Development of a Set of "Closed Laboratories" for an Undergraduate Computer Science Curriculum

Wm. A. Wulf
University of Virginia

Computer Science Report No. TR-91-17
September 23, 1991

Department of Computer Science
University of Virginia
Charlottesville, Va. 22903

A Proposal To The

National Science Foundation

Development of a Set of "Closed Laboratories" for an Undergraduate Computer Science Curriculum

Principal Investigator

Wm. A. Wulf

Professor of Computer Science
Department of Computer Science
University of Virginia

Abstract

As a discipline, Computer Science has seen many dramatic changes in its brief history. Through new textbooks and an evolving curriculum, the *content* of undergraduate Computer Science education has for the most part kept pace with these changes. But the *pedagogy* has hardly changed. Not only is that pedagogy out of date -- it is profoundly wrong. It emphasizes individual skill in writing short programs in a dead language, from scratch. This emphasis is the antithesis of that needed by a contemporary computing professional.

The objective of the present proposal is to support the development, evaluation and export of a new curriculum and supporting materials focused around the *practice* of computing -- especially in the first two years. We believe that this pedagogical shift has, by far, the highest leverage for improving Computer Science education.

The thrust of our new curriculum is based upon a core sequence of closed laboratories. Lecture materials and content will be supportive of the laboratories rather than the other way around. While content can always be improved, we cannot expect to make quantum improvements. Laboratories, by contrast, have been largely neglected in this "laboratory science". Thus, the output of this curriculum development can have a significant national impact on the Computer Science education.

Project Description

Introduction

As a discipline, Computer Science has seen many dramatic changes in its brief history. Through new textbooks and an evolving curriculum, the *content* of undergraduate Computer Science education has for the most part kept pace with these changes. But the *pedagogy* has hardly changed. Not only is that pedagogy out of date -- it is profoundly wrong. It emphasizes individual skill in writing short programs in a dead language, from scratch. This emphasis is the antithesis of that needed by a contemporary computing professional.

The objective of the present proposal is to support the development, evaluation and export of a new curriculum and supporting materials focused around the *practice* of computing -- especially in the first two years. We believe that this pedagogical shift has, by far, the highest leverage for improving Computer Science education.

The Computer Science Department at the University of Virginia is extensively revising its undergraduate curriculum to support this shift in emphasis. The process started in the summer of 1990 and detailed planning is currently underway; every faculty member is involved. Some of the new curriculum responds to situations unique to Virginia, while other aspects reflect the changing content of the field, as manifest in the recently released ACM/IEEE Curriculum [Tuk91]. However, some aspects of this revision are fundamental; among these is an early introduction of more, and more relevant, mathematics, and a commitment to its use in all subsequent courses.

The subject of this proposal, however, is the most fundamental aspect of the revision, namely the development of a practice-centric pedagogy, involving the introduction of modern software development techniques from the very beginning, and supported by a coordinated set of "closed laboratories" as advocated in [Den89] and [Tuk91].

The distinction between a "closed" and an "open" laboratory is common in some disciplines such as physics and chemistry, but is relatively rare in Computer Science, c.f. [Nap90, Pen90, Wen90, Ols83]. An open laboratory is the way that programming assignments are now given at most universities; the instructor assigns a programming problem which the students are then free to work on at their own pace. A closed laboratory, by contrast, is one that meets at an assigned time and place. Assignments are to be worked on, often by small groups, primarily during the assigned time -- although a report of the results of an experiment may often be done later. Frequently special equipment is available in the lab, and may have been specially set up for a particular experiment. An instructor is available during the laboratory period who guides the conduct of the experiments.

In the context of the objectives for our curriculum, current pedagogy has many drawbacks. The usual approach teaches students

- to construct relatively small programs, at most a few hundred lines,
- to construct them in a programming language that is rarely used, even in our own graduate education,
- to construct them afresh for each assignment or course,
- to construct them largely without the use of helpful tools, and
- to construct them in isolation or occasionally as a member of a small group, and
- to believe that "any solution" that works is acceptable.

This is the antithesis of real-world practice where programs are thousands or millions of lines long, are often extensively modified and maintained rather than merely constructed, are manipulated in a tool-rich environment, where work is almost always a team effort, and where the form of a solution has a profound impact on future cost and performance.

The thrust of our new curriculum is based upon a core sequence of closed laboratories focusing on the *practice of computing*. Lecture materials and content will be supportive of the laboratories rather than the other way around. We feel that, while content can always be improved, we cannot expect to make quantum improvements. Laboratories, by contrast, have been largely neglected in this "laboratory science". Thus, the output of this curriculum development can have a significant national impact on the Computer Science education.

Although some laboratories have been implemented ([Epp91], [Nap90], [Ols83], [Pen90], [Pra78], [Ree90], [Rob91]), they are primarily isolated examples limited to a single course. By contrast, we propose to make the closed lab experience the cornerstone of our entire undergraduate experience with the introduction of five new and carefully designed closed labs. We are aware of at least one parallel effort with somewhat similar goals to our own but with a different emphasis, [Tur91]. This parallel effort, however, makes laboratories subordinate to a "breadth first" reorganization of the content as opposed to our subordination of content to a practice-centric, "laboratories first" approach.

Lest anyone misunderstand, however, we are *not* talking about a training-oriented curriculum. If anything, we are designing a far more fundamental, more mathematical curriculum than is common today. The difference is that we believe those fundamentals must be introduced early, in a context that makes sense, and must be exercised repeatedly and consistently in subsequent courses. Thus, for example, we require a full year of discrete mathematics in the second year[1], and have upgraded all the subsequent courses to utilize and reinforce this material. A very pleasant side effect of this approach is that students will be far better prepared for a computing career.

To ensure the success of the effort we will employ several techniques:

- To ensure that the laboratories have a component of realism, we have established an Industrial Steering Committee to provide guidance and help in creating these new laboratories. We are particularly gratified that *every* company that we have approached about this has been extremely enthusiastic, and has committed to high-level technical staff participation.

- We expect to host a workshop of distinguished Computer Science educators in the Spring of 1992 to vet our ideas.

- We will hold annual meetings, beginning in the third year, to assist other universities in adopting the results of our developments.

- We will apply a number of techniques to evaluate the impact of our approach -- including

    - annual review by peer educators,
    - review by an industrial advisory committee,
    - constant feedback will be sought from our students and teaching assistants, and

[1] The first year is filled with normal engineering core material.

- regular evaluation questionnaires to our graduates and their employers.

The following sections outline more of the details of our plan.

Context

At the University of Virginia Computer Science is in the School of Engineering and Applied Science. This provides a context in which the Department functions; specifically we feel that it is appropriate that our graduates be oriented toward the pragmatic, "engineering" aspects of Computer Science. Good engineering is rooted in solid mathematics and science, and a grounding in these fundamentals is essential, especially during the first two years.

A set of principles are guiding the development of the new curriculum structure. Among these are: the early introduction of relevant mathematics and its application in subsequent courses, earlier introduction to software engineering concepts and their application in subsequent courses, a conscious effort to provide more "out of classroom" opportunities for the undergraduate to interact with faculty and graduate students, and the use of at least one closed laboratory in virtually every semester of the first three years.

All of these will be evident in the later description of the curriculum. However, central to our considerations is the focus on the practice of computing, and the use of closed laboratories to achieve this focus. The next section describes our rationale for this decision.

Rationale for the Closed Laboratories

Many universities, including the University of Virginia, offer a software engineering course that exposes the student to the problems of developing and maintaining realistic systems. Unfortunately, such courses have limitations. For example, they are typically taken late in a student's career and so other courses cannot presume the knowledge or skills gained from them. Further, the materials and skills taught in such courses are often excessively powerful for the small programs with which the student is familiar -- and students may develop an unfortunate negative attitude about the relevance of the material as a result. Perhaps the most significant disadvantage is that they cram too much material into a single semester, thus failing to cover each topic adequately -- or leaving out important topics.

Closed laboratories are not a panacea for these problems, but, combined with revised course content, they have the potential to ameliorate them greatly. For those courses that include a laboratory, the course content, organization, timing, and goals will be driven by the laboratory exercises rather than the other way around. To ensure that this approach yields truly innovative and valuable educational opportunities, each laboratory will be different but each will be characterized by one or more of the following themes:

Exercises will be based on large hardware/software systems.

> The goal will be to use existing systems that are many thousands of lines in length and unfamiliar to the student. The use of such systems will force the student to understand and *use* the notion of abstraction in a way that small programs cannot. Typical exercises will involve modification and enhancement of a system, thus promoting skill development appropriate for system maintenance activities. The laboratory experiments can thus stress analytic as well as synthetic techniques and skills.

Exercises will be based on industrial hardware/software systems.

To the extent possible within normal corporate proprietary precautions, we will solicit real hardware/software products from our industrial partners for use in the laboratories -- or at least simplified, "sanitized" versions of such systems. The intent is to ensure realism and relevance of the exercises to the extent feasible.

Exercises will employ the "studio" model.

In other design disciplines, such as architecture, students benefit from exhibiting their designs to and being critiqued by their peers. We refer to this as the *studio* model. The traditional open laboratory model encourages students to work independently, and they never get the opportunity to see and benefit from alternative solutions whether good or bad. The closed laboratory concept includes provision for students to present and discuss all types of information produced during software development.

Exercises will employ the "case-study model."

Many innovative teaching approaches in other disciplines make extensive use of the systematic study of existing artifacts. We refer to this as the *case-study* model since the method is exemplified by the case studies used in business schools. Computer science can make very effective use of the approach by, for example, providing the design documents and source code for a large compiler to the students and engaging in an analysis of the compiler. Such analysis might involve comprehension of large sections of the code, writing programs to analyze the system, modifying the system in some creative way, or perhaps instrumenting the system to obtain metric or performance data.

These activities are impractical in a conventional classroom structure and in open laboratories but are immediately possible with our notion of a closed laboratory. We note that there are many benefits to case studies of this type. Students do not learn merely the internal workings of a specific system such as a compiler. They learn about the entire domain of such applications, how to analyze a large system, how to understand an existing system, and how to modify an existing system.

Exercises will involve stimulating applications.

In a closed laboratory, it is possible to have computer controlled devices, such as sound generators, robot arms, motors, and advanced graphics, that are impractical in an open laboratory. Use of such equipment in laboratory exercises adds immense stimulation and interest, and promotes effective education in ways that are well recognized but hard to quantify -- the "fun factor." Such facilities are quite impractical in an open laboratory for obvious reasons, but we believe will add measurably to the retention of students in science and engineering generally, and Computer Science in particular.

Exercises will provide exposure to modern CASE tools.

The closed laboratory provides the essential environment to permit students to gain experience with advanced CASE tools. Such tools cannot be made universally available in open laboratories because the cost would be prohibitive. Even if cost were not a factor, it is not possible to learn to use such tools effectively without considerable instruction and support during the learning activity. This is provided by the closed laboratories. Finally, the motivation for the use of such tools does not exist in the usual, small open laboratory exercise, but coupled with team experiments on the realistic systems discussed earlier, the closed laboratories can provide ample motivation.

Student activities will be as part of a group.

As noted in [Pen90], the typical open-laboratory exercise encourages, even forces, a student to work independently. Unless required specifically as part of an assignment, students working together are viewed as cheating. Some effort is expended in classic software engineering courses to organize group projects but group interaction takes place outside of the classroom. The open laboratory model is not conducive to learning skills associated with working in a group; it assumes such skills despite the fact that no opportunity for the required training exists.

The closed laboratory is very effective in dealing with this problem. Interaction is encouraged but controlled. Groups members can be observed interacting amongst themselves but interaction between groups is not only discouraged but practically limited by an appropriate "office-like" laboratory physical structure. Yet because there are many separate experiments in one lab, and because a student experiences many laboratories, they will be part of many lab groups over time. Even without the many other benefits, we would have adopted closed laboratories as the center piece of our curriculum because they stimulate group work. As Penny and Ashton say [Pen90]:

> *"The greatest advantage of this laboratory-style approach is that the instructor can be unequivocal in encouraging students to work together."*

Naps [Nap90] gives an interesting, but orthogonal, characterization of the laboratories that he has developed at Lawrence University; characterizations such as this will be used to help balance the total collection of experiments.

- *Discovery Labs*, in which the student is presented with phenomena and encouraged to deduce properties of an algorithm, data structure, or whatever.

- *Improvement Labs*, in which the student is given, say, an algorithm and is asked to improve some property of it.

- *Comparison Labs*, in which a student is asked to analyze different algorithms or data structures and make relative judgements about them.

- *Reinforcement Labs*, in which the student experiments with implementations of particularly subtle concepts in order to reinforce their understanding of them.

So, for all these reasons, we have made the closed laboratories central to our new curriculum. Having said that, there remains the hard work of developing and evaluating them, and putting it in to a form that can be exported. That is why we are submitting this proposal.

The Larger Curriculum Context

Since the curriculum is still being designed, it is not appropriate to present all of its details since they will surely change. Rather, the next two subsections describe the process by which we are designing the curriculum and courses, and give a snapshot of our current thinking.

The Curriculum

As noted earlier, a set of principles guided our development of the curriculum structure. Among these were: the early introduction of relevant mathematics and its application in subsequent courses, earlier introduction of software engineering concepts and their application in subsequent courses, a conscious effort to provide more "out of classroom" opportunities for the undergraduate to interact with faculty and graduate students, and a focus on the practice of computing -- to the latter end, we plan at least one closed laboratory in virtually every semester of the first three years.

From our initial considerations emerged decisions that may be controversial; among these is the decision to use C++ as the principal teaching language, and to adopt a common programming environment for the entire program. These decisions arose from a variety of considerations, but principal among them was our desire to teach modern computing methods using practical tools and applications *from day one*.

The content of the curriculum is not especially unusual, and generally follows the most recent ACM/IEEE recommendations [Tuk91]. Nor is the particular packaging of the material apparently unusual from a superficial inspection; the packaging, however, *was* carefully chosen to get both mathematical and engineering skills introduced early and exercised repeatedly in subsequent courses.

The resulting structure of "core" courses is shown in Figure 1; this core provides the common prerequisites for later specialization. In addition to the core, we require a normal engineering sequence (calculus, physics and chemistry, etc.), three additional mathematics courses, six humanities courses, and five advanced Computer Science courses (chosen from a list of 12). A senior thesis is also required of all engineering students; we will combine this with a senior seminar in which various issues, such as professional ethics, will be covered. Figure 2 illustrates a sample student program[2]. In both Figures we have used the following course identifiers; boldface and asterisk denote an associated laboratory:

1CS*	*Introduction to Computer Science*	**2SW***	*SW Development Methods*
2DM	*Discrete Mathematics for Engineers*	**2PDR***	*Program and Data Representation*
2DL	*Digital Logic (a joint CS/EE course)*	**3ALG***	*Algorithms*
3CA	*Computer Architecture*	3DM	*Discrete Mathematics for CS*
3SW*	*Software Engineering*		

[2] In Figure 2 we have taken some license with course names and numbers to avoid lengthy explanation.

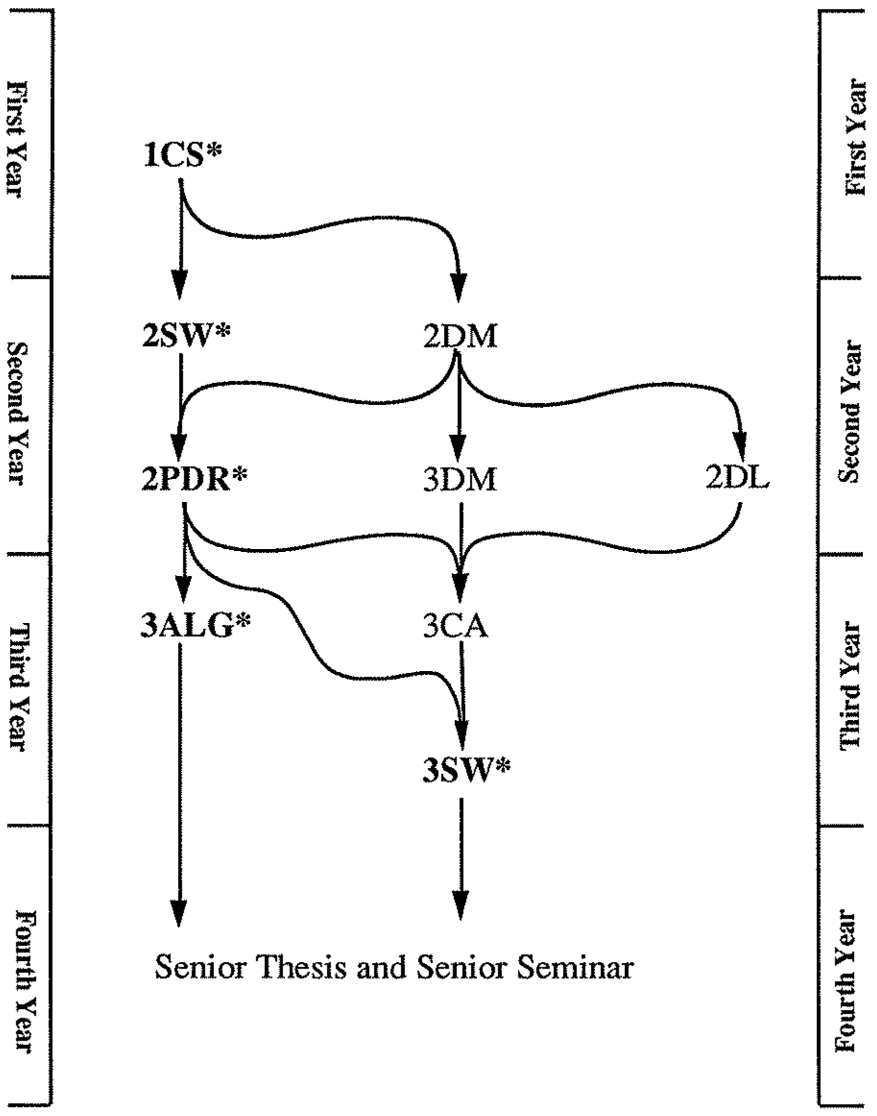


Figure 1: The New Prerequisite Structure

The Courses

We are in the process of refining the specific courses, thus we can only describe the rationale and process -- and give a snapshot of where we are.

Each of the courses shown in Figure 1 was given a preliminary specification in Fall 1990, and is being refined by a subcommittee of 3-5 faculty at the present time. The subcommittees consist of those faculty most knowledgeable about the subject area, and who are most likely to teach it for the next few years. Each committee also has members serving on committees for adjacent courses in the prerequisite hierarchy in order to smooth the interfaces between them. Unlike the common "list of topics," each course specification was given in terms of a set of "postconditions" defining

- the skills that a subsequent course could presume, and
- the knowledge that a subsequent course could presume

Figure 2: A Sample Student Program

	First Semester			**Second Semester**	
Math 101	Calculus I	4	Math 102	Calculus II	4
Chem 141	General Chemistry	4	Phys 104	General Physics	4
Engr 110	Engr. Graphics	3	**CS 1CS***	**Intro to CS**	3
Engr 160	Engr Concepts	3	----	Science Elective	3
H101	Humanities	3	----	Gen Ed Elective	3
		17			17
	Third Semester			**Fourth Semester**	
Math 205	Calculus III	4	CS 3DM	Discrete Math II	3
CS 2DM	Discrete Math I	3	Engr 208	Digital Logic	3
Phys 205	General Physics II	4	----	Science Elective	4
CS 2SW*	**SW Dev Methods**	4	**CS 2PDR***	**Prg & Data Rep.**	4
----	Humanities Elective	3	----	Gen Ed. Elective	3
		18			17
	Fifth Semester			**Sixth Semester**	
CS 3ALG*	**Algorithms**	4	**CS 3SW***	**Software Engr**	4
CS 3CA	Computer Arch	3	CS ----	CS Elective	3
Math 310	Probability	3	CS ----	CS Elective	3
----	Engr Elective	3	EE 333	Microcomputers	3
----	Gen Ed Elective	3	----	Gen Ed Elective	3
		16			16
	Seventh Semester			**Eighth Semester**	
Hum 401	Senior Thesis I	3	Hum 402	Senior Thesis II	3
CS 4S1	Senior Seminar I	1	CS 4S2	Senior Seminar II	1
CS ----	CS Elective	3	CS ----	CS Elective	3
CS ----	CS Elective	3	CS ----	CS Elective	3
Math 308	Linear Algebra	3	Math 551	Scientific Computing	3
----	Gen Ed Elective	3	----	Gen Ed Elective	3
		16			16

Moreover, the elements of this skill/knowledge set were rated as one of:

mastery which implies a thorough understanding of the concept or technique and an ability to identify and select this alternative as the best choice in a particular situation without prompting from the instructor.

knowledge of, or skill in the use of which implies a good understanding of the concept or technique, and the ability to apply it to a situation when guided to do so by the instructor.

exposure implies a modest familiarity with the concept or technique, at least to the extent of knowledge of its existence and utility, but no skill in unaided application.

To illustrate the approach, the following is an edited version of the preliminary specification of 2PDR, the course on program and data representation. This course is similar to a first course in computer organization, but emphasizes the general notion of representation in the context of machine representation of HLL concepts.

CS 2PDR Representation of Program and Data (4 hrs credit)

Data structuring techniques and the representation of program and data structures during program execution. Operations and control structures and their representation during program execution, including recursion and dynamic storage allocation. Representations of numbers and arithmetic operations, arrays, records, hashing, stacks, queues, trees, graphs, and related concepts. A simplified computer, with an associated assembly language, is used to illustrate the run-time representations of programs and data. 3 lecture and 2 lab hours per week.

Prerequisites: CS 2SW CS 2DM

Postrequisite skills and knowledge:

	skills	knowledge
mastery	number conversion simple assembly language	machine organization representation of scalars arrays & records pointers simple control structures stack frames & recursion
knowledge	debugger - static/dynamic performance measurement choice of data structure non-rectangular arrays, ...	representation of add'l structures lists, strings, decision tables, state machines, parameter conventions, expr. evaluation, dope vectors &
exposure		encodings:encryption/CRC/ERC, hashing, tree representation & traversals

Laboratory: Closed lab. Initial lab sessions will focus on machine/assembly language programming. Later sessions will use the C++ compiler to experiment with the actual representation and performance of various representational techniques. Case studies include the impact on space and speed of representational changes in real systems.

This approach to specification of the curriculum has proven to be very useful for explicating the *real* prerequisite structure, for ensuring reinforcement of critical knowledge and skills, and for developing continuity between courses.

As stated at the beginning of this section, we are performing this sort of definition for each course. The curriculum committee is overseeing the process, but the entire faculty is engaged in it and most faculty are serving on at least two such committees to ensure smooth transitions. It is perhaps worth noting that Virginia is unusual among top research universities in its historical (i.e., Jeffersonian) attitude toward teaching and education. The strong faculty commitment exhibited in this case is close to the "norm" for the school.

The Plan

The plan for developing the entire curriculum, and the laboratories in particular has several aspects:

- organization and staffing,
- physical and software infrastructure,
- timetable for the development of individual laboratories,
- evaluating and exporting the results, and
- finances.

Organization and Staffing

The Principal Investigator will have overall responsibility for management of the proposed development. The Departmental Curriculum Committee (Wulf (Chair), Knight, Pausch, and two undergraduate students) have overall responsibility for the intellectual and pedagogical content of the curriculum. Pairs of tenured faculty have joint responsibility for the development of particular laboratories as follows:

1CS	Cohoon and Knight
2SW	Wulf and Knight
2PDR	Wulf and Weaver
3ALG	Cohoon and Reynolds
3SW	Jones and Knight

Each of these individuals is listed as an investigator on the grant, although the support requested for them is small. The Department will use its discretionary resources and other strategies to provide "release time" to these individuals of at least the equivalent of a full faculty member per year.

Central to the oversight of the laboratory development will be an Industrial Steering Committee composed of experienced technical individuals that understand the knowledge and skills needed in industry. We expect them to help us make pragmatic decisions about the curriculum content, and also to identify hardware/software systems that can be extracted from their organizations for use in the laboratories. We are gratified that, on very short notice, Sperry Marine, Science Applications International, Digital Equipment, IBM, and GE have committed to provide advisors.

One or more engineers from industry will join us full time during the development of the laboratories; unfortunately, although there is good reason to believe that will happen, arranging for it was not possible in time for the submission of the proposal.

We are requesting support under the grant for a full-time laboratory technician. The Dean of the School of Engineering has committed to provide a permanent position for this person after the termination of the grant. In the early years, this individual will help to develop the laboratory experiments and materials; as the laboratories mature, this position will evolve into one more of setting up experiments and installing new ones as they are developed by regular faculty.

We are in addition requesting support for an individual that, after some thought, we have decided to refer to as a 'software toolsmith'. This person will work closely with the lab manager who will need support for the integration of tools and software artifacts into the lab for use in experiments. For example, if one or more lab sessions requires the use of

CASE tools, this person would be the in-house expert who will help the lab manager with training, setting up, and various other support functions.

We have requested support for Teaching Assistants under this grant. The Department routinely has eleven Teaching Assistants funded by the School of Engineering to assist with undergraduate courses. The School will add one TA in support of this proposal specifically to assist with development of 1CS, the first laboratory course. You will note, in addition to the more traditional use of graduate student teaching assistants, our proposal calls for the use of undergraduate lab assistants. We have used computer science students as teaching assistants in the past several years, and the results have been excellent.

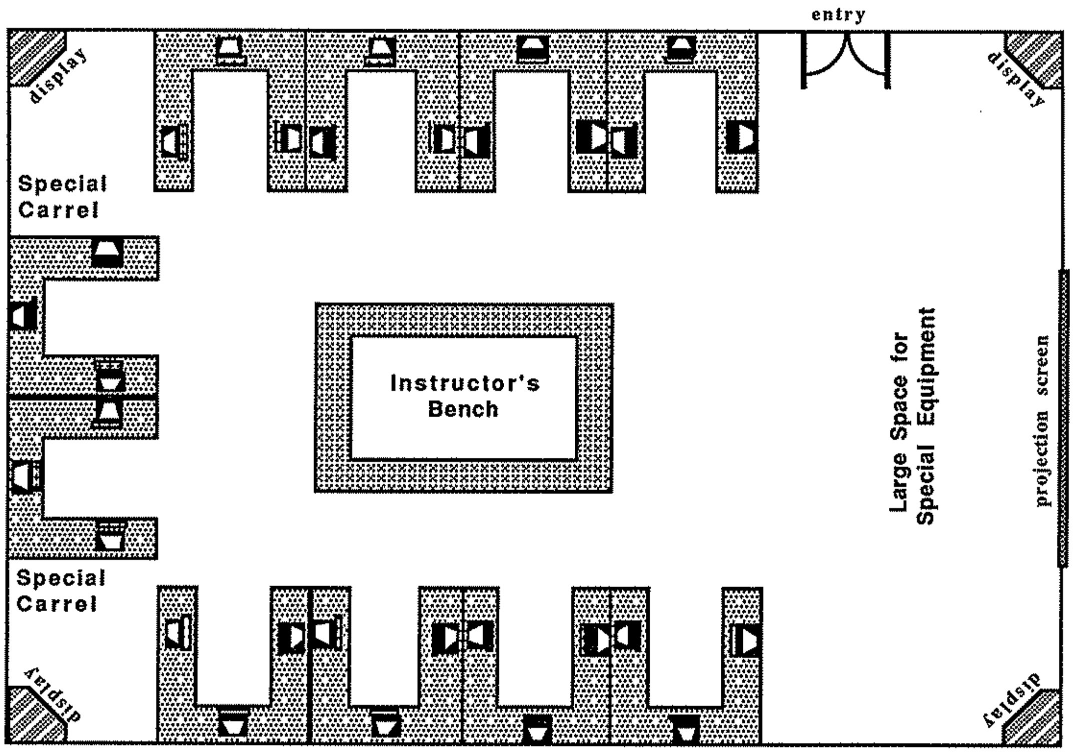
Physical and Software Infrastructure

The Dean of the School of Engineering and Applied Science has committed 2,500 square feet of space for the laboratory; he has also committed the school's technician to renovate the space, including the necessary "lab benches". It is anticipated that this space will be in scheduled for use more than 60 hours per week.

The Department will use its discretionary funds to provide additional requisite furniture, and we will approach our industry partners for additional resources. We are requesting grant support for hardware, software, and special purpose equipment. The department has already begun to use its discretionary resources to provide equipment on which faculty can develop laboratory experiments. For example, Department discretionary funds were used to buy PC's equipped just as the open laboratory machines are for faculty to use in course preparation, and an individual with extensive teaching experience has been hired solely to coordinate the new course development.

The software infrastructure, or "environment", has not been selected yet; that is a project for this coming summer. However, we want the maximum opportunity to export our results consistent with not compromising their quality. We also want to have multiple, industrial- strength systems with which the students can experiment -- which may imply particular, idiosyncratic hardware. Our current thinking involves a PC-based primary infrastructure with X-window ports into a small number of a variety of hardware platforms on which selected experiments may be performed.

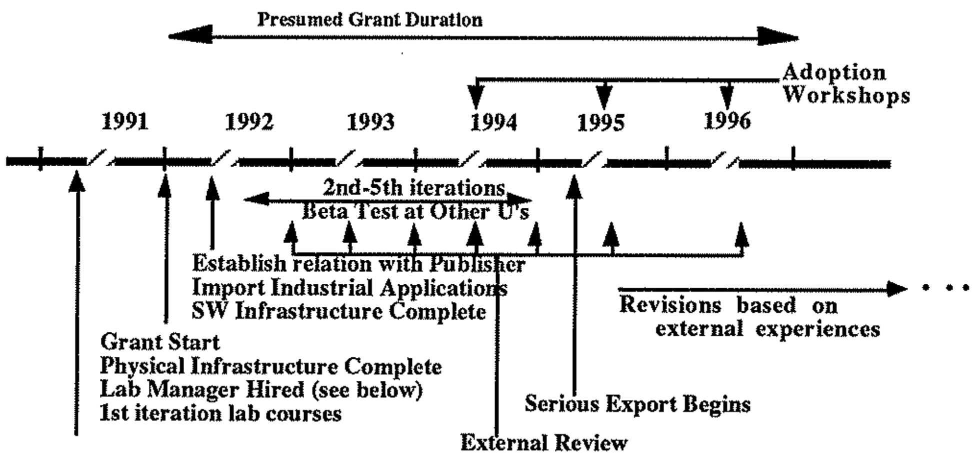
The preliminary physical layout of the laboratory is shown below. It provides 10 stand-alone carrels; these are the workcenters for teams of students who will cooperate on an experiment. The carrels open to a central area in which the instructor may set up demonstrations, and from which interesting student questions can be directed to the whole class. It also includes two "special" carrels with equipment for interfacing among various input/output devices, and a large area for robots and other special purpose equipment.



Development of Individual Laboratories

The plan consists of 4 pieces: (1) an initial refined specification of the course and laboratory content, (2) development of specific experiments and associated materials, (3) iterative testing and refinement of the laboratory experiments, and (4) beta testing the export of the laboratories to other universities (and possibly industrial sites).

In order to not disrupt existing student's programs, it will be necessary to phase in the curriculum over a 2 year period. Thus, the four steps will be executed more-or-less sequentially for a single course, but different courses will be at different stages at any particular moment. Overall, however, we expect to adhere to the schedule shown below; note that by the end of the third year we will have iterated through every laboratory course design at least twice; for some of the lower division laboratories, in fact, we expect five iterations and some "beta testing" at other universities. At that point we will begin to export the courses, educational materials, and supporting systems.



Presumed Grant Duration
Adoption Workshops
1991
1992
1993
1994
1995
1996
2nd-5th iterations
Beta Test at Other U's
Establish relation with Publisher
Import Industrial Applications
SW Infrastructure Complete
Grant Start
Physical Infrastructure Complete
Lab Manager Hired (see below)
1st iteration lab courses
Revisions based on external experiences
Serious Export Begins
External Review

In the following subsections we explore each of the pieces of the plan in more detail.

Initial Refinement of the Specifications

Since most students take their first laboratory course in the 2nd (Spring) semester, we will begin to phase in the new curriculum in January of 1992. To that end, subcommittees involving the entire faculty are refining the "core" courses (those in the prerequisite structure) now, with the intent of developing a version suitable for Departmental approval. Revisions as needed are being made, and approval of the full school faculty will be sought this Fall (the appropriate Deans and heads of affected departments have been kept fully appraised of the changes, and are supportive).

Development of the Experiments and Related Materials: Industry Involvement

The process of refining the course content includes creating sample laboratory experiments; that process is underway. In addition, however, we want a significant number of the experiments to involve "industrial strength" software. To that end, we are forming a number of partnerships with industry -- we expect the industrial partners will provide advice (via our Industrial Steering Committee), actual sample systems on which we will experiment, and in some cases personnel to work part- or full-time on incorporating those systems into our laboratory environment.

We have been gratified that every company we approached responded enthusiastically to the idea, and in principle agreed to participate; letters from them are included in an Appendix. Only the precise form of their contribution remains to be negotiated. We expect, however, to have substantial help from this source!

We have requested a full time technician, a software 'toolsmith', and student support under the grant; all of which we consider to be crucial. In the present context they are crucial to the import and assembly of the software environment supporting the experiments, and to making the "industrial strength" sample systems accessible. In the longer run, however, we do not want to build merely another computing environment; quite the contrary! Instead, we see one of the main strengths of the closed laboratory as the ability to be "people intensive" at just those points when the students need it most. Thus, both the technician and the students will relatively quickly transmute into true teaching assistants.

Iterative Testing and Refinement

The courses we denoted 1CS, 2SW, and 2PDR are replacements for courses that are currently taught every semester. Thus, allowing for the need to be phased in, they can be taught 6, 5, and 4 times respectively during the three year period of the grant (before we begin serious export). The courses denoted 3ALG and 3SW are currently taught yearly, so although we would like to increase their frequency we are not counting on being able to teach them more than twice during the grant period.

Evaluation, Beta Testing and Export

The "product" of this effort will consist of several related but distinct kinds of things:

- experience that we will document with reports and papers,
- a collection of exportable, "shrink wrapped" materials, that can be used within our course structure or adapted to another, including
  - a hardware/software environment supporting the laboratories,
  - a collection of documented experiments, and
  - a set of materials (manuals, etc.).

Evaluation involves several difficult aspects, but all share the notion of peer review -- by our students, by an industrial advisory committee, and by our colleagues at other universities.

Each year we will host a gathering of computer science educators from other universities that are also experimenting with incorporating labs more effectively into their undergraduate computer science programs. The purpose will be two-fold; to allow us to learn from them what they are trying at their institutions, and to have them critique our program.

Because each course is being specified by a list of "postconditions" that define the skills and knowledge that a subsequent course could presume, a large measure of the evaluation process will be done by the students and their professors in the subsequent courses. It is our belief that this methodology will be far more efficacious than the traditional one, and will be shown to be so by test results, student interviews, and student self-assessment. Several undergraduates have already expressed excitement over the proposal and are only sorry they will not be able to take advantage of it. A plan is being developed whereby third year students can TA for 1st year classes, and 4th year students for 2nd year. The TA's will have just been through the same learning process and so know how best to help the younger students. They are much less likely to intimidate their peers, and the learning process is thus substantially enhanced.

We expect to distribute 6- and 12-month follow-up questionnaires to graduates of the new program and their employers to assay its effectiveness and uncover areas for improvement.

As noted earlier, we have already established an industrial advisory committee. We expect them to provide continuing guidance on the relevance of the entire curriculum, but especially the laboratories.

Ultimately, the "proof of the puddin" is whether our colleagues at their universities adopt our approach and materials. To facilitate their evaluation of the approach we will conduct "adoption workshops" in the 3rd, 4th and 5th year of the grant.

Beginning in year two, we will beta test selected portions of the laboratories. The overhead of large beta tests is well known to us, so we plan to start small; a full plan for this will be in place by the end of the first year of the grant.

We have discussed possible distribution channels and media with several publishers, but it is premature to make this decision. But we plan to widely disseminate the results, and advertise the availability not only of our lab experiments and documentation, but with a series of reports documenting our findings, pointing out pitfalls, and in indicating areas for improvement and additional innovation.

All of this will be available via FTP to anyone with a PostScript printing capability. Seldom can one adopt as radical a change as this curriculum represents, however, without one-on-one interaction with its originators. We therefore plan to use the adoption workshops to "teach the teachers" and to help them adapt the curriculum to the particular needs of each institution.

Money has been set aside to provide for travel for some faculty to present the results of our new curriculum and closed labs, and of course extensive use will be made of our networking facilities to keep others aware of our results.

Summary

Focusing on the practice of computing is the highest leverage point for improving Computer Science education today. Closed laboratories are the best available technique for exploiting that focus. Laboratories with a realistic flavor can ameliorate many of the worst aspects of the current Computer Science curricula. A *sequence* of such laboratories, coupled both to each other and to the basic core prerequisites of the curriculum compound the leverage. This is particularly important during the first two years of a student's career and that is where our focus falls.

The Computer Science faculty at Virginia is committed to developing a superior undergraduate program, and closed laboratories are the centerpiece of that program. The Department has committed its discretionary funds and some time of almost every faculty member to this effort. The School has committed space, resources for space renovation, faculty salary funds, an additional Teaching Assistant, and a permanent State-support laboratory manager position (after grant termination). In addition, *the University has reduced the overhead on this grant to 8%, rather than the current 54%.* Industry is committing their most valuable resource -- the time of their best technical talent.

We are still in the process of refining the details, but a great deal has been done with respect to the curriculum in general, specific courses, and establishment of the process of developing the laboratories. We have made initial contact with potential industrial partners. Their response has been enthusiastic.

Only a few of us have been involved in a minor way in the Computer Science curriculum development community in the past. We are pursuing the considerable effort described here because we need it for our own program. The description of Current and Pending Support and the Curricula Vitae will indicate that the principals involved in this proposal have a history of being research faculty as well as educators. We aim to balance research and education at the undergraduate level. They are complementary. And we believe that the "time is right" to make a major change in undergraduate Computer Science education across the nation. We will export the results of this effort -- experience and laboratory materials -- and will pay close attention to the issues of exportation from the outset.

Educational Facilities Support

Current Undergraduate Education Facilities

At the University of Virginia computation support for undergraduate courses is provided by the Academic Computer Center (ACC) whose Director is Professor Alan Batson, a senior professor in the Computer Science Department. Relations between the Computer Science Department and ACC are cordial and the Department strives to maintain computing systems that are compatible with and leverage the ACC computing capabilities. ACC provides open laboratories containing over 150 PCs with supporting software such as MS-DOS and Turbo compilers, for education in the School of Engineering. They also provide 41 workstations running Unix with its standard utilities. All machines are networked and connected with Apple LaserPrinters which are scattered through the School's laboratories. In addition, ACC provides two IBM RS/6000 machines to the School of Engineering. These act as both file servers and compute engines for educational purposes.

ACC has a comprehensive maintenance and user consulting service. Although our proposed physical laboratory is closed by its nature and ACC cannot take full responsibility for a closed lab, we expect to be able to run an environment which is similar, or complementary to, the ACC environment. ACC then could provide maintenance, including backups. In addition, ACC user consultants would be knowledgeable about the basic closed laboratory software.

Requested Laboratory Facilities

Our laboratory requires the following hardware:

- 30 high-end PC class machines (such as an Intel 486 based machine) each with 16 Mbytes of primary memory, 40 Mbytes of secondary memory, color bitmap screen, mice, an RS 232 port, and a floppy disk drive;
- 10 megabit per second Ethernet ;
- sufficient, persistent student disk storage space;
- special purpose hardware devices such as miniature elevators and miniature robot arms with controller processors where necessary;
- laser printers;
- transparency projector and screen; and
- computer screen and VCR displays (large screen monitors) with RGB to NTSC encoders.

We require a software environment that will support all five course labs. That environment has not been specified. There are conflicting issues that are discussed in the rationale below. We strongly desire a single environment for all laboratories with more advanced software tools augmenting the environment for the upper level labs. Additional software tools will include at least

- compilers for at least ANSI-C, C++, and the 2SE assembly language;
- editor (EMACS/jove);
- compilers for any software artifacts to be studied, Ada, and Pascal;
- CASE tools;
- device drivers for the special experiment hardware devices;
- SUIT, the locally built user interface construction toolkit;
- graphics libraries;

animation software;
word processing software; and
presentation creation software.

File system services and access to a few selected software tools and software artifacts for study that are only available on a specific kind of computer not available in the laboratory will be provided by ACC machines on the network.

Rationale

First, it might strike the reader that the laboratory is quite small measured in the number of students that it can accommodate. This is by design to ensure that the lab manager and lab assistants are available for consultation.

We believe that PCs will be a better hardware platform for the laboratory than workstations for several reasons. First, the PC interface is intrinsically simpler due to the historic development of software on the two classes of computer. This simplicity is important for first and second year laboratories. Students should not have to wade through computer arcanea but should be able to perform simple assignments simply -- both in and out of the laboratory. Second, PCs are and will remain cheaper than workstations. Hence, university adoption costs will be less. This will be important as we come to export our laboratories.

Third, both the computing capacity of the PCs (c.f. Intel 486) and the software tools available on PCs are increasing to make them amenable to support the upper level courses. Some of us believe that we have no alternative in the near future to teaching selected upper level laboratories using Unix. We believe that we planned for a migration path -- both hardware and software -- that permits use of PCs with adequate local disk space, that will let us run Unix and other paged software if necessary. Fourth, MS-DOS and X-windows -- now available on PCs -- and other such software are ubiquitous. That means that the student will encounter them elsewhere. In addition, other universities are likely to run such software to support education. That too vastly eases their adoption of our laboratories and the necessary, supporting software.

Specific hardware and some software choices may depend upon donations or matching equipment grants that one or more of our industrial partners may be willing to make. However, we are sensitive to the issues of exporting the software which is required for exporting our laboratories to other universities. We want to minimize the cost to other universities without sacrifice of quality. So we will pursue special arrangements with software vendors as the specification of our software environment begins to take shape. Only after we have pursued these potential relationships further will we know how to best leverage NSF funds for equipment -- hardware and software.

Bibliography

[Den89] Denning, P. et. al., “Computing as a Discipline", CACM, 32, 1 (Jan 1989).

[Epp91] Epp, E., "An Experimental Computer Science Laboratory", SIGCSE Bulletin, v23, n1 (Mar 1991).

[Fol88] Foley, J. et. al., Report of the NSF Computer Science Education Workshop, SIGCSE Bulletin, v20, n3 (1988).

[Nap90] Naps, T.L., “Algorithm Visualization in Computer Science Laboratories", SIGCSE Bulletin, v22, n1 (Feb 1990).

[Ols83] Olson, L.J., “A Lab Approach for Introductory Programming", Proceedings of the 14th SIGCSE Symposium on Computer Science Education, Orlando Fl., 1983.

[Pen90] Penny, J.P. and Ashton, P.J., “Laboratory-Style Teaching of Computer Science", SIGCSE Bulletin, v22, n1 (Feb 1990).

[Pra78] Prathner, R.E. et. al., “A Lecture/Laboratory Approach to the First Course in Computing", SIGCSE Bulletin, v10, n1, (Feb 1978).

[Ree90] Reek, M., “An Undergraduate Operating Systems Laboratory Course", SIGCSE Bulletin, v22, n1 (Feb 1990).

[Rob91] Roberge, J and Suriano, C., "Embedding Laboratories in the Computer Science Curriculum", SIGCSE Bulletin, v23, n1 (Mar 1991).

[Tuk91] Tucker, Allen (editor), et al, "Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force", ACM Press Order Number 201910, IEEE CS Press Order Number 2220

[Tur91] Turner, J. (Clemson Univ.) Private Communication

[Wei89] Weiner, D. J., “Teaching of Assembly Language as a Laboratory Science", SIGCSE Bulletin, v21, n4 (Dec 1989).

[Wen90] Wenner, P. “The Laboratory Component of a Computer Organization Course", SIGCSE Bulletin, v21, n4 (Feb 1990).