Computer Productivity Initiative

Kurt J. Maly[1], Dennis E. Ray[1], J. Christian Wild[1], Irwin B. Levinstein[1], Stephan Olariu[1], C. Michael Overstreet[1], Nageswara S. V. Rao[2], Tijen Ireland[1] and George Kantsios[1]

[1] Department of Computer Science, Old Dominion University
Norfolk, VA 23529-0162, email: cpi@cs.odu.edu

[2] Intelligent Systems Section, Center for Engineering Systems Advanced Research,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6364

Abstract. Over the last three decades computer science has evolved into a mature and experimentally oriented discipline with a well defined curriculum. Only recently have we come to realize that as a discipline computer science must reach beyond its own subject area to applications in other disciplines in order to stay relevant. Most computer science curricula teach principles and programming skills in isolation from an application perspective, provide limited laboratory experience, and introduce inadequate integration with non-CS components. The Computer Productivity Initiative, described in this paper, proposes to alleviate these problems by integrating a multi-year project into the curriculum. The project involves courses taken in three different years of the curriculum. It includes hardware and software issues and also addresses engineering, business, and other non-CS issues. The initiative uses prototyping and simulations in the development of specifications for an integrated television communication and display computer system. The students apply principles of productivity and make extensive use of leading-edge technologies both in the process of the project development and the product being developed. They hone essential career-oriented skills in the areas of management, formal presentations, and group problem solving. This paper is a report of work in progress. It emphasizes the implementation issues we are facing and the integration of evaluation into our curriculum development. It describes our preparation for the dissemination of a model curriculum when we are able to demonstrate that the approach is adaptable to CS departments across the country.

1 Introduction

The Computer Productivity Initiative (CPI) is a concerted effort designed to modify the traditional computer science curriculum to better prepare new computer professionals for the careers awaiting them. Its formal birth came in response to a singular complaint echoed by many fronts concerning the quality and preparation of the newly graduated students. Several deficiencies have been identified, in part, by many studies and papers. In particular, reports by the ACM/IEEE Joint Curriculum Task Force [2] and NSF Computer Science Education Workshop Report (Foley *et al.* [9]), British Computer Society and Institution of Electrical Engineers [1], and articles by Denning [10], Parnas [18], Shaw

[22] and Wulf [25] discuss several aspects of computer science education. The CPI innovation evolved during various departmental faculty meetings and was subsequently supported by the Old Dominion University. This support led to the commitment of combined financial resources by the National Science Foundation and Old Dominion University to develop this complex project over several years.

The continuing discussion of what higher education institutions should be teaching the new computer scientist is documented in several studies, and is a focal point of educational professionals (Parnas [18], Shaw [22]). Their evaluations call for changes in the current educational curriculum to meet the needs of year 2000 and beyond (see the report of the Commonwealth of Virginia's Commission on The University of the 21st Century [3]). Locally, the notion that shifting needs require curricular changes has been diagnosed from several symptoms uncovered in annual departmental self-assessment activities. During interviews with graduating computer science majors over the past few years, students have reported difficulties in describing, during job and graduate school entrance interviews, how their background meets the emerging requirements of computer scientists. Graduates who have been practicing professionals for one to five years report that their initial work experience was devoted to several months of company training programs before being allowed to take on active company responsibilities. Discussions with business and industry leaders as well as government and military representatives, including those on our advisory board, have revealed that employers have the same impression of recent graduates. This picture, when brought into focus, displays a curriculum which develops graduates who are less than prepared for immediate productive efforts in entry level positions. They are often found to be functionally incapable of productive work without significant training beyond their university education.

Interestingly enough, the same newly graduated professionals, although considered inept and incapable of productive efforts, are perceived by the same employers as having a sound education and a great promise of potentially significant productive efforts in the future. They are perceived as certified computer scientists who, generally, do not possess the skill to apply their scientific education in the fields where they are needed. Their capabilities are strong but fall just short of societal, business, industry, and graduate school requirements and expectations.

In this paper we describe the CPI effort that is currently in progress. As we shall explain, the current project Logically Integrated Family Entertainment System (LIFE), involves designing a modular interface unit to a picture on picture television system that extends the functional use of the television to incorporate typical household functions. Three basic modules were described: home safety and security, message data base and television directory, and remote telephone access and operation. All modules interact with the family television either as the primary or secondary window of the family's picture in picture television.

The remainder of this paper is organized as follows. In Section 2 we identify some of the shortcomings we have found in our computer science undergraduate program. We have incorporated a multi-year project into the computer science curriculum; a brief outline of our approach is described in Section 3. Our plan and its execution is based on a design described in Section 4. Section 5 describes our evaluation approach to ensure the soundness of the model curriculum we are planning to disseminate. We report the current status of the implementation cycle and future work in Sections 6 and 7 respectively.

2 Limitations of Current Curricula

In recent years, the discipline of Computer Science is increasingly being perceived as a laboratory science. Some of its concepts are best illustrated by a repetitive application of a sequence of investigative actions: analysis, proposed solution, experimentation, observation, and new analysis. As history has taught us, the availability of technology is not enough, we must also learn to make productive use of that technology.

The traditional computer science curriculum is successful in addressing the fundamental concepts of the field, as described in several ACM/IEEE reports on the contents of the CS curriculum. In particular, the ACM/IEEE curriculum achieves the objectives for which it is designed [2]. Students are carefully guided through the development of problem solving methodologies and programming skills. Typically the students are exposed to theoretical concepts which underlie computer science. These theoretical concepts are illustrated using well understood and well structured applications such as compilers, operating systems and databases. These illustrations generally include clear specifications of a problem to be solved (as do most laboratory assignments) and so students are rarely exposed in classes or laboratories to the system development process which precedes these clear specifications. Some enhancements to the conventional curriculum could provide a significant exposure to that process, beginning with a broad, unstructured problem and aiming at a structured solution which improves productivity of the process for which the project is carried out. These enhancements will enable the students to go beyond the following limitations associated with traditional curricula of computer science at the undergraduate level:

1. *Teaching of principles and programming skills in isolation from application perspective:* The material from several courses is rarely integrated in a way to illustrate its applicability to practical projects.
2. *Limited laboratory experience:* Conventional instructional methods based on lecturing in the class and leaving the students to themselves to perform the implementation work do not provide students with adequate laboratory experience.
3. *Restricted comprehensive framework that does not provide for enough integration with non-CS components:* Typical industrial projects are not restricted to contain only modules that are exclusively CS-related; often integration of hardware/software modules that require knowledge of engineering, etc., are required to complete the project. But, in traditional curricula there are not many efforts to integrate solutions from outside of the CS domain.

4. *Inadequate emphasis on life-long skills:* Life-long skills, such as justification presentations of the project or its subsystems to groups, project planning, and cost estimation are virtually unexposed to students. Also, group interaction and cooperation skills are not formally developed in the curriculum, yet the students are expected to possess such skills as soon as they enter graduate school or work environments.

Some of these limitations have been identified in the past by several educators. Various universities have efforts underway to address these problems. For example, the initiative at University of Virginia [5] addresses the issues of limited laboratory experience. The Software Engineering Institute addresses several issues of solving large scale real-life problems in the domain of software engineering [6].

3 The CPI Solution:

The primary objective of this initiative is to increase the effectiveness of the undergraduate learning experience. This objective will be achieved by applying a multi-year, real-life project with a coordinated infrastructure of formal laboratories. The effect of the proposed initiative will be the education of computer scientists who have a thorough understanding of fundatmental CS principles, and who can apply this knowledge to enhance the productivity of an application process.

The principle features of this initiative can be summarized by presenting them as foci of coordinated effort.

Focus 1: *Multi-year project experience:* Students will have the opportunity to solve well-specified problems in the regular course work and also to apply these principles in a less structured environment of a project. The hands-on experience in the laboratories provides both motivation and application of the principles. The following topics will be emphasized:
(a) planning, specification and development; design, implementation, evaluation and improvements;
(b) use of material from different courses; and
(c) management skills, presentation skills, and group problem solving and coordination

Focus 2: *Experience beyond computer science:* By the choice of a real-life project, the students will be exposed to various issues such as engineering, societal and economic impact, and other similar concerns that are not covered in mainstream CS curricula. Of a particular interest is the impact of non-traditional subjects such as security, human to system interfaces, available technology and its acceptance by society on the solution to the project.

Focus 3: *Exposure to leading-edge technologies:* While the main emphasis of this initiative is to develop graduates who are capable of increasing the productivity of the users of computing technology, students will learn to increase their own effectiveness, both as individuals and as teams, through the use of computer technology, e.g. multi-media, hyper-case, and software engineering tools.

Focus 4. *Principles of productivity:* Students will be exposed to concepts to improve productivity using computers and measure the productivity gains in the context of a concrete project.

Focus 5. *Portable Curriculum:* As a tangible output, we expect to produce a portable curriculum package so that CS departments at other universities can implement CPI without incurring all of the initial development costs. The proposed package consists of the following items:

(a) Instructional Material: course folders, instructors manual, curriculum modifications, video tapes, and textbooks.
(b) Organizational Material: manpower and resources analysis, facilities, and equipment.
(c) Software: Our evaluation approach is based on evaluating the decisions in the development process, and adapting our model curriculum to other environments by using only those portions which are appropriate and developing individualized components for the remainder.

The major innovations of this initiative are:

1. The fundamental characteristic of our effort is to provide students with general problem-solving methods which relate to the use of computers to improve productivity.
2. An infrastructure which integrates a coordinated set of formal laboratories into the curriculum.
3. Exposing and teaching students life-long skills to develop coherent and persuasive application development plans.
4. Teaching students effective scientific methods for experimentation.
5. Effective utilization of computer technology emphasizing the problem-solving process.

In developing this solution we considered alternative solutions including the senior projects found in many existing curricula. We decided instead on a broader approach. First, the project pervades the entire curriculum: students are exposed to parts of the project early in their curriculum and will carry it out in its entirety at the senior level. Secondly, this project is done by the class as a group, where different solutions are proposed, presented and discussed to arrive at a final solution. In particular, our approach is different from the method used at the University of Virginia [5] and Clemson University (Turner [23]) in that a single larger scale project carried out by groups is a unifying factor in the laboratories of various courses. This integrating factor introduces software engineering principles that are presently being investigated for incorporation into the undergraduate education. A need for exposure to such principles in undergraduate education has been recognized universally; (see [1, 4]). An alternative we did not consider was a separate undergraduate program in software engineering [6, 12]; instead our approach is based on integrating software engineering methods into the existing curriculum at a larger scale. Thus, our approach is also different from those that attempt to incorporate software engineering principles into individual courses Bruegge [7], Kantioudi *et al.* [13], Lively and Lease [14], and Offutt and Untch [15].

Finally, although our approach is laboratory-based, it goes beyond adding formal laboratories to an existing set of courses, since the laboratories are integrated to bring material from various courses to bear on the solution of a single focussed project. Efforts that introduce closed laboratories into individual courses are described in Epp [8], Olson [16], Penny and Ashton [19], Prather *et al.* [11], Ray [20], Roberge and Suriano [21].

4 Implementation

Overcoming the limitations outlined in Section 2 requires changes to the curriculum. The fundamental CPI approach is to coordinate student activities (e.g., laboratory assignments, class projects, course papers, class presentations) such that each activity always achieves two or more educational requirements simultaneously. One of these achievements is always a CPI issue.

To incorporate the five foci of Section 3 into the curriculum required us to define the details of how to coordinate the various courses in the program. Our implementation adapts the existing curriculum so that a large scale real-life project is developed by students in several courses during several semesters. We recognize that the typical industrial project has many factors such as: the development of functional requirements and system specifications, the selling of the proposal to higher management, the construction of a development plan with schedules, budgets, and personnel requirements, the recognition and resolution of perceived risks, and the development of detailed system specifications which have gone through a cycle of evaluation and refinement.

In choosing a method for implementing the foci, we had to overcome an apparent contradiction. In a typical large-scale development project that has software as a component, the software is developed over several years by various professionals devoting their full time effort to the project. Students, even as a team, are at best available for two semesters (mostly one) and are just part timers (they take other courses). We needed a method which allowed seniors to do the planning but which permitted freshmen, sophomores and juniors to have the experience of participation in a large project at a level that is consistent with their knowledge and skills. The pieces of the large project must as well conform to the individual course concepts and required materials. The large project must be sufficient to allow the lower class explorations and significant enough to require major efforts by the design and specification courses. The logical connectivity for these efforts is the laboratory associated with the courses.

The coordination of the laboratories, the CPI project, and the overall program is administered by the CPI committee. The committee is composed of seven full time faculty and two graduate research assistants. One faculty member serves as the coordinator of project elements. Others function not only in an advisory capacity but individually as course coordinators for each of the courses associated with the project. The committee meets weekly to analyze outstanding problems and develop solutions. The members of the committee are augmented by an advisory board composed of seven members from diverse organizations or activities (education, research, production, system design, or development). The advisory board and committee meet three to four times a year to further define the function and operation of CPI.

With the concurrence of the advisory board, the committee chose to include two lower level courses along with the senior level Computer Based Productivity laboratory courses (CS 400 and 401) for the initial implementation year of CPI. CS 250 (Problem Solving and Programming) was selected as the sophomore level course primarily because of its long standing use of a formal laboratory and its fundamental problem solving course concept. CS 300U (Computers in Society) was selected as the junior level course because of its two laboratory like activities (paper and class presentations) and for its source of critical non-CS resource information for support of CS 400 and CS 401 laboratories. Additional courses, such as Data Structures, Graphics, and Software Engineering, may be incorporated into the CPI concept but as informal, non-participating courses for evaluation purposes.

4.1 Support Courses

The fundamental modifications of CS 250 include the specific use of the CPI project as the theme for programming assignments and course examples all at the sophomore level. Prototypes will be given to the class to use in performing experiments with the experiment results reported to the senior courses. The critical difficulties expected with these modifications is the availability of problem definition from the senior level productivity laboratories early enough for use in the sophomore course. The CPI coordinator will function as the expediter to insure sufficient statement definitions are described in CS 400 and 401 suitable for CS 250 use. While the flow of information from CS 400 is critical for the achievement of CS 250 course requirements, the reverse flow of product definition is expected to provide direct support to the CS 400 laboratory efforts.

CS 300U currently requires two major forms of student class activities: a formal course paper and student presentations of topics in a forum environment. To utilize this course and the student presentations in the large CPI project, the topics of both the paper and the forum issues will be focused on such critical factors as:

Personal security	Energy sources	Expected life span	Occupations
Education	Personal Privacy	Food supply and type	Government
Transportation	Health and medicine	Economics	Labor supply
General tools	Communication	Business finances	

The subject requirements are products of the ongoing CS 400 and CS 401 courses. The CPI coordinator will function as the expediter to insure that the required topics and issues are identified and provided in a timely fashion suitable for use in CS 300U. The results of these taskings are critical components of the senior level courses. The flow of information with the junior level courses is critical in both directions for the success of the CPI project.

The CPI modifications to the lower level courses (i.e. CS 250 and CS 300U) do not alter the course structures nor the theory or skill development of the courses. The modifications affect the subjects and general theme of the student activities that are assigned during the normal concept-learning within each of the courses. This is viewed as a critical factor in that curriculum-evaluating bodies must not see substantive changes in these core courses that may affect certification decisions as result of CPI incorporation. Fundamentally, we believe that the installation of CPI must not adversely impact the scope of theory, concepts, and knowledge of existing courses.

4.2 Senior Courses

The senior level laboratory courses are the vehicles for ensuring that the primary CPI objectives are achieved. The sequence of CPI senior level laboratories provides students with a clear view of applying the many theories and concepts illustrated and learned in previous courses. It is at this level that all of these knowledge-based skills are applied. Students are faced with new problems that did not appear in their previous courses. They are tasked to develop a solution to a typical problem of increasing the productivity of a system. The problem is the CPI project and is presented as a non-structured requirement. The students must develop the problem solution taking all product-related concerns into account in their decision-making process. In this context, the students participate as leaders of small groups, managers of a larger group, and support technicians and professionals. They function essentially as project managers, defining the problem, developing solutions, and evaluating alternatives for feasible solutions. They must present formal documents for project continuation and produce specifications suitable to be used for production development.

The senior level laboratory students will task lower level courses with software programming requirements, prototype experimentation, impact studies, problem solution plans and implementation algorithms. They will evaluate the products from the lower level courses and use the information to support their ongoing efforts in developing problem solution specifications. The laboratory course material provides students with a guided set of methods to meet the course requirements. It is in the application of these methods that students will likely develop the major productivity improvements. Their laboratory requirements will largely be similar to the requirements of productive computer scientists in their first few years after graduation. Their laboratory experiences will virtually qualify them as assistant project managers who are knowledgeable in most of the facets of applying the technical computer science knowledge to actual problems either in industry or in advanced graduate studies. They will have explored the fields of planning and scheduling, resource allocation and budgeting, to the incorporation of computer hardware and software systems to produce the specifications for a tangible product that society needs and will buy.

4.3 Project Execution

The selected CPI project is carried out over the period of one year (or more years if the problem is sufficiently large) in various courses. Two main courses at senior level deal with the generation of a solution to the project; this step involves many issues not traditionally considered in software development for university course projects. The goal of this initiative is not to develop commercially acceptable systems but rather to develop system specifications that solve a problem. The specifications include a detailed design description, the expected functional performance, cost budget analysis, and a production plan based on analyses, simulations, and prototyping. The project selected as the CPI project will require a solution that consists of not just computer software or hardware, but requires the design of a system where a computer is only one solution factor albeit an important one. The initial selection of the expansion of using the family television as an interface for other family activities as the CPI project meets these requirements. The LIFE system as described by the initial 10 students of CS 400 is a challenging project that defines the hardware and software modules that support many aspects of family life using the family television as a graphic display and communication device.

Since the CPI project theme will have been used in several supporting courses, computer science students will have the sense of participating in a computer productivity project for several years. By the time the student is presented with the entire task of the project, the majority of the basic computer science material has been seen in various courses. The senior students will then be able to readily integrate material of previous core courses into a single project solution at the senior level. It is the early piecemeal introduction of the CPI project that provides an increased productivity for the senior students, an efficient design and description of the problem solution that is the gathering of many of the pieces the students have seen in previous courses.

For most students, their project experiences will involve the same basic project. Developing an entirely new project each year imposes an excessive burden on the faculty. Therefore it is essential that projects be reusable. Reuse of projects can take two forms: the same project is repeated from year to year, as is often the case in engineering senior projects; or the original project is refined, revised, and improved from one year to the next. Our method combines the two approaches because we wanted every student to have the initial "blue sky" development experience when presented with an open ended problem area, but we wished to incorporate the best of the previous classes' work. The seniors will initially approach the problem area in a blue-sky fashion, but the instructor will introduce worthy materials to supplement their work after they have achieved a definition of their project goals.

We are convinced that the single theme project introduced and incorporated in courses throughout the curriculum will enhance the student's ability to rapidly grasp the concepts of the new course. At the senior level laboratories the students will see the total picture of the CPI project. Because of their previous knowledge of the project theme, they will not require the "learning curve" time to begin active project prosecution and specification development. They will also likely reflect on tasks they received at the lower level courses and use that knowledge in generating similar requirements to support their laboratory requirements. This integrated and coordinated project theme across the course boundaries is the CPI innovative concept that we believe will solve many of the problems with our curriculum of today.

5 Documentation and Evaluation

We are using a locally developed software-engineering tool, D-Hyper-Case, to document a decision-based history of the project and to help us determine ways to evaluate the project's effectiveness. Details of D-Hyper-Case can be found in Wild, Maly and Liu [24]. D-Hyper-Case takes its inspiration from Parnas' suggestion [17] that each program module encapsulate one design decision. The tool permits each decision to be described in terms of the problem to be addressed, the alternatives considered, the decisions made, and the relations (links) of the decision to other problems and decisions. The result is a directed acyclic graph with a decision at each node. The graph nodes depict solved problems which likely generate new problems. Solutions links are then established in both directions. Each decision node may be linked to the code and documentation that implements the decision.

We have used this tool to record the history of the project in a problem-oriented fashion. A sketch of part of the decision tree may be seen in Fig. 1. A sample node is shown in Fig. 2. The initial decision to revise our curriculum is progressively refined as we move toward the bottom of the decision tree. Each node lists the alternative which we considered in making the decision and each decision is given one or more justifications which explain why a particular alternative was chosen or rejected.

These justifications, which we call "prejustifications," represent our best judgment at the time of making the decision: they are a pre-evaluation of a number of what-if states, but they may not provide evidence that the decision made was actually the best one. To evaluate the actual course taken, i.e., to "post-justify", requires that we collect data on the project with an eye toward determining whether our decisions were good ones and to provide useful information to other computer science departments which might wish to use our project as a model for curricular revision of their own.

If all of the alternatives could have been simultaneously chosen, then we could use the alternatives as control groups and collect data with an eye toward establishing whether the favored alternative was indeed the correct one. In general, such a course is far from feasible. Nevertheless, we decided to consider at each node what sort of data could be used to validate our implementation relative to the other alternatives. Such data, if collected, would serve several purposes.

(1) In some cases we can find somewhat comparative data. If, for example, we considered doing nothing or requiring our students to take courses in other

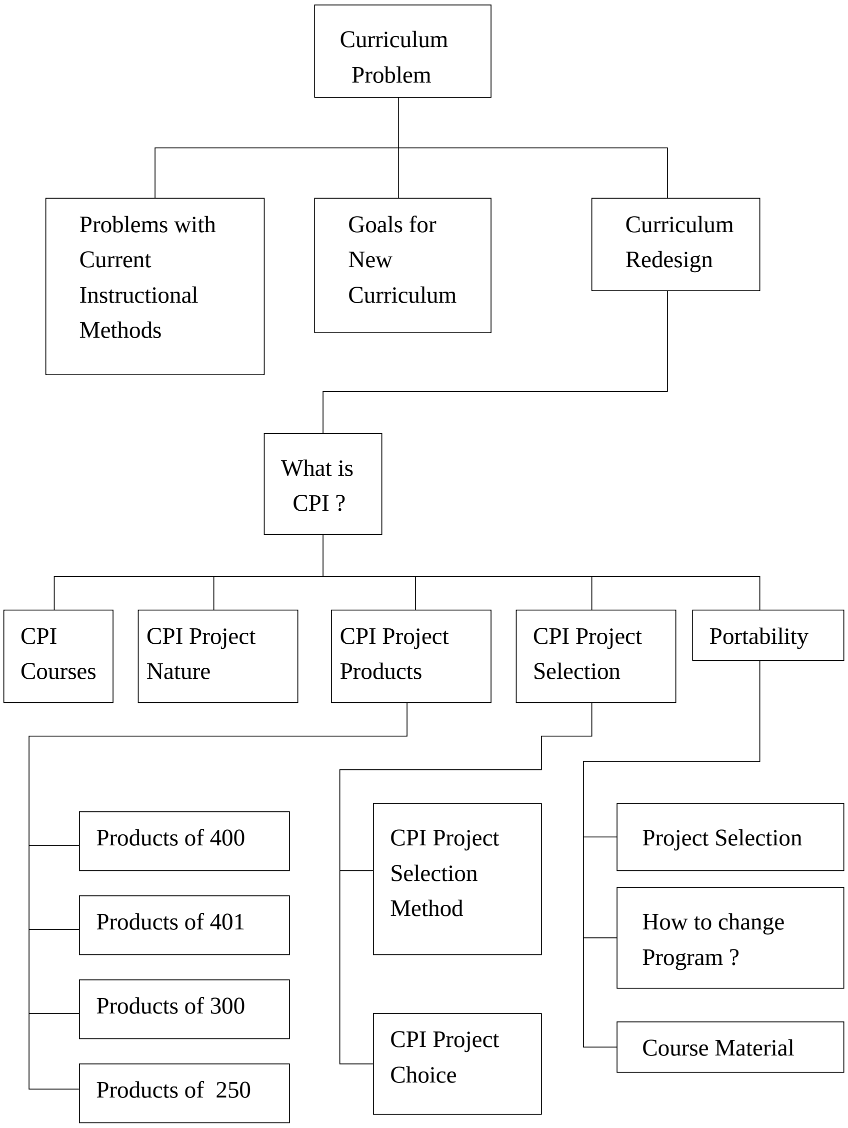

Fig. 1. Decision Tree

departments, we could gather similar data on our current students or those of them who choose to take such courses.

(2) The data gathered displays what was done, even if there is no comparison with other possible alternatives. We could demonstrate significant improvement in student oral presentation skills or show that major elements of the curriculum had been used in a course without gathering comparative data.

(3) The data would often be useful to others considering curricular changes of

PROBLEM: CPI-Project-Nature
What sort of project schedule should we use?

ALTERNATIVES:
1. Progressive development by senior classes over several years. From planning to full prototype.
2. New project every year.
3. Recycling of same project over several years.

DECISION: Recycling of project over several years.

JUSTIFICATIONS:
• Rejected progressive development because students in different years would learn quite different things. A new project every year would require too much ramp up effort each year. Decided to recyle using a "blue sky" approach each time, allowing instructor to introduce material generated by earlier runs of the course as appropriate.

EVALUATIONS:
• To some extent the alternatives can be compared from our own experience. The first time course is offered is like alternative 2 except that we must factor out ramp up costs for the whole program.
• Measures: hours spent by directors and assistants on project specific matters (leaving out e.g., getting facility in shape, overall course design.) Second run of course will show effort saved by not using alternative 2. Requires time logging.
• What are differences in what students learn in first run vs. second? Measures: Comparative pre- and post-ratings of desired skills; can use student portfolios (including taped presentations), director's observations, advisory board's observations of student presentations. Also ask students what use they made of predecessors' work.
• Are there differences in student efforts, either in total time or distribution thereof between the first two runs. Requires time logging.
• Cannot directly document alternative 1 from own experience. If another CS department, or even a department from another discipline, pursues this approach we can get information from them.

UPLINK: What-is-CPI? (5)
DOWNLINK: (none)

Fig. 2. CPI: Node 7: CPI-Project-Nature

their own. Our course of action may be one of the alternatives they reject. Then our data would serve, to some extent, as a control on their own.

The question of what data to gather begins with relatively high level descriptions near the root of the tree. Those descriptions are refined as we move toward the leaves until we have concrete descriptions of what we want to measure.

6 Current Status

After much discussion about feasible projects, appropriate course contents, delivery approaches, and evaluation mechanisms, we realized that we needed a test offering of courses that included as many CPI features as possible. Tables 1. and 2. are simple summations of a large number of hours of debate over the project to be used. We would use a test class experience as a mechanism to better identify successful techniques and administrative procedures, validate implementation concepts, and provide tangible data to be used to further define the initiative. While we expected to gain valuable knowledge as to what would be successful and what would not and the degrees of each, we also anticipated the identification of new issues. Although the implementation of CS 400 required a large expenditure of resources, the results have exceeded our expectations. The data gained has served to crystallize the CPI approach and served to guide creation of clearer solutions to the myriad of problems to be solved before the first offering of the two senior courses beginning in the Fall of 1993. Of particular importance has been the experiences and data the courses have yielded to support further decisions on several complex CPI issues. The more important of these are discussed here.

Problem-Solving in the curriculum, a critical challenge: This issue of expanding fundamental problem solving is one of the central concerns not only to CPI but as well to all CS curricula. CS 400 students were initially tasked to describe potential enhancements to the family television. They had to take this completely unstructured open-ended problem scenario and identify potential solutions to the problem. The descriptions of their selected enhancements, along with their reasons for the selections formed the basis for the eventual CPI project description. Essentially, the students defined a set of application solutions to a real-world problem. Some of their findings were used in CS 300 as topics in both oral and written presentations.

Multi-year project: While the full complexity of introducing a multi-year project was only partially evaluated in this test, several of the complex timing issues previously anticipated were demonstrated. The decision of selecting the specific content of each course that mutually supports the multi-year approach has been narrowed somewhat. The original expectations of significant support for the senior courses from the lower level courses may be too great without larger modifications to the lower level courses. These lower level courses will require additional course material, with associated course changes, to provide the required level of support.

Abbreviation	**Project Description**
Naval Charts	Automatically update and disseminate that change frequently
Library	Build a computerized library card catalog
HITTCom	Integrate TV, telephone and computer to allow people to control lighting, security, temperature, appliances
Toll Booth	Automate toll booth collection on highways
VIT	Loading and unloading of ship containers using cranes
Radar	Analysis, design, specification and emulation of a radar interface
Filled Landmass	High performance filled landmass capability using the world vector shoreline
Teaching Project	Programs to teach math and science to teach junior high school students
Student Database	Expansion of university databases to include photographs
Re-Engineering	Writing programs to convert different source codes to standard source code, e.g., from COBOL to Ada

Table 1. Project descriptions

Advisory Board use as student evaluators: One of the roles initially defined for the Advisory Board was to become a board of directors for the students. The students were tasked to present to this board the description of their product along with their plans to develop and produce the product. They were challenged to view this board as the decision panel for the continuance of their project. Thus, they were required to justify and support each of their decisions. These students had never been faced with a critical review board presentation. They met the challenge and made the presentation. The advisory board provided critical comments on both the technical quality of the proposed project and the quality of the presentation. The results of this first test case were enlightening to the students, generally supportive of CPI, and also highlighted additional considerations of the process. The new considerations include issues of: how do we ensure success by the students in this critical presentation, how do we ensure the students do not embarrass the university in front of the advisory board, and how do we ensure the board provides the necessary feedback to the students and the CPI faculty?

Value of using speakers in class: The CS 400 test course arranged for a guest speaker on a topic that was critical to one of the course's presentations. The use of guest speakers was viewed as a method of expanding the course content with relevant and current topics. The speaker was a recognized industry professional who provided current business data. Scheduling considerations prevented additional speakers from also participating in the course. The students appeared to readily accept the ideas of the speaker easier than the same ideas from the instructor. Again, new issues were identified as a result of the outside speaker's presentation. Coordination of timing is more difficult than expected. For maximum effect, students must actively participate in the presentation. How to insure this participation is of significance.

Incorporation of leading-edge technology: The early offering of the CS 400 course brought with it a sense of incompleteness of the course preparation. Students initially expressed concern with the promises of equipment and facilities. The students were initially excited about using multimedia equipment and facilities to support them in the generation of their course requirements. However, because of the piecemeal delivery of the multimedia hardware and software, students were faced with rapid introduction and use of new technology without a prolonged training period. While this seemed a shortcoming, in retrospect it caused students to "make do" with what they have available in keeping with typical real-world problem solutions. It is clear however, that adequate training on new multimedia systems will be required to fully use the technology to support course requirements. Either a tutorial or synthesized set of documentation or both may be required.

Criteria for project selection	**Explanation**
Lifelong skills	skills such as design processes, writing, communication, presentation, managing groups, developing mission statements, budgets for projects, work for clients
Productivity gain	Productivity gained by the user of the system
Relevance to local industry and government agencies	Partnership in education
Application orientation (non-CS)	Amount of non-CS knowledge
Time investment of the buyer	Amount of time needed for the students to complete the project, by an outside party, members of CPI board, faculty, industrial employees, students and other experts or clients
Time investment of builders	
Time investment of domain experts	
Relevance to society	Benefits of the project to society

Table 2. Criteria for project selection.

Interoperability between senior and support courses: The CPI faculty has described several critical and largely unresolved issues in the orderly installation and administration of the project throughout all courses. The test offering, although restricted in this sense, indicated that the complexities of the coordination problem are indeed as difficult as currently expected. The generation of requirements at the senior courses cannot be delayed; nor can the generation of the supportive results be delayed from the lower level courses. The content of the task must be sufficiently clear to be prosecuted by the support course. The results must be of the required quality to be useful at the senior level courses. The issue that remains is: how do we insure these minimums are met or what is the alternative if one or more timely requirements is not met.

Class trips to project installation sites: Early expectations included the necessity of field trips to view and experience the installation and use of similar components of the chosen CPI project. The trip to the "Smart House" in Baltimore, Maryland, was one of the more enlightening aspects of the CS 400 test course. Students were exposed to the myriad of hurdles that seemingly simple problem solutions can generate. They saw actual installation decisions that had cost, efficiency, and functional factors mitigating the easy technical problem solution. The use of this trip was effective but required significant scheduling efforts. Other projects will likely not easily find host implementation sites that are available for student viewing.

While the use of the test courses provided significant assistance in solving many of the outstanding CPI issues, they also added new issues. The prototype CS 400 course has provided the faculty a functional testbed for many aspects of the project. Without this prototype and partially supportive courses, it is expected that valuable time and effort would have been lost in following arbitrary decisions. The data provided has only partially been digested. The end of the Spring semester will begin the full evaluation and design phase of the project.

7 Summary and Future Plans

While the project is on schedule, we are in the middle of the effort and much remains to be done. Much of the remaining work is routine though nontrivial such as the additional course development for CS 401, and handling the "ripple effect" of these courses into existing computer science courses, particularly our two undergraduate software engineering courses. These topics are not discussed further here. Instead we focus on two issues which we perceive as key risks to the success of the project: resource requirements and effectiveness.

From anecdotal evidence (both personal experiences and discussion with colleagues) it appears that many believe that project-focused curricula are beneficial but too time-consuming for faculty and hence perhaps undoable given typical university funding, staffing, and reward structures. This may be, and our experience thus far does not contradict this. Obviously, accurate assessment of the additional efforts required is confounded by the work resulting from any new curriculum development. We are keeping time records to document both faculty and student time. The lack of a model of how to achieve the goals we have in mind increases the development time significantly. The lack of support materials (for example textbooks, workbooks, or even sample multi-class projects) obviously increases instructor preparation time.

We will address the issue of additional effort in two ways: by record keeping (to estimate the additional time required) and by developing curriculum support materials so that others can use this approach more efficiently. After a course is repeated at Old Dominion University, comparison of instructor times for the first and second offerings (and also with existing project-oriented classes) will provide an initial indication of the additional time required. The key to reducing any increased overhead with this approach is in providing adequate support material. This material will include standard curricular material (lecture outlines and notes, syllabi, suggested readings) and sample schedules, sample project descriptions, practitioner guidebooks, guidelines to students for conducting group meetings, presentations, and evaluations, sample documents produced by students, video tapes of some required student activities (interviews with clients, student presentations, board evaluations). Low cost dissemination techniques will be put in place (such as use of FTP to allow retrieval of documents) to share these materials. We will also develop and report effort measures from our experiences; we will make available a variety of materials which others can use for similar courses. If other schools choose to participate, we can provide a repository for related materials developed elsewhere.

We will continue to work on mechanisms for measuring the costs versus the benefits of this curriculum. Many cost metrics are obvious (such as measuring instructors' preparation time) and in general, costs are more easily quantified, but the benefits to computer science students over the course of their careers is more difficult to measure. Most of our students are employed by business and government immediately upon graduation. For these students, some benefits are immediately evident. For example during our initial offering CS 400 we saw dramatic improvements in students' oral presentations (and these students had already taken a required speech class). Many other benefits are less tangible but measurable. For example, we will track the students who have taken these courses so that we can compare them with earlier graduates who did not take these courses. Still the development of reliable and inexpensive metrics requires additional work.

However, many important benefits are difficult to quantify. For one example, we perceive several types of benefits in involving an external board made up of major business and government employers in the area: bringing these people onto campus to discuss how undergraduate computer science education should be conducted benefits them and us; having board members interact directly with our students has benefits (and some risks). As a second example, many of our students go on for M.S. and Ph.D. degrees. This exposure to many of the "real" difficulties in providing automated solutions to real-life problems should give them an improved perspective of computer science, but we see no inexpensive measures to validate this expectation.

We will provide documentation of the costs associated with this approach (equipment, space, supplies, travel). This is relatively easy. We will also continue to struggle with effective evaluation mechanisms; some procedures are in place to collect required data. We have in place (using D-Hyper-Case) a mechanism for recording our decisions along with their justifications and anticipated effects. We will measure some effects for the choices we have made and compare them with the anticipated results. Additional metrics must be defined and evaluation procedures developed to provide them.

For the Old Dominion University faculty participating in this project, many of our discussions focused on four questions: 1) what should we do, 2) how should we do it, 3) what will it take to do it, and 4) how will we know if it works? At this point, we have reached agreement (with some details left to resolve) on the first 2 though as we offer the courses, ideas about 1 and 2 will be revised. We are collecting data on 3, and have some initial mechanisms in place to deal with 4. But evaluation and dissemination techniques will continue to be a topic of our committee meetings. We feel that our initial offering of CS 400 has demonstrated many aspects of the feasibility and some of the benefits of what we propose; other aspects remain to be demonstrated.

8 Acknowledgments

This work is funded the by National Science Foundation under grant #CDA-9214930 and Old Dominion University.

References

1. A report on undergraduate curricula for software engineeing. British Computer Society and Institution of Electrical Engineers, 1989.
2. ACM/IEEE-CS Joint Curriculum Task Force, 1991. Computing Curricula 1991.
3. The Case for Change. Commonwealth of Virginia, Commision on The University of The 21st Century, Richmond, Virginia, 1992.
4. Discipline review of computer studies and information sciences education, 1992.
5. Computer science curriculum. Department of Computer Science, University of Virginia, Charlettsville, Virginia, 1993.
6. P. A. Bailes and E. J. Salzman. A proposal for a bachelor's degree program in software engineering. In N. E. Gibbs, editor, *Software Engineering Education; SEI Conference 1989*. Springer-Verlag, New York, 1989.
7. B. Bruegge. Teaching an industry-oriented software engineering course. In C. A. Sledge, editor, *Software Engineering Education; SEI Conference 1992*. Springer-Verlag, New York, 1992. 65-87.
8. E. Epp. An experimental computer science laboratory. *SIGCSE Bulletin*, 23(1), 1991.
9. J. Foley et al. Report of the NSF Computer Science Education Workshop. *SIGCSE Bulletin*, 20(3), 1988.
10. P. Denning et al. Computing as a discipline. *Communications of the ACM*, 1, 1989.
11. R. E. Prather et al. A lecture/laboratory approach to the first course in computing. *SIGCSE Bulletin*, 10(1), 1978.
12. N. E. Gibbs. Software engineering and computer science: the impending split? *Education and Computing*, 7:111–117, 1991.
13. M. Kantioudi, J. S. Collofello, K. W. Collier, and S. Medeiros. Teaching an industry-oriented software engineering course. In C. A. Sledge, editor, *Software Engineering Education; SEI Conference 1992*. Springer-Verlag, New York, 1992. 323-338.
14. W. M. Lively and M. Lease. Undergraduate software engineering laboratory at Texas A & M university. In C. A. Sledge, editor, *Software Engineering Education; SEI Conference 1992*. Springer-Verlag, New York, 1992. 315-323.
15. A. J. Offutt and R. H. Untch. Integrating research, reuse, and integration into software engineering courses. In C. A. Sledge, editor, *Software Engineering Education; SEI Conference 1992*. Springer-Verlag, New York, 1992. 88-98.

16. L. J. Olson. A lab approach for introductory programming. In *Proceedings of the 14th SIGCSE Symposium on Computer Education*, Orlando, Florida, 1983.
17. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(2), 1972.
18. D. L. Parnas. Education for computing professionals. *IEEE Computer*, 23(1):17–22, 1990.
19. J. P. Penny and P. J. Ashton. Laboratory-style teaching of computer science. *SIGCSE Bulletin*, 22, 1990.
20. D. E. Ray. Course syllabus: CS250- Problem Solving and Programming. Department of Computer Science, Old Dominion University, Norfolk, Virginia, 1988.
21. J. Roberge and C. Suriano. Embedding laboratories in the computer science curriculum. *SIGCSE Bulletin*, 23(1), 1991.
22. M. Shaw. We can improve the way we teach CS students. *Computing Research News*, 4(1):2–3, 1992. letter to the editor.
23. J. Turner. private communication. Clemson University, 1993.
24. C. Wild, K. Maly, and L. Liu. Decision-based software development. *Software Maintenanace: Research and Practice*, 3:17–43, 1991.
25. W. A. Wulf. SE programs won't solve our problems. *Computing Research News*, 3(5):2, 1991. letter to the editor.

This article was processed using the LaTeX macro package with LLNCS style