

# 1 Results of Previous NSF Support

I have been an investigator on two NSF grants in the past five years, both as principal investigator:

**Award Number: ILI-LLD DUE-9750953**

**Amount: \$ 100,000**

**Duration: September 1, 1997 – August 31, 1999**

**Title: An Electronic Laboratory for Operating Systems and Computer Networks**

**Award Number: ILI-CCD DUE-9752165**

**Amount: \$ 202,568**

**Duration: September 1, 1998 – August 31, 2001**

**Title: A Web-Based Electronic Laboratory for Operating Systems and Computer Networks**

The original LLD proposal was for three years. As the LLD program was being phased out, the project was funded for one year and I was asked to submit a proposal to CCD to cover the remaining two years. Because of the uncertainty in the funding, it was difficult at first to find qualified graduate students to work on the project, delaying the start of the work. Each of the grants was given a one-year extension and the work proceeded as planned, with the extended schedule. Since the two grants make up a single project, I will describe them together.

The purpose of these projects was to redesign the undergraduate curriculum in operating systems and computer networks to make them more hands-on. A number of interactive simulations were developed to allow students to experiment with various computer science concepts in a laboratory-like environment. One of these projects is described in detail in Section 3.2.

The first part of the project concentrated on the development of tools for implementation and testing of the animations. An animation package called JOTSA, Java On Time Synchronous Animation[6], was completed soon after the project started. This package allows animations to proceed at a predictable rate, independent of the underlying hardware. It also allows events to be fired when an animation completes. This tool was used to produce an animated event-driven simulation of data link layer protocols in which the timing of the animations controlled the event timing.

Another tool, JELI (Java Experimental Laboratory Interface)[9], has recently been completed. This package provides a number of Java utilities for developers of interactive Java simulations including statistical distributions and remote logging.

About a half dozen animations are in various stages of completion and have been used in my courses. The completed ones are available on the web and have been reported at SIGCSE conferences. These have been used at several other institutions. The work is scheduled to be finished by August, 2001 and an outside evaluation of the project is currently in progress.

To date, there have been four publications based on this work and at least two more are expected.

- “The JOTSA animation environment,” *31st Hawaii Intl. Conf. on System Sciences* (Jan 1998) pp. 655–664.
- “Empirical exploration in undergraduate operating systems,” *Proc. 30th SIGCSE Technical Symposium on Computer Science Education*, (1999) pp. 311-315. (with K. Robbins)
- “Experimentation with bounded buffer synchronization,” *Proc. 31st SIGCSE Technical Symposium on Computer Science Education*, (2000) pp. 330–334.
- “Remote logging in Java using Jeli: A facility to enhance development of accessible educational software,” *Proc. 31st SIGCSE Technical Symposium on Computer Science Education*, (2000) pp. 114–118.

A complete description of the work completed to date including downloads of the software is available on the web[10].

## 2 Goals and Objectives

Experimentation is the centerpiece of the traditional scientific method. Experimental exploration can provide new insights, eliminate unproductive approaches and validate theories and methods. Walter Tichy [11] cites several examples in the systems software area where commonly-held assumptions were shown to be false by careful empirical studies. Computer science, as a discipline, has a notoriously poor record in the area of empirical validation. Several studies of computer science publications [12, 14] have shown that the percentage of papers providing no substantiation for claims that needed experimental verification is much higher than in other disciplines in either the hard or soft sciences.

In some respects the weak computer science tradition in experimentation is not surprising given the discipline's rapid emergence and constant pressure for change. There is also little evidence for movement towards a more empirically grounded approach. While undergraduate computer science majors may take a laboratory science as part of their general education requirement, few computer science programs provide students with empirical experience in their discipline.

The 1991 curriculum report of the ACM [13] indicates that each area of the curriculum should employ the processes of theory, abstraction and design. It lists the following elements of abstraction:

- Data collection and hypothesis formation
- Modeling and prediction
- Design of an experiment
- Analysis of results

These elements are often lacking in both the undergraduate computer science curriculum and in traditional computer science research literature. Laboratory courses such as physics and courses in statistics typically required of a computer science major are not designed primarily for computer science majors and so the experimental design and analysis are rarely applied to topics in computer science. Students view these courses as less important than their computer science courses and often learn just enough to get through the final exam.

This project was motivated by the current work described in Section 1 in which some of the simulations required students to formulate hypotheses and test them. Most of the students had no idea how to do this [7], even though they had taken both Technical Physics (which included a lab) and two probability and statistics courses. Something was clearly lacking in our current curriculum.

The purpose of this project is to design a curriculum in which undergraduate computer science majors can learn experimental design and analysis in a computer science context. The course materials will be centered around a textbook that has two types of chapters, content chapters that develop experimental design concepts and project chapters that apply these concepts to particular

problems. The project chapters will take two forms: paper case studies and simulation.

The paper case studies will take an experimental problem that has been addressed in the literature and analyze how concepts developed in the content chapters were applied in the selected paper. An annotated version of the actual paper will be available in an appendix or on the web, but the paper will be summarized with peripheral ideas removed in the textbook version. The students will then design and run a similar experiment on an available system. While many research papers explore the performance of high end systems not readily available to undergraduates, the student's measurements would typically be made on a standard workstation. The students will be provided with utilities for making actual measurements on standard computing systems. Emphasis in these case studies will be on collection and analysis of data. The analysis of the research paper provides a much needed introduction to computer science research in the undergraduate curriculum.

The most extensive projects will be based on simulations, because this venue allows the student the most flexibility in forming and testing hypotheses. Interactive simulations will be developed that allow students to experiment on computer systems. The simulations will be written in Java and be runnable through a standard browser. A typical project would have the student form a hypothesis, make a prediction, design an experiment, collect data, and analyze the results.

The goals of this proof of concept project are threefold. First, to produce a draft of a textbook. Secondly to identify about a dozen interactive simulations and papers from the computer literature that can be used for the projects. Thirdly, to develop a sufficient number of these projects to offer a prototype course. Since projects typically take 2 weeks to complete, 6 projects would be sufficient for the first time the course is offered. The most time consuming part of this process will be the development of the simulations. Tools produced from previous work will expedite this development.

The individual projects could be used by themselves to supplement the material in a traditional computer science course or used together with the textbook in a separate course in experimental design and analysis. The materials will be disseminated through presentations at computer science conferences and via the web.

If the pilot project is successful, the remaining projects will be developed and a full textbook will be produced and published along with a CD-ROM containing the simulation software. A total of at least 12 projects would allow the course to be offered in successive semesters without the need to repeat projects.

### **3 Detailed Project Plan**

The material for this project falls into three general categories: the content chapters of the textbook, the simulations, and the paper case studies.

#### **3.1 The Textbook**

The textbook, tentatively titled *Experimental Design and Analysis in Computer Science*, will be organized along the lines of my book, *Practical UNIX Programming* [5], with content chapters interspersed with project chapters. The content chapters will contain material similar to that found in standard courses in probability, statistics and experimental design. These chapters stand on their own. The purpose of the content chapters is to present the results of the underlying theory, not to provide proofs or derivation of results, unless these add to the understanding of the material. The content will be driven by what is needed to do the projects. Projects will be chosen so that all of the important content material of the textbook will be needed.

The book *The Art of Computer Systems Performance Analysis* by Jain[3] will be used as a model for the these chapters. The Jain text is meant as a graduate level text and reference, and is not readily readable by undergraduate students. I believe that I can make the required material accessible to computer science undergraduates, assuming only a background in single variable calculus.

The project chapters will be of two types. Projects that use a simulator will describe how the particular simulator works as well as suggest projects that can be done using the simulator. Projects that revolve around case studies of research papers will present the research papers in a form accessible to the students.

During this proof of concept phase, it is envisioned that a detailed outline of the text will be created, along with project chapters for 4 interactive simulations and 2 case studies.

## **3.2 Interactive Simulations**

The motivation for this project was my experience with having students attempt to design experiments to test a hypothesis about process scheduling algorithms. As part of the previous work described in Section 1, several interactive simulations were produced and used in an undergraduate course in operating systems. One of the projects given to the students involved design and analysis of an experiment and the results showed the need to further develop this material in the undergraduate curriculum. Another simulation had students draw a conclusion from an experiment they ran, and their answers also showed a fundamental lack of understanding of analysis of experiments.

The purpose of this section is to describe the components of an interactive simulation and give an example of one that has already been developed and used. Good interactive simulations are time consuming to produce, with much of the time needed for the design of the user interface, the input of configuration information, and the output of results. Tools already produced for the simulator described here work will reduce the time needed to produce new simulations.

### **3.2.1 The Process Scheduling Simulation**

This simulation was developed for a previous project and will be one of the simulations used for this project. It will serve as a model for the other simulations. It can be run over the web through a standard browser or can be downloaded and installed locally. It provides a testbed for experimentation with process scheduling algorithms. The simulator interface shown in Figure 1 makes it easy to run experiments on collections of processes with different scheduling parameters and to compare such statistics as throughput and waiting time. The operation of the simulator is described in a SIGCSE paper [7] and a user's guide [10].

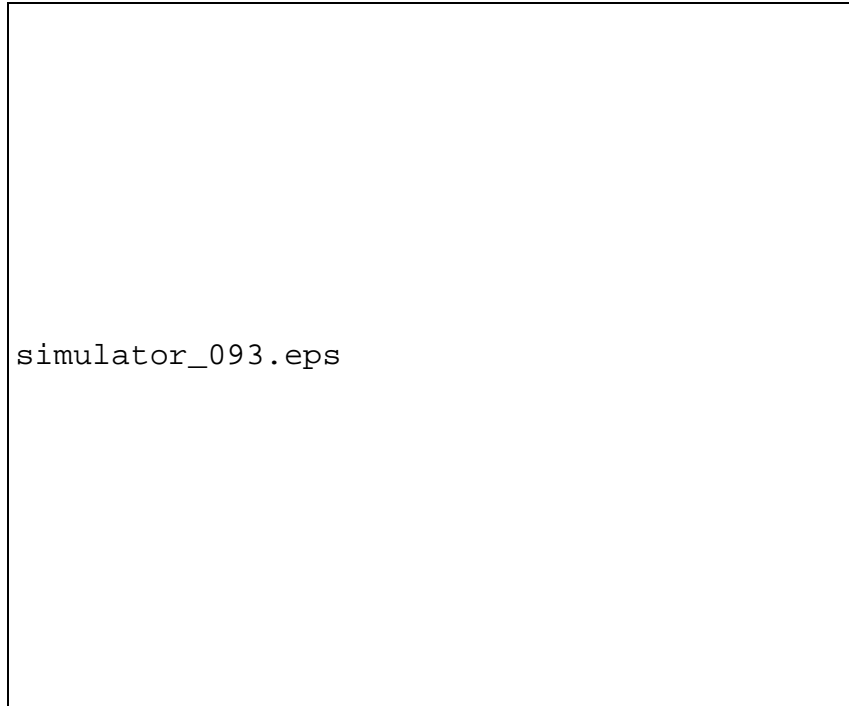


Figure 1: A view of the process scheduling simulator window.

An *experimental run* is the result of running the simulator with a given collection of processes with a given scheduling algorithm. A set of processes is described by giving the number of processes and the distributions for the interarrival times, CPU burst times and I/O burst times. Several sets of such processes can be used for one experimental run. An experimental run produces statistics such as average waiting time and CPU utilization. An example of a file, `myrun.run`, which specifies an experimental run is shown in Figure 2. It specifies 30 processes, 15 with long CPU bursts and 15 with shorter ones.

Typically an experiment consists of a number of experimental runs in which only one parameter varies, such as the scheduling algorithm, the number of processes, or the CPU burst distribution. For example, a file called `myexp.exp` containing the information shown in Figure 3 specifies an experiment with two experimental runs using the same processes, but the first uses the scheduling algorithm FCFS (first-come/first-served) and the other uses SJF (shortest job first). This experiment produces the tables shown in Figure 4 and the graphs in Figure 5.

```

name myrun
comment Two types of processes
algorithm SJF
seed 5000
numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration uniform 10.0 15.0
cpuburst constant 10.0
ioburst uniform 10 20
basepriority 1.0
numprocs 15
firstarrival 0.0
interarrival constant 0.0
duration constant 4.0
cpuburst constant 1.0
ioburst uniform 10.0 20.0
basepriority 1.0

```

Figure 2: The file myrun.run which describes 30 processes.

```

name myexp
comment Two runs
run myrun algorithm FCFS key "FCFS"
run myrun algorithm SJF key "SJF"

```

Figure 3: The file myexp.exp which describes 2 runs using the processes from Figure 2.

| Description | Algorithm | Time   | Processes | Finished | CPU Utilization | Throughput | Turnaround Time | Waiting Time |
|-------------|-----------|--------|-----------|----------|-----------------|------------|-----------------|--------------|
| myrun_1     | FCFS      | 257.05 | 30        | 30       | .96104          | .116711    | 215.05          | 176.41       |
| myrun_2     | SJF       | 256.90 | 30        | 30       | .96158          | .116777    | 127.53          | 88.91        |

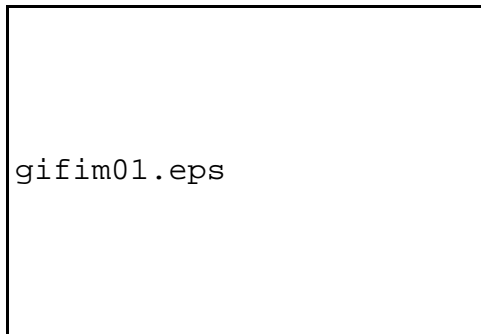
| Description | Algorithm | Turnaround Time |         |         |       | Waiting Time |         |         |      |
|-------------|-----------|-----------------|---------|---------|-------|--------------|---------|---------|------|
|             |           | Average         | Minimum | Maximum | SD    | Average      | Minimum | Maximum | SD   |
| myrun_1     | FCFS      | 215.05          | 169.60  | 257.05  | 31.45 | 176.41       | 138.82  | 202.24  | .66  |
| myrun_2     | SJF       | 127.53          | 32.60   | 256.90  | 61.10 | 88.90        | 1.82    | 229.82  | 2.32 |

Figure 4. Tables generated by the simulator.

One of the ways to use this simulator in an operating systems course is to propose a hypothesis and have the students construct an experiment to test it. Here is an example:

**Hypothesis:** *If the CPU utilization is high and if all other factors are kept the same, the relationship between the number of processes and the average waiting time is linear. Devise tests to support or disprove the hypothesis.*

The students would then design specifications for an experiment, run the simulator, and analyze the results.



gifim02.eps

gifim03.eps

Figure 5. Graphs and Gantt charts.  
Above: Average waiting time graph.  
Right: Gantt charts for the two runs.

### 3.2.2 Interactive Simulation Implementation

All simulations will be implemented in Java so that they can be run from a standard browser without any additional software installation. They can also be run locally on any machine containing a Java runtime system by downloading the simulation software from the web. A number of difficulties arise in the implementation of interactive simulations. A tool called JELI, Java Experimental Laboratory Interface, developed in previous work helps to solve these problems.

**Probability:** Jeli contains routines for portable random number generation and standard distributions such as constant, uniform, exponential, and normal. If a seed is specifically given, results on different runs can be repeatable. Otherwise, a different seed will be generated for each run. Repeatability is important when students are designing an experiment so that they can search the parameter space for results, e.g. to get a given CPU utilization, and then rerun the experiment with logging turned on to produce a printout.

**Configuration:** Jeli contains a facility for handling files containing keyword-value pairs where the value may be an integer, a floating point value, a string, or a probability distribution. These are used to parse the experimental run and experiment files in the process scheduling simulator.

**Input from a browser:** An applet run from a browser cannot access local configuration files directly. Jeli contains a facility for inputting configuration values using cut and paste when a

required input file cannot be found. Using cut and paste is usually sufficient for these types of experiments.

**Logging:** Jeli contains utilities for creating HTML files to record the results of the simulations. Applets run from a browser are not allowed to write to the local machine, so Jeli contains a facility for storing the log on the machine from which the applet was loaded. The log file can be accessed through a standard browser and then printed locally.

**Comments:** The Jeli logging facility allows the student to enter notations into the log, allowing the log file to be used like an experimentalist's notebook. The notations are automatically put in HTML format.

**Graphs:** The Jeli graphing facility contains utilities for producing standard graphs and Gantt charts. Axes are automatically scaled to fit the data. Once displayed, the user can interactively fine tune the graph parameters such as scaling and color. These graphs can be automatically converted to GIF format with links to them put in the log file so the graph can be displayed and printed using a standard browser.

**High density widgets:** Interactive simulations often need to display a large amount of data in a small space. The Jeli package has a number of widgets designed for this purpose.

### 3.3 Paper Case Studies

One of the goals of this project is to incorporate computer science research into the undergraduate curriculum. Analysis of a current or classical research paper is one way to do this.

The paper case studies can:

- illustrate the methods discussed in the content chapters
- provide a basis for students to do real experiments which they can analyze
- introduce students to the computer science literature

Network performance measurements are particularly well suited for case studies because these measurements are of interest to students. They have all had experience and frustration with network delays. The SIGCOM and SIGMETRICS conferences and the ACM/IEEE Transactions on

Networking regularly have papers that can be used effectively as the foundation for the case studies.

A typical case study chapter starts with the background needed to understand the paper, outlines the premise and main results of the paper, discusses the techniques used, and describes a project which students can do using the same techniques. An annotated copy of the paper will be available in an appendix or on the web.

One such paper is an analysis of **pathchar** [2], a tool for determining network characteristics [1]. The **pathchar** program attempts to determine link characteristics such as bandwidth and latency using a method similar to **traceroute**. It uses the time-to-live field of IP packets to get information about the first  $n$  hops along a given path. The values derived from using two consecutive values of  $n$  give information about a given hop.

The difficulty in using time-to-live is that different packets undergo different delays at a given host because of queuing delays. If one packet has a delay of 20 milliseconds for 4 hops and another has a delay of 25 milliseconds for 5 hops, then the added delay of 5 milliseconds assumes that the second packet also had a delay of 20 milliseconds for the first 4 hops. This, in fact, will not be the case, as successive packets with 4 hops will experience a variety of delays.

The problem is handled by sending a large number of packets with the same number of hops and determining the minimum delay for that number of hops. If the calculation requires an average of 10 packets to achieve a delay close to the minimal value for one hop, then one would expect that 100 packets would be needed to find one which obtained the minimal delay for two hops. The number of packets needed is an exponential function of the number of hops. If 10 packets are needed for one hop and the path length is 10 links (a small number for typical traffic outside a given LAN) the experiment would require  $10^{10}$  probes, an unreasonable number. Problems addressed in the case study paper include a determination of the number of probes required, an analysis of the types of probes that should be used (how many of each packet size) and the accuracy obtained for a given measurement.

The ideas are accessible to students even if they have not taken a networks course. The first

half of the paper describes the measurements that were made and analyzes the results using some standard methods which should be familiar to the students from the content chapters of the book.

Some of the methods used are:

- scatter plots
- least squares fit
- filtering
- cumulative distributions

The paper also discusses a number of important issues that are often ignored:

- accuracy of the underlying model
- error estimates
- problems relating to taking the difference of similar quantities
- the effect of noise
- number of samples required

The **pathchar** tool used in the paper is freely available for many Unix systems including Linux, Solaris, and Free BSD. Other than one of these Unix systems, no special equipment is needed to make the measurements. Students with an Internet connection can make measurements from their home system to any other host on the Internet.

The project chapter describing this paper would start with a brief introduction describing IP packet routing including a discussion of the time-to-live field. The chapter would then explain how **traceroute** works and encourage the students to experiment with it. Next, the problem of determining latency and bandwidth of the links along a path will be discussed along with the ideas behind using time-to-live to determine the characteristics of a single link.

The **pathchar** utility will be introduced along with some exercises using both **traceroute** and **pathchar**. These exercises will allow students to find destinations that may be interesting to analyze in the full project. This will be followed by a discussion of the problems associated with using **pathchar** as a motivation for the paper.

The paper will be discussed with emphasis on the contributions made by the paper and the

techniques used to determine whether **pathchar** can give valid results in a given situation. One of the issues addressed is the tradeoff between using more packet sizes or larger samples for each packet size. This could form the basis for a project in which the accuracy of several different methods are compared. The total time to do the experiment would be fixed, and students could vary the mix of samples taken. Interval estimates for each experiment could be obtained using the methods discussed in the paper and the results compared. The paper found that with two different paths, it was better to use more different packet sizes than to use more probes per size.

### **3.4 Deliverables**

The deliverables will consist of the material needed to offer the course a single time and a report of the results of the evaluation of the prototype project.

An outline for the text will be created along with 6 projects. The first project would be assigned about 2 weeks into the course and each project will take about two weeks for the students to complete, so 6 projects should be sufficient for a 15-week course. Four of the projects will be interactive simulations and two will be based on papers from the computer science literature. Two of the simulations have already been produced, the one described in Section 3.2 and another one completed for the previous project on bounded buffer synchronization [8].

The textbook outline will be submitted to a publisher for consideration. I have already been in contact with a Prentice Hall editor about this project.

### **3.5 Future Plans**

If this proof of concept project is successful, a full development EMD proposal will be submitted to NSF. The full textbook with CD-ROM containing at least 12 project chapters will be developed and published. Each project chapter which describes one of the interactive simulations, will have several suggested projects for each simulator. This would allow the same simulator to be used in successive semesters without duplication of assignments.

This book is designed to be used in a one-semester senior level computer science course. About

6 projects could be covered in a typical semester, allowing the instructor to choose those topics most suited to student needs at a particular institution, or to vary the projects from semester to semester..

The prototype book and projects would be field-tested at a number of institutions.

## **4 Impact**

This innovative curriculum will better prepare students to be able to design experiments and analyze the results. These skills have been neglected in traditional computer science programs. It is hoped that this course will encourage the creativity of the students and convince more students to continue their education towards a masters or PhD degree.

## **5 Experience and Capability of the Principal Investigator**

The principal investigator has been teaching computer science for many years and has been active in computer science curriculum development. A previous NSF ILI grant resulted in the publication of a computer science textbook [5]. He has developed two Java packages which will be employed in the current project, JOTSA [6] for doing animation and JELI [9], a collection of utilities for interactive simulations.

This interest in using interactive simulations for teaching originated with a project that used XTANGO for a microprogramming simulator [4]. This early work led to the two recent NSF grants described in Section 1. Three presentations at SIGCSE [7, 8, 9] describe this work.

## **6 Evaluation Plan**

Evaluation of this proof-of-concept project will take two forms.

The individual projects will be available on the web, allowing instructors at other universities to freely incorporate them into other courses. Those downloading the software or using it on line will be asked to fill out a short form evaluating the individual projects they are using. They will

also be asked to participate in a more comprehensive evaluation of the project when it moves into the full development stage.

The prototype course is tentatively scheduled to be offered at UTSA as a special studies course in Fall 2002. A list of expectations will be distributed to the students at the beginning of the course and a questionnaire will be administered at the end of the course to see how well these expectations were met. Student input will be critical in fine tuning the individual projects as well as in determining the scope of the full development.

## **7 Time Schedule**

Project begins: Summer 2001.

Summer 2001: Begin the outline of text (chapter and section titles).

Read literature, looking for papers for the case studies.

Fall 2001: Create the first case study prototype and the first new simulation prototype.

Spring 2002: Present a paper at SIGCSE 2002.

Summer 2002: Create second paper prototype and second new simulation prototype.

Prepare the draft of the book for student use.

Fall 2002: Teach first prototype course.

Refine all materials.

Spring 2002: Present paper and workshop at SIGCSE 2003 and prepare full development grant.

## **8 Dissemination of Results**

A web page for this project will be set up when the project begins and progress will be available on this web site. Two papers will be written describing this project and submitted to the annual SIGCSE conferences in 2002 and 2003. An outline for the book and all of the software will be available on the web for download or on CD ROM. A workshop on teaching using this material will be proposed for SIGCSE 2003.

## References

- [1] A. B. Downey, “Using pathchar to estimate Internet link characteristics,” *ACM SIGCOM 99*, pp. 241–250
- [2] V. Jacobson, “pathchar—a tool to infer characteristics of Internet paths,” presented at the Mathematical Sciences Research Institute (MSRI); the slides are available from <ftp://ftp.ee.lbl.gov/pathchar>, April 1997.
- [3] R. Jain, *The Art of Computer Systems Performance Analysis, Techniques for Experimental Design, Measurement, Simulations, and Modeling*. John Wiley & Sons, Inc., 1991.
- [4] S. Robbins, “A microprogramming animation,” *IEEE Trans. on Education*, vol 41, No. 4, 1998, pp. 293–300.
- [5] K. A. Robbins and S. Robbins, *Practical Unix Programming: A Guide to Concurrency, Communication, and Multithreading* Prentice Hall, 1996.
- [6] S. Robbins, “The JOTSA Animation Environment,” *31st Hawaii Intl. Conf. on System Sciences*, 1998, pp. 655–664.
- [7] S. Robbins and K. A. Robbins, “Empirical exploration in undergraduate operating systems,” *Proc. 30th SIGCSE Technical Symposium on Computer Science Education*, 1999, pp. 311–315.
- [8] S. Robbins, “Experimentation with bounded buffer synchronization,” *Proc. 31st SIGCSE Technical Symposium on Computer Science Education*, 2000, pp. 330–334.
- [9] S. Robbins, “Remote logging in Java using Jeli: A facility to enhance development of accessible educational software,” *Proc. 31st SIGCSE Technical Symposium on Computer Science Education*, 2000, pp. 114–118.
- [10] S. Robbins, “A web-based electronic laboratory for operating systems and computer networks,” available at <http://vip.cs.utsa.edu/nsf/index.html>.
- [11] W. F. Tichy, “Should computer scientists experiment more?” *IEEE Computer*, May 1998, pp. 32–40.
- [12] W. F. Tichy, et al., “Experimental evaluation in computer science: A quantitative study,” *J. Systems and Software*, Jan 1995, pp. 1–18.
- [13] A. B. Tucker, et al., *Report of the ACM/IEEE Joint Curriculum Task Force*, December, 1990, available at <http://www.acm.org/education/cur91/>.
- [14] M. V. Zelkowitz and D. R. Wallace, “Experimental models for validating technology,” *IEEE Computer*, 1998, pp. 23–31.