Bridging the Gap between the Undergraduate and Graduate Experience in Computer Systems Studies

Lori Carter and Scott Rae
Department of Math and Computer Science
Point Loma Nazarene University
San Diego, California 92106
{lcarter, srae}@ptloma.edu

Abstract

This paper presents the contents of a Special Topics Class used to introduce undergraduate students to the different approach to learning and thinking found in the graduate level Computer Systems environment. During the first half of the semester, the students received instruction in reading and writing technical documents, and had experience with professor-guided self-study. During the second half of the semester, the students presented an idea for a simulator, built it, and used it to conduct simple experiments. After completing the experiments, they were encouraged to brainstorm variations to the algorithms or architectures they simulated, which would improve performance. Student evaluations revealed that for 50% of the students, their interest in graduate school increased as a result of the class.

1 Introduction

The Computer Science undergraduate experience tends to be one of passive learning, where the professors are the producers and the students the consumers. Although in early classes this approach is necessary, and in later classes it is certainly changing [11] [8] [9] there is still a chasm between the undergraduate style of learning, and the graduate school experience, where the student becomes the producer. The disparity between the two levels of education affects both the number of students considering graduate school and the quality of the students entering graduate programs. For some students, obtaining a Ph.D. is not even a consideration. With a bachelors degree in hand, they are competent programmers and tired of sitting in classes. Completely unaware of the challenges and opportunities that await, the graduate school scenario they view as "more of the same" is unappealing. Conversely, if the student has had some exposure to the opportunities for fascinating research, creativity and autonomy found at the Ph.D. level, the tendency is to wonder if their undergraduate education has been adequate to make them a viable candidate for such an undertaking. If these students do pursue graduate education, they are, in fact, often ill-equipped to succeed.

This paper describes a class that was designed to bridge the gap between the undergraduate approach to Computer Systems studies, and the graduate level approach. Where the undergraduate curriculum can be quite broad and structured, the graduate counterpart tends to be more correctly characterized as narrow, directed self-study. Successful graduate students must be able to find the necessary information and produce the desired results with a modicum of guidance from an instructor or advisor. They should have the ability to think "outside of the book", dreaming of new approaches rather than simply embracing what has gone before. In addition, graduate students are expected to be able to read, write, and present technical papers; using very different skills than those taught in English Composition and Speech 101.

In an effort to introduce students to the graduate level approach to learning and thinking, and to improve their confidence and competence with skills required at that level, a Special Topics course was used to introduce Juniors and Seniors to Computer Systems research. During the course of the semester, the students were required to learn Java (their first language was C++), using the directed self-study method. The end product of this class for each of the students was a Java Applet that simulated some component of Computer Architecture or Operating Systems, and which was amenable to simple experimentation. In preparation for building the simulator, each student presented a powerpoint presentation on the proposed project, receiving suggestions on design and presentation skills from their peers. They were introduced to technical papers written on

their topics, and received instruction on, and practice in, reading these papers. The students wrote a proposal and final paper on the chosen simulation using the technical style exemplified by the papers they read. Student evaluations revealed that for most of the students, their interest in graduate school had increased as a result of their experiences. The vast majority of students indicated that confidence in their ability to find the solution independently had dramatically increased. The remainder of the paper discusses the various components of the class, and the responses of the students involved.

2 Syllabus

The semester long Special Topics course was divided into 2 distinct halves, with grade weights assigned as follows:

Phase 1 (individual):

Java, DOS labs	30%
Technical Paper Summaries	15%
Paper/Java/DOS Quizzes	15%
Project Proposal	40%

Phase 2 (group if desired):

Powerpoint presentation	20%
Progress Meetings	30%
Final project (paper, demo)	50%

The content of the first half of the semester was to be completed on an individual basis, while the second half of the semester could be completed in groups of 2 if desired. It is interesting to note that of the 14 students involved in the class, only 3 groups were formed. Most of those that chose to do the work as individuals indicated that the motivation for this choice was to reduce their dependence on others, and to increase their individual learning.

During the first 8 weeks, the students split their classroom time between laboratories and lectures. The lecture time was used for the introduction of Java topics and learning technical communication skills, while the labs focused mainly on self-learning of the Java language. Many of the Java Labs were based on the Sun Java Tutorial [17]. The schedule for lectures and labs is shown in Figure 1. Weeks 9-16 were spent building the simulator, beginning with a powerpoint presentation of their proposed design. Students met with the professor weekly to check progress and discuss changes. At the end of the semester, each student produced a simulation demonstration and final paper detailing their simulator design, experimental results, and future work.

Week	Lecture	Lab
1	Intro to DOS, Java	Intro to DOS, "A Cup of Java" (Sun Tutorial)
2	Intro to Technical Papers	Group Analysis of organization, content, language of 3 systems papers
3	Basis Java Constructs, Discuss paper 1	Sun Tutorial using "Click Me" applet
4	Intro to Java Interfaces, Discuss paper 2	Modifying applets to implement Listeners
5	Java Threads, Layouts Discuss paper 3	Animation Lab
6	Writing Proposal, Finding background papers	Finish Animation Lab Locate background papers
7	Writing a Simulator	Analysis of Turing Machine applet simulation
8	Technical Presentations	proposal draft meetings

Figure 1: Lecture and Lab schedule for the first 8 weeks.

3 Directed Self-Study of the Java Environment

When this class was originally envisioned, an extensive search for a suitable Java textbook was conducted. Although many fine books exist, the conclusion was reached that part of the goal of this class was to encourage life-long self-learning. Consequently, the class was taught without any textbook, using internet-available resources and encouraging self-discovery rather than "spoon-feeding". Initially this was very uncomfortable for the students. However, as they began to work though the on-line tutorials, following links to various "rabbit trails" on related topics, most began to realize that there was a huge amount of information available on the internet, and that it was more current than what is available in a textbook. In addition, they became adept at looking for and finding examples of Java code that performed tasks similar to what they were desiring to do, modifying it to fit their needs. Relying on a textbook would have limited the examples that many students would have considered.

As can be seen from the schedule in Figure 1, Java was introduced to the students through lectures, tutorials, modification and analysis of existing Java Applets and through the experience of original applet creation. The goal of the lectures was to present basic differences between C++ and Java, and to provide introduction to Java Applet requirements, components and capabilities. The use of the Sun Java tutorials guided the student through learning a specific topic, while also providing links to be followed as the student's interest was piqued with regard to

a particular subject. As a part of other labs, students were directed to sources on the internet that exemplified specific features of Java (interfaces, layouts, animation) [2] [12]. Requiring the students to answer questions about, and modify, existing applets, forced them to understand how the features worked, and provided a model for original implementations. Written analysis of an existing simulation [14] provided a framework for putting together a larger project.

4 Reading and Writing Technical Papers

In a course focused on introducing students to the post-graduate style of learning, students must be taught to digest and comprehend technical papers, the primary source of information for most recent technical developments. To develop those skills, students first reviewed three sample papers: "A Comparison of Software Code Reordering and Victim Buffers[1]," "WSClock – A Simple and Effective Algorithm for Virtual Memory Management[6]," and "Efficient Implementation of the First-Fit Strategy for Dynamic Storage Allocation[4]." The papers extended Computer Architecture and Operating Systems concepts that the students had encountered in prerequisite or corequisite courses.

To introduce the reading process, a brochure obtained from the internet [7] outlined basic steps to comprehension, such as underlining, outlining, note-taking, and defining unfamiliar words. A worksheet provided by the professor helped focus attention on five basic sections contained in some form in most papers: the abstract, the introduction, background information, methodology, and results. Students were asked to compare the structures of the three papers to see how different authors might represent the same basic information. Other questions, such as, "*How are the results presented? Text? Tables? Graphs?*" and, "*What is the purpose of the figures in each paper?*" led the students to evaluate both the contents and the presentation of the papers.

For the second phase of this component of the class, students summarized the papers' contents in a one-page write-up, based on the major points they had identified. In the summary, they were asked to begin to mimic the technically-oriented style modelled in the papers they had read. The act of condensing the information to a readable summary helped in data retention. In addition, to encourage good note-taking practices, quizzes were administered that tested the students' ability to extract details and data they were unlikely to simply retain. Students were allowed access to both their summaries and the papers, but a time limit ensured that only information that had been highlighted, noted, or outlined previously, would be helpful to the student in answering the questions.

During the second half of the semester, students had the opportunity to apply what they had learned. In preparation for designing their applet projects, the students were required to research their chosen simulation topic by locating and reading at least two existing papers closely related to the topic. At this point, the students moved from reading technical papers to writing them. After carefully researching the subject, class members wrote draft proposals in research format, detailing both the algorithmic and design requirements for the projects. The selected background papers were referenced to support the projects' relevance. Subsequent to simulation completion, final papers were written, modelled after those the students had studied, complete with figures showing actual simulation interfaces, and graphs detailing the results of their experiments.

5 Presentation

A second reason for not having a course text was economic. The variety of topics presented would have necessitated the purchase of multiple textbooks. Again, the internet was the source of expert information on good presentations. A very readable, practical essay on technical presentations was located [2]. Students were required to read the essay, and make a check-list of features that should exist in presentation. The following class period, the professor gave a powerpoint presentation on what was to be included in the student's presentation, and simulation project. The professor's presentation was critiqued by the class, based on their check-lists.

Finally, each student gave a Powerpoint presentation on their proposed simulations. This was used not only as an opportunity to practice presentation skills, but as an opportunity for peer feedback on their proposed simulator interface designs. The presenter as well as the audience received a grade for the exchange.

6 Simulation and Experiments

As was previously mentioned, the final project for this course was a Java Applet that simulated some aspect of Computer Architecture or Operating Sys-

tems. This project consumed most of the second half of the semester. During this time period, few formal class sessions were held. Instead, class time was filled with individual meetings between the student and professor, much the way an advisor-advisee meeting would occur at the graduate level. Requirements for the simulators were as follows:

- Must be completely original work, completed as a Java Applet (or Japplet)
- Must be amenable to experimentation, allowing user interaction
- Must be on a pre-approved topic
- Must be animated, showing a progression through an algorithm etc.
- Must display some kind of results

6.1 Why Design Simulators?

The final project for this course was a simulation for a variety of reasons. First, much of the systems research is completed using simulators [15]. Clearly, the simulator instructs and informs the user, particularly when the process simulated is depicted visually. A less obvious benefit is available to the constructor of the simulator. The building of the simulator improves the understanding of the topic simulated, and the process of re-creating a technique often has the effect of causing the programmer to consider other, better ways of accomplishing the simulated task.

Second, the simulator provides a target for research. Prior to designing the simulator, the students were required to find (and read) several technical papers on their subject, to improve their expertise. After construction, the simulators became a platform for conducting simple experiments. Although traditional Computer Architecture research is conducted on comprehensive simulators such as Simple Scalar [5], there are certainly arguments supporting the benefits of simpler component simulators [3]. Recent textbooks include access rights to web pages with links to applets supporting the concepts taught in their books[10].

The third reason for this project this is that it was meaningful for the students. The simulations will be used as learning tools for upcoming students in Computer Architecture and Operating Systems courses.

6.2 Simulator Implementations and Experiments

The interfaces for 4 of the simulators created are shown in Figures 2, 3, 4, and 5. Figure 2 shows the interface of a simulator designed for instruction on and experimentation with cache associativity. The user of the simulator inputs a string of memory references and the 2 levels of associativity to be compared. The user can request that the simulation run to completion and display the results (cache misses and total access time), or to step through each cache access. If the latter is chosen, the simulator displays which memory addresses fill the cache lines in each associativity version for each address of the string of references. The user learns how associativity works, which addresses are brought into the cache with a single address request, and can experiment with how different levels of associativity perform with different patterns of memory requests (random, sequential, clustered etc.).

Figure 3 displays an interface for an applet that simulates different CPU Scheduling algorithms. The user inputs tuples of information about several processes entering the ready queue for CPU time. Each tuple includes the process ID, the time of arrival in the system, and the duration of the current CPU burst for the process [13]. The output of the simulation is an animated Gantt chart displaying the CPU use of the processes. In addition, there is a text area reporting the ultimate turnaround time and response time for each process, as well as the throughput for the entire set of processes. The user learns how each of the algorithms work, and can experiment with the effectiveness of each algorithm for improving turnaround time, response time and throughput on different patterns of process requests.

Figure 4 depicts a simulator that can be used to explore different paging algorithms for the virtual memory system. The user can choose between the *FIFO* algorithm for page replacement, and the *Clock* (also known as *Second Chance*) algorithm [13] with 1 or 2 reference bits, and choose the number of frames assigned to the process . In addition, the user can enter a string of pages in the order in which they are required by the process. The output of the simulation is a visual display of each page entering its assigned frame, and the number of page faults resulting from the use of each algorithm. The project allows the user to experiment with different patterns of page accesses for several page replacement algorithms: *FIFO*, and several different implementations of *Clock* (1 bit, and 2 implementations with 2 reference bits).

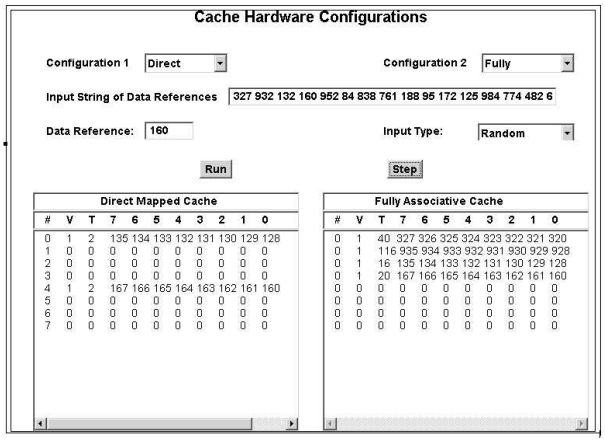

Figure 2: Simulation compares behavior of caches with different levels of associativity

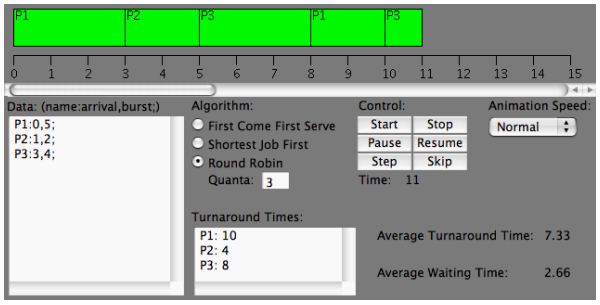

Figure 3: Interface for simulation of CPU scheduling

Although most of the students designed simulations based on ideas with which they were already familiar, some students became intrigued with new ideas based on more advanced material. Figure 5 shows an interface for a simulator demonstrating power dissipation and fan out. The user can choose a circuit and set the value for each of the circuit inputs. The simulation shows the propagation of the values through the gates, coloring outputs with red for 1 and blue for 0. Experiments using this simulator can help provide insight as to which combination of gates provides the optimal circuit in regard to power usage. Another student designed a simulator dealing with victim caches, a topic she became interested in after reading one of the introductory papers early in the semester [1]. Her experiments centered around the size or even existence of the victim cache.

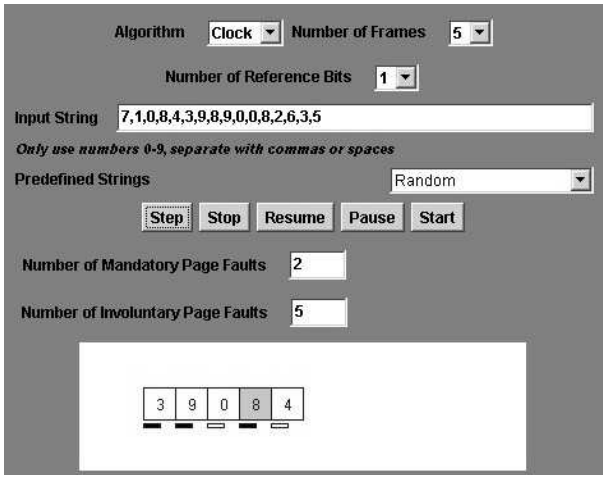

Figure 4: Simulation depicting page replacement

6.3 Progress Meetings

During the weekly progress meetings students were expected to present to the professor evidence of their progress, showing that they had completed the tasks discussed at the last meeting. In addition, they could receive individual help from the professor, and discuss possible implementation changes. Notice that this also served to assure the integrity and originality of the students' code. They were required to implement the suggestions that had been discussed the prior week. Thirty percent of the grade for the second half of the semester was based on their performance at these meetings.

7 Student Response

Overall, student response to the course was extremely positive. All agreed that it was very different from what they had previously experienced. A common thread that ran through student responses to most aspects of the class was that it was harder, but ultimately more enjoyable and satisfying than other Computer Science courses. The students learned a lot about their learning styles, and their ability to work independently. The students were asked to evaluate the learning experience provided by each component of the class on a scale of 1 to 5, where 1 is described as "fairly worthless" and 5 being "an excellent learning experience". Figure 6 shows the average ranking provided by the students for each component.

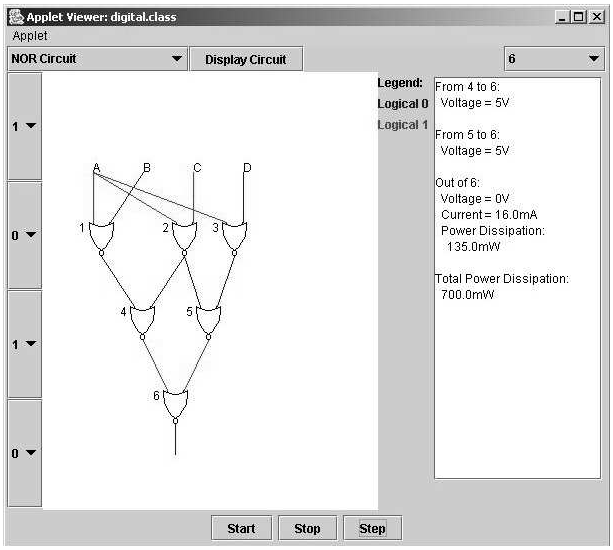

Figure 5: Interface of simulation demonstrating power dissipation

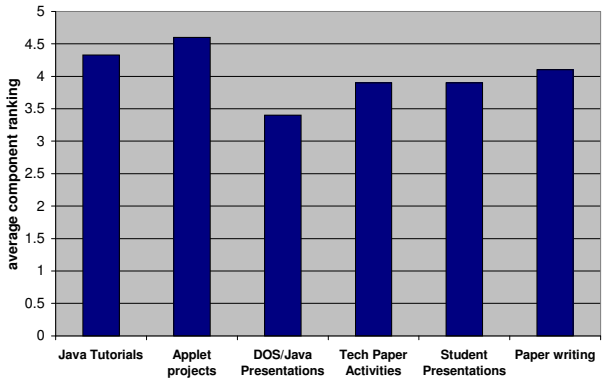

Figure 6: Student rankings of component learning experiences.

It is interesting to note that the DOS and Java presentations, the major lecture component of the class were ranked the lowest by the students, while the active, independent learning experiences (tutorials and projects) were ranked the highest. One aspect of the class, student presentations, involved the students sharing with each other things they had learned in the process of completing applet project assignments. This did not get as high a rating as was expected. When questioned about their responses, the students indicated that, although the material was interesting, it was hard to appreciate and apply when they weren't working on the same exact project.

Another questions that was asked on the evaluation was stated as follows:

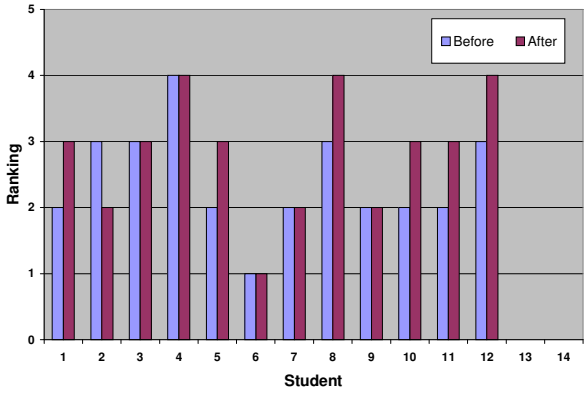

Figure 7: Student rankings on graduate school consideration before and after class

> *This class has introduced a number of things usually found only at the graduate level (reading, analyzing and writing technical papers, directed self-study, thinking more deeply about a subject than what is introduced in a textbook.) We're interested in knowing if this class has in any way influenced your desire to seek further computer science education at the graduate level. Please indicate your interest in graduate school prior to this class, and at this point in the semester on a scale of 1-5, where 1=grad school? No way, never!; 2=well, it isn't completely distasteful; 3=sounds kinds of interesting; 4=I would seriously consider it; 5=count me in!.*

Figure 7 shows the student responses for both before and after. There were 14 students involved in the class, and 2 did not take the survey. The remainder of this section describes in more depth the responses of the students to some of the class components.

7.1 Response to Self-directed Study Leading to Simulation Project

As is clear from the results shown in Figure 6, learning Java through tutorials and applet projects was generally well received. The internet-based assignments (tutorials and code modification) presented initial insights to the students as to where they

could locate the information required to complete the larger simulator project. It was still a giant leap to the implementation of this larger project. Many students were initially frustrated. They were not used to persevering when the information they wanted was not immediately available. Students approached this dilemma in 3 different ways:

1. They found examples of code that did tasks similar to what they wanted to do, copied it in to their project, and acted mystified when it didn't work.

2. They found simple examples of components of the tasks they wanted to accomplish, and tried to understand them and then put them all together, with varying degrees of success.

3. They studied complex examples of similar tasks until they really understood what was happening, and then made an overall plan for the completion of their project using similar techniques.

As can be expected, the first approach led to the most frustration. The more initial understanding that was gained, the greater the success, and ultimately, the less time was spent on completing the project. The students taking approach number 1 also gave up easily on the applet modification projects earlier in the semester. They were far too conditioned to being given a formula, and just plugging in different numbers without taking the time to understand the concept.

Fortunately, the majority of students did not take approach number one, and ultimately all were extremely proud of their successful completion of the assignment. Student comments included "it was like self-learning with a safety net - you could come to the professor if you were really stuck"; "I learned so much more this way. I had something to apply my discoveries to, and I found the answers myself, so the information will stick". Even the students who initially rebelled at the idea of self-learning gratefully received and read (on their own) texts on Java and Java applets and caught up with the rest of the class. Once they realized they weren't going to be spoon-fed, they learned how to feed themselves.

7.2 Response to Technical Paper Reading/Writing

The class members were required to read 3 technical papers over the course of 3 weeks. At first, they were resistant and frustrated. By the last paper, they had gotten quite good at extracting the important information. Several students indicated that they were intrigued with the information found in the papers, information not found in the textbook realm, and wanted to pursue the topics further. Most were delighted that they were able to make sense of the material by the third paper.

The students' writing improved dramatically over the course of the semester, beginning with the summaries and ending with the final paper. Some of the students who weren't terribly gifted in Composition 100 realized they had a talent for clear, concise technical writing. One of the students was coauthor on this paper.

7.3 Response to Simulator Building and Experimentation

Most students found the simulator building to be a very satisfying experience. Each student had the experience of becoming an expert in a small area of Computer Systems. They admitted to recognizing that their initial knowledge of their topic was imprecise. The processes of building the simulators required that they hone their knowledge of the algorithm or architecture. They seemed to enjoy the process of composing hypotheses and determining their validity.

8 Conclusions and Future Work

The Special Topics Class was designed to equip undergraduate students to be self-learners, creative thinkers and competent technical readers, writers, and presenters. It has appeared to be, for the most part, a very successful experiment. Fifty percent of the students responding to the evaluation noted an increased interest in graduate school. More informally, almost all of the students indicated experiencing a heightened level of confidence in their abilities to learn and perform.

The next time this class is taught, the number of lectures on DOS and Java would probably be decreased, freeing up time to do more in the form of tutorials, applet modification and code analysis. More attention would be paid to making the transition from simple applets to an extensive project less abrupt. In addition, the student presentations on "what I learned" would be reserved for the second half of the semester. During this time period, as the simulators were being constructed, students were much more eager to exchange information.

The internet is already an excellent source for current material in teaching computer systems [16]. We plan, in the near future, to add our contribution in the form of a web page with links to all of the student simulation projects for the purpose of teaching simple computer systems concepts.

Although this class involved only 14 students, it could easily scale to a larger class size with the help of graduate assistants. This would actually provide invaluable experience to Ph.D. students who, themselves, would like to pursue academia as a career.

9 Acknowledgements

Paul Kelly, Jeremy Bradney, Richard Trager, Steven Potter and Jenni Sapp contributed interface designs to this paper.

References

[1] I. Bahar, B. Calder, D. Grunwald, "A comparison of software code reordering and victim buffers," *Third Workshop on Interaction between Compilers and Computer Architectures*, October 1998.

[2] K. Boone, "The k-zone," http://www.kevinboone.com

[3] P. Bose, "Simulation in the small: the case for simpler models and testcases in computer architecture education and research", *Proceedings WCAE 2002, Workshop on Computer Architecture Education,* Vancouver, BC, June 10, 2000

[4] R. P. Brent, "Efficient Implementation of the First-Fit Strategy for Dynamic Storage Allocation," *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 3, July 1989.

[5] D. Burger, T.M. Austin and S. Bennett, "Evaluating future microprocessors: the SimpleScalar tool set," Tech. Rept. CS-TR-96-1308. Univ. of Wisconsin-Madison, July 1996.

[6] R. Carr and J. Hennessy, "WSClock - a simple and effective algorithm for virtual memory management," *Proceedings of the Eighth ACM Symposium on Operating System Principles,* December 1981.

[7] M. J. Hanson, D. McNamee, "Efficient reading of papers in science and technology," http://www.cse.ogi.edu/~dylan/efficientReading.html

[8] J. Herath, S. Ramnath, A. Herath, S. Herath, "An active learning environment for intermediate computer architecture courses," *Proceedings WCAE 2002, Workshop on Computer Architecture Education,* Anchorage, AK, May 26, 2002

[9] W. T. Hsu, "Experiences integrating research tools and projects into computer architecture courses," *Proceedings WCAE 2000, Workshop on Computer Architecture Education,* Vancouver, BC, June 10, 2000

[10] J. Kurose, K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, Addison-Wesley, 2003

[11] I. Papaefstathiou, C. P. Sotiriou, "Read, use, simulate, experiment and build: an integrated approach for teaching computer architecture," *Proceedings WCAE 2002, Workshop on Computer Architecture Education,* Anchorage, AK, May 26, 2002

[12] R. Sebesta, *Programming the World Wide Web,* Addison-Wesley, 2002

[13] A. Silberschatz, P. Galvin, G. Gagne, *Applied Operating Systems Concepts,* John Wiley and Sons, 2000.

[14] S. Skinner, "Turing machine simulator applet," http://www.wap03.informatik.fh-wiesbaden.de/weber1/turing/tm.html

[15] C. Weaver, E. Larson, T. Austin, "Effective support of simulation in computer architecture instruction," *Proceedings WCAE 2002, Workshop on Computer Architecture Education,* Anchorage, AK, May 26, 2002

[16] W. Yurcik, E. Gehringer, "A survey of web resources for teaching computer architecture," *Proceedings WCAE 2002, Workshop on Computer Architecture Education,* Anchorage, AK, May 26, 2002

[17] "The java tutorial," http://www.java.sun.com/docs/books/tutorial