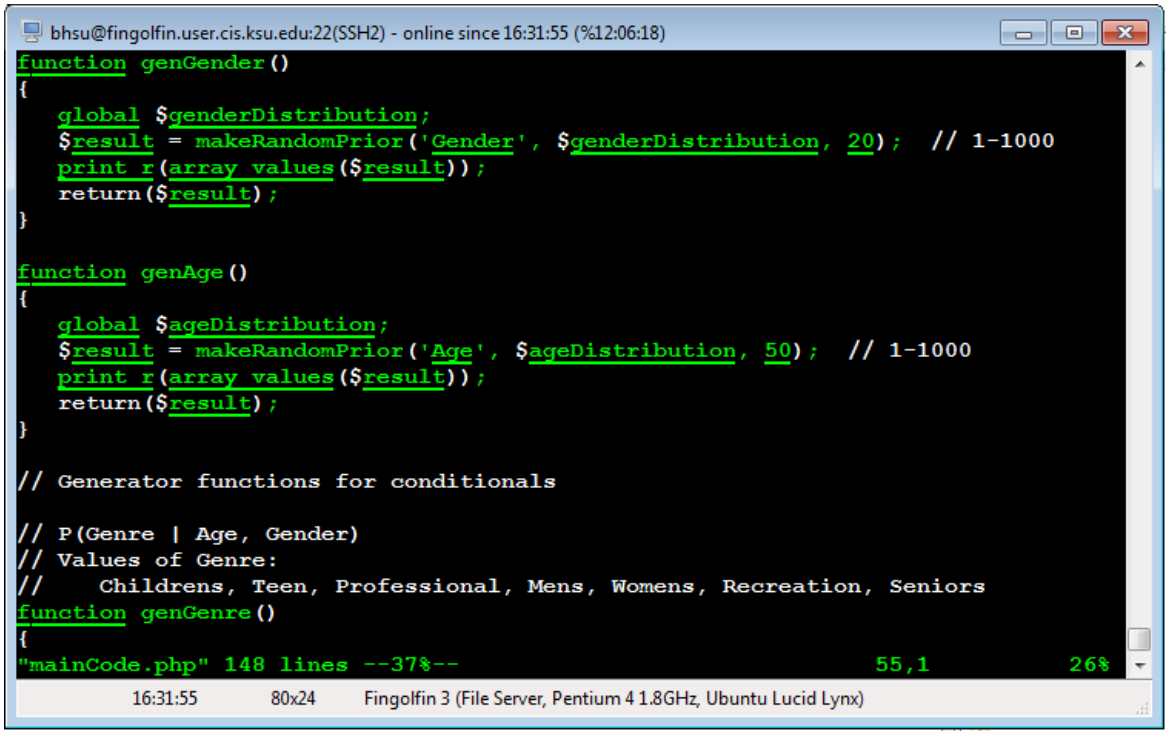

Figure 6. Reading source code and going through programming system walkthrough in KSU CIS 560 (*Database System Concepts*) and CIS 562 (*Enterprise Information Systems*).

Figure 6 shows a partial code listing with inline documentation describing functions for generating prior and conditional distribution. The comments describe the default number of instances generated (independently, for prior distributions) and the conditional probability function being represented, whether the distribution itself is represented by apportionment of likelihood ("roulette wheel" random sampling) or by a conditional probability table. Code listings such as these, given as background material on programming projects, support reading of the source code and code explanation (covered in Section 3.2.2) and a walkthrough of a instructor-supplied programming system (covered in 3.2.3). In addition, generation of data is a process that *itself* admits visualization (covered in 3.2.1), if the algorithm being used to perform random sampling is effective and transparent enough. Finally, the reconfigurability of the above programming system makes software demonstrations (such as those discussed in 3.2.4) feasible.

5. Effective Practices for Encouraging Adoption and Dissemination

Finally, we review effective practices for encouraging dissemination and adoption of lecture materials. These include development practices for comprehensive, well-established open courseware projects that adapt pre-existing content, dating back over 25 years or more. More recent large-scale online courses have been aimed at audiences of tens to hundreds of thousands.

5.1 Adoption

Encouraging course adoption is often a matter of providing a complete set of materials for a course or course unit. In the three case studies above, these are centered around a textbook – Russell and Norvig (2010) for the artificial intelligence course and Eberly (2006) for the graphics course. However, other textbooks including Foley, van Dam, Feiner, & Hughes (1991) have been used with nearly the same lecture slides, video recordings, and code listings, without significant degradation in comprehension or retention.

In addition to holistic adoption of course materials, there is also the possibility that instructors may use specific resources such as lecture slides, exercises, animations, data sets, or particular software. For example, *Bayesian Network tools in Java* or *BNJ* (Hsu, Guo, Joehanes, Perry, & Thornton, 2003; Hsu & Barber, 2004) , the author's open source software package for inference and learning using graphical models, has several thousand downloads per year and has generated dozens of adoption inquiries, but most users are the silent variety and download *BNJ* to use its format conversion utilities (Hsu, Guo, Joehanes, Perry, & Thornton, 2003) or other similar functions that are available at no cost through packages such as *BNJ*.

5.2 Dissemination and Reuse

Lessons learned about dissemination in the techniques and domains surveyed are as follows:

1. **Open content requires (some) open access.** Institutional access limits dissemination to students of research collaborators and other instructors. Some consumers of open educational resources –both administrators and students – may prefer this mechanism for dissemination, because of convenient communication features, or perceived security and privacy. However, a public wiki can easily supplement this kind of channel by mirroring all open content from the course.
2. **Recycle creative input from students and instructors.** Several generations of the artificial intelligence and graphics courses have yielded term projects that themselves provide data for algorithm animation (Hsu & Barber, 2004); code for review (implementing the algorithms described in Section 4.1); additional examples and documentation for walkthroughs (such as shown in the "instructor-supplied code" in Figure 6 of Section 4.3); standalone applications and demonstrations (Hsu, Cunningham, & Hart, 2008); and modules that are contributed back by third-party developers and can be incorporated into a programming system (Hsu, Guo, Joehanes, Perry, & Thornton, 2003) .
3. **Share across media.** *The MASSFORGE Project*, whose goal is to develop "the open-source core of a full-featured artificial life and intelligent agents-based multi-character animation system" (Hsu, Cunningham, & Hart, 2008), began in 2005 as a series of independent study projects. Since then, it has produced several demos on YouTube that illustrate aspects of computer-generated animation: particle systems, 3-D rotations, and repurposing of models (land vehicles and spacecraft). Similarly, these videos demonstrate aspects of game artificial intelligence: flocking and herding models, target acquisition, follow-the-leader behavior, and dynamic path finding. The videos, in turn, have been used to help students brainstorm in the instructor's graphics and artificial intelligence courses, and come up with new ideas for projects.

Bibliography

Abelson, H. (2005). *6.001 Structure and Interpretation of Computer Programs, Spring 2005 - Video Lectures.* Retrieved April 17, 2012, from MIT OpenCourseWare: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-001-structure-and-interpretation-of-computer-programs-spring-2005/video-lectures/

Abelson, H. (2007, May). The Creation of OpenCourseWare at MIT. *Journal of Science Education and Technology, 17*(2), 164-174.

Anderson, R., Anderson, R., & Simon, B. (2004). Experiences with a Tablet PC Based Lecture Presentation System in Computer Science Courses. *SIGCSE '04 Proceedings of the 35th SIGCSE technical symposium on Computer science education .*

Apache Software Foundation. (2011). *OpenOffice: The Free and Open Productivity Suite*. Retrieved from OpenOffice.org: http://www.openoffice.org/

Asay, M. (2007, October 2). *Why choose proprietary software over open source? Survey says!* Retrieved from CNet News: http://news.cnet.com/8301-13505_3-9789275-16.html

Attwood, R. (2009, September 24). Get it out in the open. *Times Higher Education*. Retrieved April 17, 2012, from http://www.timeshighereducation.co.uk/story.asp?storycode=408300

Baecker, R., Wolf, P., & Rankin, k. (2004). The ePresence Interactive Webcasting and Archiving System: Technology Overview and Current Research Issues. *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2004* (pp. 2532-2537). Washington, DC, USA: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (ELEARN) 2004.

Beepa. (2012). *FRAPS show fps, record video game movies, screen capture software*. Retrieved from Fraps: Real-time video capture & benchmarking: http://www.fraps.com/

Ben-Ari, M., Berglund, A., Booth, S., & Holmboe, C. (2004). What do we mean by theoretically sound research in computer science education? *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 230-231). Leeds, United Kingdom: ACM.

Bias, R. (1991). Interface-Walkthroughs: efficient collaborative testing. *IEEE Software*, 94-95.

Blender Foundation. (2012). *blender.org Home*. Retrieved April 17, 2012, from blender.org: http://www.blender.org

Bonaccorsi, A., & Rossi Lamastra, C. (2003). Why Open Source Software Can Succeed. *Research Policy, 32*(7), 1243-1258.

Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition).* Boston, MA, USA: Addison-Wesley Professional.

Carbonell, J. R. (1970, December). AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction. *IEEE Transactions on Man-Machine Systems, 11*(4), 190-202. doi:10.1109/TMMS.1970.299942

Carmichael, P., & Honour, L. (2002). Open Source as appropriate technology for global education. *International Journal of Educational Development*, 47-53.

Chambers, J. A., & Sprecher, J. W. (1980, June). Computer Assisted Instruction: Current Trends and Critical Issues. *Communications of the ACM, 23*(6), 332-342.

Chang, M. Y., Chiu, C.-M., & Hung, S.-S. (2010). A Machinima-like 3D Animation Production System. *10th WSEAS International Conference on Systems Theory and Scientific Computation* (pp. 208-217). Taipei: WSEAS Press.

Chklovski, T., & Gil, Y. (2005). An Analysis of Knowledge Collected from Volunteer Contributors. *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)* (pp. 564-571). Menlo Park, CA, USA: AAAI Press / The MIT Press.

Clancy, M. J., & Linn, M. C. (1999). Patterns and Pedagogy. *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education* (pp. 37-42). New York, NY, USA: ACM Press.

Collins, A., Brown, J. S., & Newman, S. E. (1987). *Cognitive Apprenticeship: Teaching the Craft of Reading, Writing, and Mathematics.* Champaign, Illinois: University of Illinois at Urbana-Champaign.

Corbi, T. A. (1989). Program Understanding: Challenge for the 1990s. *IBM Systems Journal, 28*(2), 294-306.

Creative Commons Corporation. (2011). *History - Creative Commons.* Retrieved April 17, 2012, from Creative Commons: http://creativecommons.org/about/history

D'Anjou, J., Fairbrother, S., Kehn, D., Kellerman, J., & McCarthy, P. (2005). *The Java Developer's Guide to Eclipse, 2nd Edition.* Boston, MA, USA: Addison-Wesley.

Dedrick, J., & West, J. (2003). Why Firms Adopt Open Source Platforms: A Grounded Theory Of Innovation And Standards Adoption. In J. L. King, & K. Lyytinen (Ed.), *MIS Quarterly Special Issue - Workshop on Standard Making: A Critical Research Frontier for Information Systems*, (pp. 236-257).

Deek, F. P., Kimmel, H., & McHugh, J. A. (1998). Pedagogical Changes in the Delivery of the First-Course in Computer Science: Problem Solving, Then Programming. *Journal of Engineering Education*, 313-320.

Depow, J. (2003). Open Source Software: Two learning management systems. *International Review of Research in Open and Distance Learning, 4*(2). Retrieved April 15, 2012, from http://www.irrodl.org/index.php/irrodl/article/view/135/215

desJardins, M., & Littman, M. (2010). Broadening student enthusiasm for computer science with a great insights course. In G. Lewandowski, S. A. Wolfman, T. J. Cortina, & E. L. Walker (Ed.), *Proceedings of the 41st ACM Technical Symposium on Computer science education (SIGCSE 2010)* (pp. 157-161). New York, NY, USA: ACM Press.

DiBona, C., Ockman, S., & Stone, M. (1999). *Open Sources: Voices from the Open Source Revolution.* Sebastopol, CA, USA: O'Reilly Media. Retrieved from http://oreilly.com/openbook/opensources/book/

Dijkstra, E. W., Hoare, C. A., & Dahl, O.-J. (1972). *Structured Programming* (Vols. A.P.I.C. Studies in Data Processing, No. 8). Waltham, MA, USA: Academic Press.

Downes, S. (2007). Models for Sustainable Open Educational Resources. *Interdisciplinary Journal of Knowledge and Learning Objects*, 29-44.

Duffy, P. (2008). Engaging the YouTube Google-Eyed Generation : Strategies for Using Web 2 . 0 in Teaching and Learning. *Strategies*, 119-130.

Eberly, D. H. (2006). *3D Game Engine Design, 2nd Edition: A Practical Approach to Real-Time Computer Graphics.* San Francisco, CA, USA: Morgan Kaufmann.

Edwards, S. H. (2003). Rethinking computer science education from a test-first perspective. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 148-155). New York, NY, USA: ACM.

Fitzgerald, B. (2006). The Transformation of Open Source Software. *Management Information Systems Quarterly*, 587-598.

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1991). *Computer Graphics: Principles and Practice in C (2nd Edition).* Reading, MA, USA: Addison-Wesley.

Free Software Foundation. (2012, February 22). *Various Licenses and Comments about Them.* Retrieved April 17, 2012, from The GNU Operating System: http://www.gnu.org/licenses/license-list.html

Gaspar, A., & Langevin, S. (2007). Restoring "Coding with Intention" in Introductory Programming Courses. *Proceedings of the 8th Conference on Information Technology Education* (pp. 91-98). New York, NY, USA: ACM Press.

GBDirect. (2004). *Benefits of Using Open Source Software.* Retrieved April 17, 2012, from Open Source Software, Free Software and Software Libre: http://open-source.gbdirect.co.uk/migration/benefit.html

Gloor, P. A. (1992). AACE-algorithm animation for computer science education. *IEEE Workshop on Visual Languages 1992 Proceedings*, (pp. 25-31). Cambridge, MA.

Guseva, I. (2009). *Bad Economy Is Good for Open Source.* Retrieved April 17, 2012, from CMS Wire: http://www.cmswire.com/cms/web-cms/bad-economy-is-good-for-open-source-004187.php

Hars, A., & Ou, S. (2001). Working for Free? – Motivations of Participating in Open Source Projects. *Proceedings of the 34th Hawaii International Conference on System Sciences.*

Hasler, B. S., Kersten, B., & Sweller, J. (2007). Learner Control, Cognitive Load and Instructional Animation. *Wiley InterScience*, 713-729.

Herrmann, T. (2009). Systems Design with the Socio-Technical Walkthrough. In B. Whitworth, & A. de Moor, *Handbook of Research on Socio-Technical Design and Social Networking Systems.*

Herrmann, T., Kunau, G., Loser, K.-U., & Menold, N. (2004). Socio-Technical Walkthrough: Designing Technology along Work Processes. *Proceedings of the 8th Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and Practices* (pp. 132-141). New York, NY, USA: ACM Press.

Hertzum, M., & Frøkjær, E. (1996). Browsing and querying in online documentation: a study of user interfaces and the interaction process. *ACM Transactions on Computer-Human Interaction*, 136-161.

Holzinger, A., Kickmeier-Rust, M., & Albert, D. (2008). Dynamic Media in Computer Science Education; Content Complexity and Learning Performance: Is Less More? *Educational Technology & Society*, 279-290.

Hossain, M. S., Rahman, M. A., & El-Saddik, A. (2004). A Framework for Repurposing Multimedia Content. *Canadian Conference on Electrical and Computer Engineering*, (pp. 971-974).

House, C. H., & Price, R. L. (2009). "The Secret Sauce". In C. H. House, & R. L. Price, *The HP Phenomenon: Innovation and Business Transformation* (pp. 216-256). Stanford, CA, USA: Stanford University Press.

Hsu, W. H., & Barber, J. (2004, July 28). Retrieved from Bayesian Network tools in Java (BNJ): http://bnj.sourceforge.net/

Hsu, W. H., Cunningham, D., & Hart, J. (2008, August 6). *The MASSFORGE Project Community: An Intelligent Agent-Based Multi-Character Animation Project.* Retrieved April 17, 2012, from LiveJournal: http://massforge.livejournal.com/

Hsu, W. H., Guo, H., Joehanes, R., Perry, B. B., & Thornton, J. A. (2003, November 5). *Bayesian Network Tools in Java v2.* Retrieved from SourceForge.net: http://sourceforge.net/project/shownotes.php?release_id=195787

Hundhausen, C. D. (2002). Integrating algorithm visualization technology into an undergraduate algorithms course: ethnographic studies of a social constructivist approach. *Computers and Education*, 237-260.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing*, 259-290.

IEEE. (1998). *IEEE Standard for Software Reviews.* New York, NY, USA: Institute of Electrical and Electronics Engineers (IEEE Std. 1028-1997).

Iiyoshi, T., & Kumar, V. (2008). *Opening Up Education: The Collective Advancement of Education Through Open Technology, Open Content, and Open Knowledge.* Cambridge, MA, USA: The MIT Press.

Jonassen, D. H., Howland, J. L., & Marra, R. M. (2011). *Meaningful Learning with Technology.* Boston, MA, USA: Pearson Education.

Karat, C.-M., Campbell, R., & Fiegel, T. (1992). Comparison of Empirical Testing and Walkthrough Methods in User Interface Evaluation. In P. Bauersfeld, J. Bennett, & G. Lynch (Ed.), *Conference on Human Factors in Computing Systems (CHI 1992)* (pp. 397-404). New York, NY, USA: ACM Press.

Kearsley, G. (1982). Authoring systems in computer based education. *Communications of the ACM* , 429-437.

Kerr, I. M., & Bornfreund, M. (2007, June 2). *Open Source Software.* Retrieved April 17, 2012, from Canadian Internet Policy and Public Interest Clinic (CIPPIC): http://www.cippic.ca/open-source/

Knuth, D. E. (1984). Literate Programming. *The Computer Journal, 27*(2), 97-111.

Koning, B. B., Tabbers, H. K., Rikers, R. M., & Paas, F. (2007). Attention Cueing as a Means to Enhance Learning from an Animation. *Wiley InterScience*, 731-746.

Krishnamurthy, S. (2002). Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects. *First Monday*. Retrieved April 15, 2012, from http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1477/1392

Kulik, C.-L. C., & Kulik, J. A. (1991). Effectivenessof Computer-Based Instruction:An Updated Analysis. *Computers in Human Behavior*, 75-94.

Kuniyoshi, Y., Inaba, M., & Inoue, H. (1994). Leaning by Watching: Extracting Reusable Task Knowledge from Visual Observation of Human Performance. *IEEE Transactions on Robotics and Automation* , 799-822.

Lakhani, K., & Hippel, E. V. (2003). How open source software works: “free” user-to-user assistance. *Research Policy, 32*(7), 923–943.

Lawrence, A. W., Badre, A. M., & Stasko, J. T. (1994). Empirically Evaluating the Use of Animations to Teach Algorithms. *Visual Languages*, 48-54.

Lee, A. (2012, April 16). *virtualdub.org: Proof that I had too much free time in college*. Retrieved April 17, 2012, from virtualdub.org: http://www.virtualdub.org/

Lerner, J., & Tirole, J. (2002, June). Some Simple Economics of Open Source. *The Journal of Industrial Economics*, 197–234.

Lerner, J., & Tirole, J. (2004). *The Economics of Technolgy Sharing: Open Source and Beyond - NBER Working Paper No. 10956.* Cambridge, MA, USA: National Bureau of Economic Research.

Lester, J. C., Converse, S. A., Stone, B. A., Kahler, S. E., & Barlow, S. T. (1997). Animated pedagogical agents and problem-solving effectiveness: A large-scale empirical evaluation. *Proceedings of the 8th World Conference on Artificial Intelligence in Education*, (pp. 23-30).

Lin, Y.-T., Chi, Y.-C., Chang, L.-C., Cheng, S.-C., & Huang, Y.-M. (2007). Web 2.0 Synchronous Learning Environment using AJAX. *Proceedings of the 9th IEEE International Symposium on Multimedia* (pp. 453-458). New York, NY, USA: IEEE Press.

Linn, M. C., & Clancy, M. J. (1992, March). The Case for Case Studies of Programming Problems. *Communications of the ACM, 35*(3), 121-132.

Littman, M. (2007). *Computer Scientist Michael Littman shares his videos on Computer Science, Artificial Intelligence, and Family.* Retrieved April 17, 2012, from YouTube: http://www.youtube.com/user/mlittman

Liu, C. (2006). Software Project Demonstrations as not only an Assessment Tool but also a Learning Tool. *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2006)*, (pp. 423-427).

Loidl, S., Mühlbacher, J., & Schauer, H. (2005). Preparatory Knowledge: Propaedeutic in Informatics. In R. T. Mittermeir, *From Computer Literacy to Informatics Fundamentals: International Conference on Informatics in Secondary Schools – Evolution and Perspectives* (Vol. LNCS 3422, pp. 104-115). Heidelberg, Germany: Springer.

Lowood, H. (2006). High-Performance Play: The Making of Machinima. *Journal of Media Practice, 7*(1), 25-42.

Lowood, H., & Nitsche, M. (2011). *The Machinima Reader.* Cambridge, MA, USA: The MIT Press.

Luo, L. (2009). Web 2.0 Integration in Information Literacy Instruction: An Overview. *The Journal of Academic Librarianship*, 32-40.

Mauthe, A., & Thomas, P. (2004). *Professional Content Management Systems: Handling Digital Media Assets.* West Sussex, UK: John Wiley and Sons.

Mayer, R. E. (1999). Multimedia Aids to Problem-Solving Transfer. *International Journal of Educational Research, 31*, 611-623.

Mayer, R. E., & Anderson, R. B. (1992). The Instructive Animation: Helping Students Build Connections Between Words and Pictures in Multimedia Learning. *Journal of Educational Psychology, 84*(4), 444-452.

Mayer, R. E., & Moreno, R. (2002). Animation as an Aid to Multimedia Learning. *Educational Psychology Review*, 87-99.

Mehlenbacher, B. (2002). Assessing the usability of on-line instructional materials. *New Directions for Teaching and Learning*, 91-98.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., . . . Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. *Working group reports from ITiCSE on Innovation and technology in computer science education* (pp. 131-152). New York, NY, USA: ACM.

Ncube, C., & Maiden, N. (2000). COTS Software Selection: The Need to make Tradeoffs between System Requirements, Architectures and COTS/Components. *ACM SIGSOFT COTS Workshop: Continuing Collaborations for Successful COTS Developmen.* New York, NY, USA: ACM Press.

Nourie, D. (2005, March 24). *Getting Started with an Integrated Development Environment (IDE).* Retrieved April 17, 2012, from Oracle Sun Developer Network: http://java.sun.com/developer/technicalArticles/tools/intro.html

Novick, D. G. (2000). Testing Documentation With "Low-Tech" Simulation. In S. B. Jones, B. W. Moeller, M. Priestley, & B. Long (Ed.), *Proceedings of IEEE Professional Communication Society International Professional Communication Conference and Proceedings of the 18th Annual ACM International Conference on Computer Documentation: Technology & Teamwork* (pp. 55-68). New York, NY, USA: IEEE Press.

Obrenovic, Z., Starcevic, D., & Selic, B. (2004). A model-driven approach to content repurposing. *IEEE Multimedia*, 62-71.

O'Hara, K. J., & Kay, J. S. (2003). Open source software and computer science education. *Journal of Computing Sciences in Colleges*, 1-7.

Open Source Initiative. (2006). *The Open Source Definition (Annotated), Version 1.9.* (B. Perens, & K. Coar, Eds.) Retrieved April 17, 2012, from [i]Open Source Initiative: http://www.opensource.org/osd.html

Pane, J. F., Corbett, A. T., & John, B. E. (1996). Assessing Dynamics in Computer-Based Instruction. In M. J. Tauber (Ed.), *Conference on Human Factors in Computing Systems: Common Ground* (pp. 197-204). New York, NY, USA: ACM Press.

Patterson, D. A. (2006). Computer science education in the 21st century. *Communications of the ACM, 49*(3), 27-30.

Plaisant, C., & Shneiderman, B. (2005). Show Me! Guidelines for Producing Recorded Demonstrations. *2005 IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 171-178). IEEE.

PLATO History Foundation. (2011, March 22). *PLATO History: Remembering the Future*. Retrieved April 17, 2012, from http://www.platohistory.org/.

Poindexter, S. E., & Heck, B. S. (1999). Using the in Your Courses: What Can You Do? What Should You Do? *IEEE Control Systems*, 83-92.

Prabaker, M., Bergman, L., & Castelli, V. (2006). An Evaluation of Using Programming by Demonstration and Guided Walkthrough Techniques for Authoring and Utilizing Documentation. In R. E. Grinter, T. Rodden, P. M. Aoki, E. Cutrell, R. Jeffries, & G. M. Olson (Ed.), *Proceedings of the 2006 Conference on Human Factors in Computing Systems (CHI 2006)* (pp. 241-250). New York, NY, USA: ACM Press.

Ramsey, N. (1994, September). Literate Programming Simplified. *IEEE Software, 11*(5), 97-105.

Rankin, K., Baecker, R., & Wolf, P. (n.d.). ePresence: An Open Source Interactive Webcasting and Archiving System for eLearning. *Proceedings of E-Learn 2004.*

Raymond, D. R. (1991). Reading source code. *Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research.* CASCON '91.

Raymond, E. S. (1999). *The Cathedral and the Bazaar.* Sebastopl, CA, USA: O'Reilly Media.

Rieber, L. P. (1990). Animation in Computer-Based Instruction. *Educational Technology Research and Development, 38*(1), 77-86.

Rößling, G., Malmi, L., Clancy, M., Joy, M., Kerren, A., Korhonen, A., . . . Velázquez Iturbide, J. Á. (2008, December). Enhancing Learning Management Systems to Better Support Computer Science Education. (42-166, Ed.) *SIGCSE Bulletin, 40*(4), 1.

Rowley, D. E., & Rhoades, D. G. (1992). The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure. In P. Bauersfeld, J. Bennett, & G. Lynch (Ed.), *Conference on Human Factors in Computing Systems (CHI 1992)* (pp. 389-395). New York, NY, USA: ACM Press.

Ruiz, J. G., Cook, D. A., & Levinson, A. J. (2009). Computer animations in medical education: a critical literature review. *Medical Education*, 838-846.

Russell, S. J., & Norvig, P. O. (2010). *Artificial Intelligence: A Modern Approach.* Boston, MA, USA: Pearson.

Rutgers University. (2012). *rutgers.edu.* Retrieved April 17, 2012, from YouTube: http://www.youtube.com/user/rutgers

Sametinger, J. (1994). The Role of Documentation in Programmer Training. In M. Woodman, *Programming Languages: Experiences and Practice.* Chapman & Hall.

Schaffert, S., Bischof, D., Bürger, T., Gruber, A., Hilzensauer, W., & Schaffert, S. (2006). Learning with Semantic Wikis. *Proceedings of the First Workshop on Semantic Wikis – From Wiki To Semantics (SemWiki 2006).* Budva, Montenegro. Retrieved April 17, 2012, from http://www.schaffert.eu/wp-content/uploads/Schaffert06_SemWikiLearning.pdf

Shank, R. C., Berman, T. R., & Macpherson, K. A. (1999). Learning by Doing. In C. M. Reigeluth, *Instructional-Design Theories and Models: A New Paradigm of Instructional Theory (Vol. II)* (pp. 161-181). Mahwah, NJ, USA: Lawrence Erlbaum Associates.

Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., & Zhu, W. L. (2002). Open Mind Common Sense: Knowledge Acquisition from the General Public. In R. Meersman, Z. Tari, & M. Papazoglu (Ed.), *On the Move to Meaningful Internet Systems 2002: Confederated International Conferences CoopIS, DOA, and ODBASE 2002 Proceedings. LNCS 2519*, pp. 1223-1237. Heidelberg, Germany: Springer.

Singh, V., Twidale, M. B., & Rathi, D. (2006). Open Source Technical Support: A Look at Peer Help-Giving. *Proceedings of the 39th Hawaii International Conference on System Sciences.*

Sleeman, D., & Brown, J. S. (1982). Introduction: Intelligent Tutoring Systems. In D. Sleeman, & J. S. Brown, *Intelligent Tutoring Systems* (pp. 1-10). Orlando, FL, USA: Academic Press.

Smith, M., Barton, M., Bass, M., Branschofsky, M., McClellan, G., Stuve, D., . . . Walker, J. H. (2003). DSpace: An Open Source Dynamic Digital Repository. *Corporation for National Research Initiatives*.

Soloway, E. (1986, September). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29*(9), 850-858.

Spinuzzi, C. (2002). Modeling Genre Ecologies. In K. Haramundanis, & M. Priestley (Ed.), *Proceedings of the 20th Annual International Conference on Documentation (SIGDOC 2002)*, (pp. 200-207).

Spinuzzi, C., & Zachry, M. (2000, August). Genre Ecologies: An Open-System Approach to Understanding and Constructing Documentation. *ACM Journal of Computer Documentation, 24*(3), 169-181.

Starr, C. W., Manaris, B., & Stalvey, R. H. (2008). Bloom's Taxonomy Revisited: Specifying Assessable Learning Objectives in Computer Science. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 261-265). New York, NY, USA: ACM Press.

Stasko, J., Badre, A., & Lewis, C. (1993). Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. *INTERACT '93*, 24-29.

Stephenson, C., Gal-Ezer, J., Haberman, B., & Verno, A. (2005). *The New Educational Imperative: Improving High School - Final Report of the CSTA Task Force.* New York, NY, USA: Computer Science Teachers Association, Assocaition for Computing Machinery.

Stevenson, D. E. (1993). Science, Computational Science and Computer Science: At a Crossroads. In S. C. Kwasny, & J. F. Buck (Ed.), *Proceedings of the 1993 ACM Conference on Computer Science (CSC 1993)* (pp. 7-14). New York, NY, USA: ACM Press.

Stork, D. G. (1999, May/June). The Open Mind Initiative. *IEEE Intelligent Systems and Their Applications, 14*(3), 16-20.

Suppes, P., & Morningstar, M. (1969, October 17). Computer-Assisted Instruction. *Science, 166*, 343-350.

Thatcher, J. D. (2006). Computer Animation and Improved Student Comprehension. *Journal of the American Osteopathic Association*, 9-14.

Tufte, E. R. (1990). *Envisioning Information.* Cheshire, CT, USA: Graphics Press.

Tufte, E. R. (1997). *Visual Explanations: Images and Quantities, Evidence and Narrative.* Cheshire, CT, USA: Graphics Press.

Tufte, E. R. (2006). *Beautiful Evidence.* Cheshire, CT, USA: Graphics Press.

Van Meer, E. (2003). PLATO: From Computer-Based Education to Corporate Social Responsibility. *Iterations: An Interdisciplinary Journal of Software History, 2*, 1-22. Retrieved April 17, 2012, from http://www.cbi.umn.edu/iterations/vanmeer.html

Van Wyk, C. J. (1990, March). Literate Programming: An Assessment. *Communications of the ACM, 33*(3), 361, 365.

Verbert, K., GaSvevic, D., Jovanovic, J., & Duval, E. (2005). Ontology-based learning content repurposing. *Special interest tracks and posters of the 14th international conference on World Wide Web* (pp. 1140-1141). Chiba, Japan: ACM.

von Krogh, G., & Spaeth, S. (2007). The open source software phenomenon: Characteristics that promote research. *The Journal of Strategic Information Systems*, 236-253.

Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). The Cognitive Walkthrough Method: A Practitioner's Guide. In J. Nielsen, & R. L. (editors), *Usability Inspection Methods* (pp. 105-140). New York, NY, USA: Wiley.

Wikipedia. (2012, April 16). *Content Management System.* Retrieved April 17, 2012, from Wikipedia: http://en.wikipedia.org/wiki/Content_management_system

Wikipedia. (2012, April 17). *Informatics (academic field).* Retrieved April 17, 2012, from Wikipedia: http://en.wikipedia.org/wiki/Informatics_(academic_field)

Wikipedia. (2012, March 12). *Open Content.* Retrieved April 17, 2012, from Wikipedia: http://en.wikipedia.org/wiki/Open_content

Wikipedia. (2012, April 13). *OpenOffice.org.* Retrieved April 17, 2012, from Wikipedia: http://en.wikipedia.org/wiki/OpenOffice.org

Wiley, D. (Ed.). (2011). *Defining the "Open" in Open Content.* Retrieved April 17, 2012, from Open Content: http://www.opencontent.org/definition/

Winslow, L. E. (1996). Programming Pedagogy -- A Psychological Overview. *ACM Special Interest Group on Computer Science Education Bulletin*, 17-22.

Xiao, W., Chi, C., & Yang, M. (2007). On-line Collaborative Software Development via Wiki. *Proceedings of the 2007 International Symposium on Wikis* (pp. 177-183). New York, NY, USA: ACM Press.

Yang, A. T. (2001). Computer security and impact on computer scienc eeducation . *Proceedings of the sixth annual CCSC northeastern conference on The journal of computing in small colleges* (pp. 233-246). Middlebury, Vermont, United States: Consortium for Computing Sciences in Colleges.

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. *Proceedings. 25th International Conference on Software Engineering*, (pp. 419-429).

Zhang, D., Zhao, J. L., Zhou, L., & Nunamaker, J. F. (2004). Can e-learning replace classroom learning? *Communications of the ACM*, 75-79.

---

i