

NSF Division of Undergraduate Education (DUE)
Course, Curriculum, and Laboratory Improvement (CCLI), NSF 01-58
Educational Materials Development (EMD)
Proof-of-Concept Proposal

C for Example

Project Summary

An undergraduate education is a critical link between high school and a society increasingly dependent upon computer technology. SMET majors increasingly need computer programming as a fundamental skill. Students effectively learn computer programming by using examples, yet the number of examples that can be presented in class and in a textbook is limited.

The cognitive approaches of collaborative and discovery learning supported by the intellectual scaffolding of a robust set of examples can give students a deep understanding of concepts they discover on their own. Assignment of computer programming problems just beyond a student's current skill level functions as the 'leading question', where carefully constructed examples give the student the information necessary to construct a mental model leading to the solution to the problem. This can help overcome the cognitive bottleneck where too few students negotiate the transition from lower-level skills to higher-level skills, from basic rote memorization and re-telling skills to more abstract knowledge synthesizing and transforming skills.

The difficulty lies in getting a complete set of the right examples. This proposal describes a web-based system to create and give intuitive access to between 300 and 500 examples illustrating the main points in the C programming language.

A prototype will be implemented in a pilot study over two years, being used with 300 undergraduate students taking an introductory programming course at UIC. Evaluation will be done using both web-based statistics and evaluation forms, comparing student skill levels and attitudes before and after the system's implementation.

Results from Prior Support

The PI currently has a \$50K startup-grant as a third-year continuation of NSF's Post-Doctoral Fellowship in Science, Math, Engineering, and Technology Education (PFSMETE), which will conclude 12/31/01. The first two years (at \$50K each) of the fellowship were spent working with the NSF-funded Center for Learning Technologies in Urban Schools under Louis Gomez. Two papers based on this work ("Web Page Annotator", and "Intelligent Indexing") were presented at the March 2000 International Conference on Technology and Education (ICTE) in Tampa, Florida.

During this third year of funding a web curriculum filter/editor has been created that allows students and teachers to edit any static web-based curriculum. This is significant since one in three U.S. colleges now offers an accredited degree online *. The filter/editor's changes are stored locally so that subsequent visits to the same page automatically have the changes applied. Applied to on-line curriculum, this can be used:

- By teachers to substitute a list of materials with a different list, or to share annotations asynchronously in designing on-line curriculum
- By students to add personal notes, cross-reference links, additional explanations and graphics to any web page. Page elements can be deleted as well.

The software functions as a plug-in to Internet Explorer and functions independently of the source or format of the web-based curriculum. It can be used with any static web page.

* Los Angeles Times, 3 March 2000.

Goals and Objectives

An undergraduate education is a critical link between high school and a society increasingly dependent upon computer technology. SMET majors increasingly need computer programming as a fundamental skill. Our goals are to:

1. *Capture the expertise* needed to answer questions of undergraduate students learning computer programming in C, representing this knowledge as a set of 300 to 500 programming examples;
2. *Index and cross-reference the examples* by topic, level of difficulty, and keyword;
3. *Create an intuitive web-based interface* to enable the captured knowledge to be used as intellectual scaffolding in a discovery-learning approach for students acquiring programming expertise.
4. *Evaluate the effectiveness* of the knowledge base and its use through prototype testing and user feedback.
5. *Disseminate* the knowledge base availability and project results through conference proceedings and newsgroups.

Detailed Project Plan

The Problem and the Solution

Students learn computer programming effectively using examples, yet the number of examples that can be presented in class and in a textbook is limited. A web browser can be used as an intuitive interface to give a student access to programming expertise that has previously been created and organized. The difficulty lies in getting a complete set of the right examples to answer students' questions.

Rapidly changing technology as well as research in Cognitive Psychology over the last two decades has given impetus to a pedagogical shift away from transmitting a body of expected knowledge to a more process-oriented approach¹. There has been a move from the behavioral

approach of direct instruction and explicit teaching² to the cognitive approach of collaborative³ and discovery learning.

Vygotsky proposed that all learning takes place in the 'zone of proximal development', where this 'zone' is the difference between what a student can do alone and what he/she can do with assistance. By building on the student's experiences and providing moderately challenging tasks, teachers can provide the 'intellectual scaffolding' to help students learn¹. Students are more to remember and have a deep understanding of concepts they discover on their own.

Discovery learning is “an approach to instruction through which students interact with their environment by exploring and manipulating objects, wrestling with questions and controversies, or performing experiments⁴.” For example, students using this approach to learn computer programming would be given an assignment currently beyond their programming skills (the ‘leading question’). Students would then ask questions (e.g. how can I display a two-dimensional board on the screen?) which an expert could answer using carefully constructed examples. These examples give the student the information necessary to construct a mental model leading to the solution to the problem.

The examples give enough information to answer the student’s questions, yet don’t trivially give the entire solution. Resolving this ‘cognitive dissonance’ is what leads students to incorporate learned concepts into a contextualized framework in long-term memory. Choosing a set of carefully crafted examples can help overcome the cognitive bottleneck where too few students negotiate the transition from lower-level skills to higher-level skills, from basic rote memorization and re-telling skills to more abstract knowledge synthesizing and transforming skills.

For this to be successful within the constraints of a traditional undergraduate introductory programming course the inquiry must be guided, and expertise must be readily available to answer students' questions by giving appropriate examples. In a large programming class (e.g. over 100 students) it is not feasible for the instructor and the teacher's assistants to provide detailed answers and examples to each students' questions. We can, however, capture this expertise and make it available as an exhaustive set of carefully constructed examples, accessible through the Internet.

Discovery learning is most successful when students have such prerequisite knowledge and undergo some structured experiences⁵. This can come from a textbook as well as from classroom lectures. This baseline of information gives students the vocabulary to ask questions, looking for examples to answer these questions. In even moderately sized classes (40 students) there isn't enough time during class, lab, and office hours for everyone to have their questions answered. Searches on the web often are ineffective for the beginning programmer since they lack the perspective to hone their searches. (e.g. Searching in Google for "C Programming Examples" yields many tutorials, but not necessarily sample programs.)

The average student is often frustrated in trying to find answers in the textbook because after one or two basic examples for each concept, the rest of the information in the book is described textually and not given as examples due to limitations of space[†]. Texts are meant as a jumping-off point. Reference manuals, on the other hand, require a level of mastery of the subject material usually not present until *after* the course is over (or even several courses). Textbooks that try to cover both bases end up being encyclopedic, where (in the author's experience) the size of the book is a disincentive to students to read it. This leaves a gap

[†] The *above-average* student on the other hand can handle this level of abstraction.

between basic programming texts and reference manuals, where average students try to get their questions answered by asking other people and looking for appropriate examples.

Presenting a complete set of examples on the web has several advantages over trying to do this in print. On the web any number of example programs can be given, organized into multiple layers of presentation. These examples can be cross-indexed according to concept, difficulty level, and related examples, as well as by keyword. This organization allows a user to get questions answered and explore in a self-paced asynchronous fashion. This can be done anytime & anywhere there is a network connection, exploring concepts to varying degrees of depth.

Interface Examples

The approach of creating a set of on-line examples as an intellectual scaffold for students learning introductory programming is already being implemented to a very limited extent by the author. Over the last 6 years of teaching the author has developed approximately 130 examples used regularly in an introductory programming course by several hundred students. (These can be viewed on the web at <http://logos.eecs.uic.edu/171>, under “Notes/Functions” and also under “Samples”.) Anecdotally the students report at the end of the semester that these examples were more valuable than the textbook. This project would enable the development of a more complete set of examples (we estimate 300 – 500 examples total), as well as providing a robust framework to cross-reference and index the examples. Implementing this using a database will also allow us to include a *generativity* component, where users can suggest new examples, highlighting the topics and keywords to be used in indexing.

C has been chosen as the language to use due to its widespread use. It is also a subset of C++, so the examples presented here will still be applicable to those introductory courses in C++ (excepting the discussion of inheritance and classes).

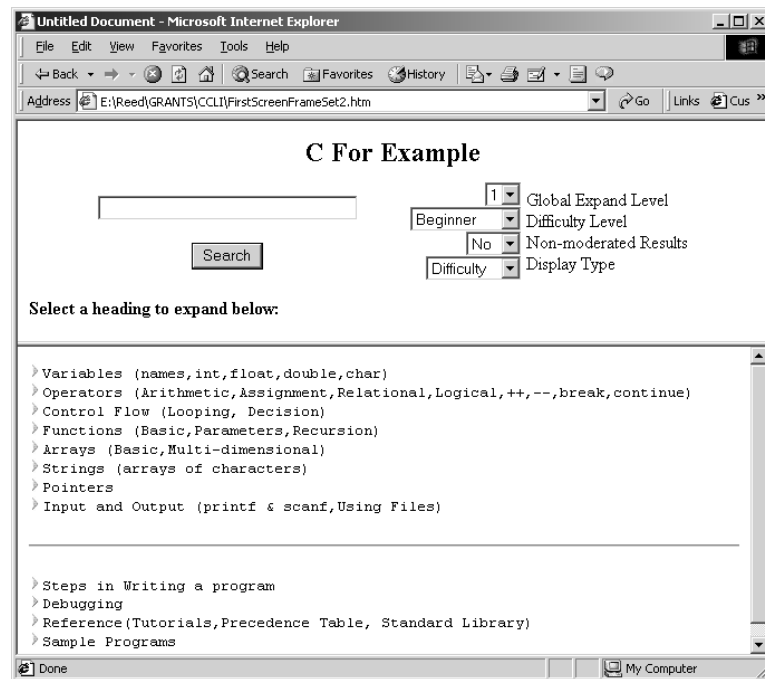


Figure 1

The initial screen in the *C For Example* system would look like the one shown in **Figure 1**. The top frame allows keyword searching, as well as setting the parameters affecting the display given in the lower frame. The *Global Expand Level* determines how many levels of an outline-format are displayed (currently 1). The *Difficulty Level* acts as a filter, determining which topics and examples will be displayed. The *Non-moderated Results* setting has to do with the generativity component, where users are allowed to submit their own examples. These submitted examples are subject to a moderator's review before being incorporated into the base set of examples. *Display Type* determines the order in which examples under a particular topic are displayed. They are either displayed in "Difficulty" order (easy to hard) or in "Popularity" order, with the most commonly accessed examples listed first.

It is important to keep in mind that the *Difficulty Level* is gauged relative to a student taking an introductory programming course[‡]. For instance the “Intermediate” difficulty level would also display the “Structures” and “Dynamic Memory” topics. Only the “Advanced” difficulty level would allow viewing the “go-to” statement, as this is not generally taught and its use is strongly discouraged.

The outline of topics can be selectively expanded. For instance selecting *Functions* (the 4th entry from the top) changes the display to look like the window shown in **Figure 2**. Note that the *Difficulty Level* has also been set to “Intermediate”, which adds the *Structures* and *Dynamic Memory* topics shown highlighted in the window.

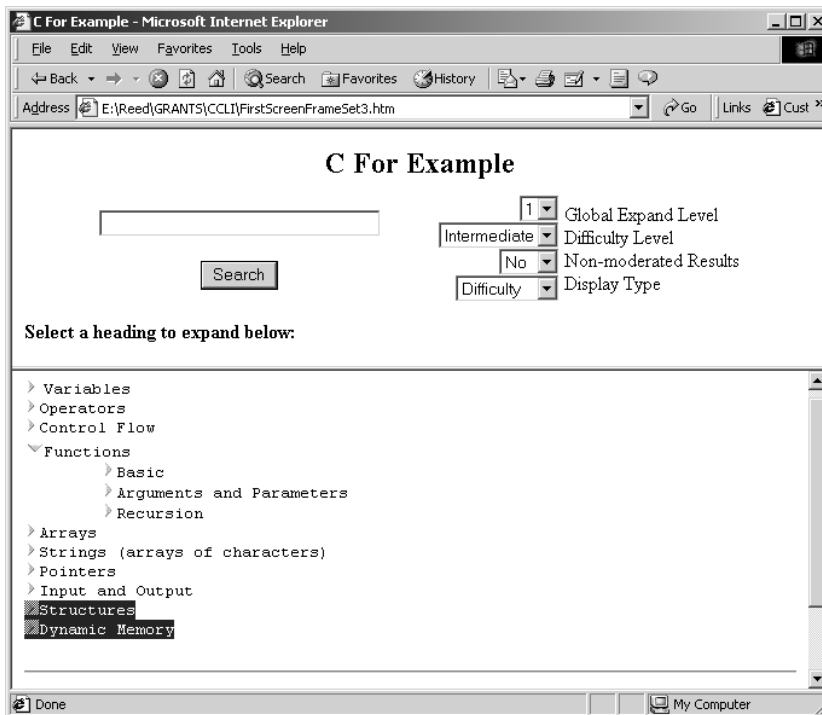


Figure 2

Consider an example where a student has been given a programming assignment that will end up being 500 to 1000 lines in length. In order to organize this program, the student must

[‡] ACM CS1 equivalent.

break it up into pieces called *functions*, where values are passed as *parameters*. The introductory student (let's call her *Kate*) would have seen the keywords *functions* and *parameters*, but might not fully understand their use. Kate's question might be "How do I break up a long program into pieces?" Kate could enter the keywords *functions* and *parameters* into the search field, or she could browse the topics starting with functions, giving the screen shown in **Figure 3**. The underlined entries are example programs.

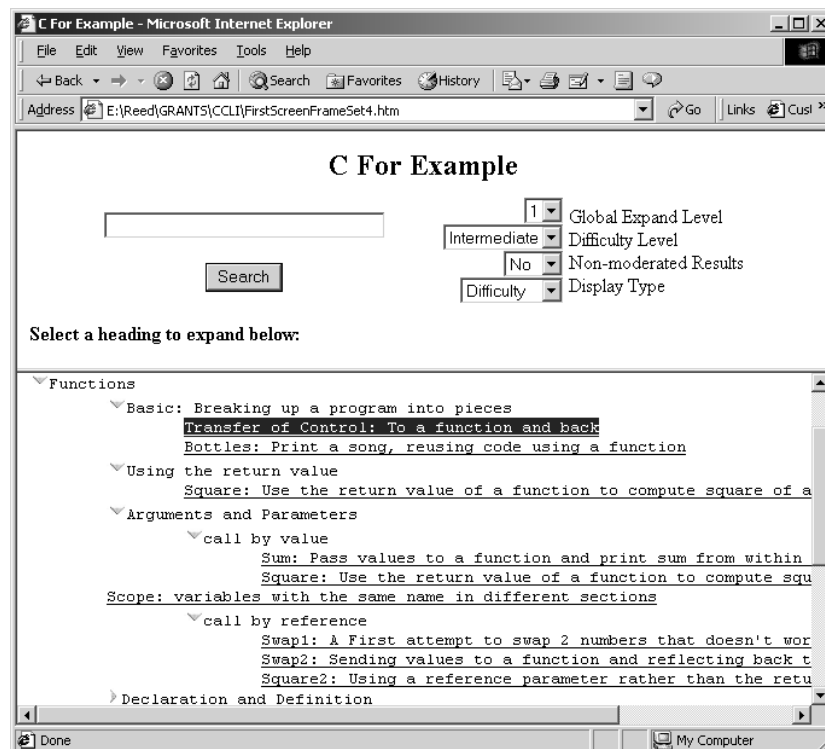


Figure 3

Selecting the highlighted "Transfer of Control" example displays it in another window as shown in **Figure 4**. The proposed interface is intuitive and easy to use, and will make it easy for a student to find examples to answer a question.

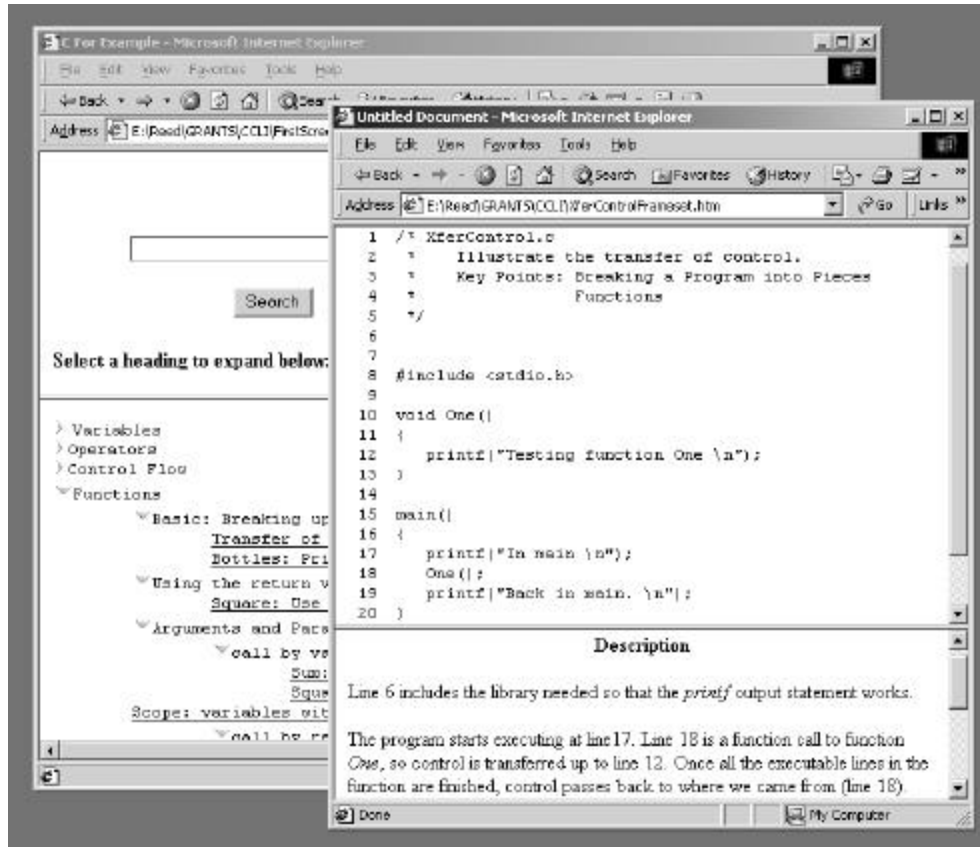


Figure 4

Technical Implementation Details

The system will be housed on a Dell PowerEdge server at the University of Illinois Computer Science department connected to the Internet at near T1 line speeds. The 300 – 500 examples will be served using a Microsoft Access database. The web server and scripting language will be either Apache with Java servlet scripting (via JDBC), or Microsoft Internet Information System (IIS) with Active Server Pages (ASP) scripting. On the client side state information will be stored using cookies, with JavaScript enabled to allow selective outline expansion as illustrated in the sample windows above.

Each example program will contain the text of the program, a header field with a list of the key concepts, and a text description. Once the database is working properly, the generativity

component will be added to allow users to submit their own examples. Text windows will be presented for the program text, description, and keywords, and the submitted program will be marked as un-moderated. After the submitted program is verified by a system moderator, it will be listed along with the other examples. Users can choose to list un-moderated examples as well by selecting that option on the main search page.

Time Table to Realize the Objectives

The proposed project will run for 2 years, across 4 academic semesters in the Computer Science department at the University of Illinois at Chicago. The PI will work closely with a Master’s Degree student who will work on the project 20 hours/week. Additionally 2 undergraduates per semester who are currently taking the introductory program course will work 10 hours per week. **Figure 5** shows a timeline of the principal activities. Database design (PI and MS student) will occur during the first half of the first semester, with a redesign period built in after the web-based interface is partially implemented. The web-based interface (PI and MS student) includes implementing the server-side scripting to get the web pages to talk to the database. It has 2 breaks during the student evaluation periods, where the system needs to be static. Indexing the contents will be done jointly by the PI and the MS student.

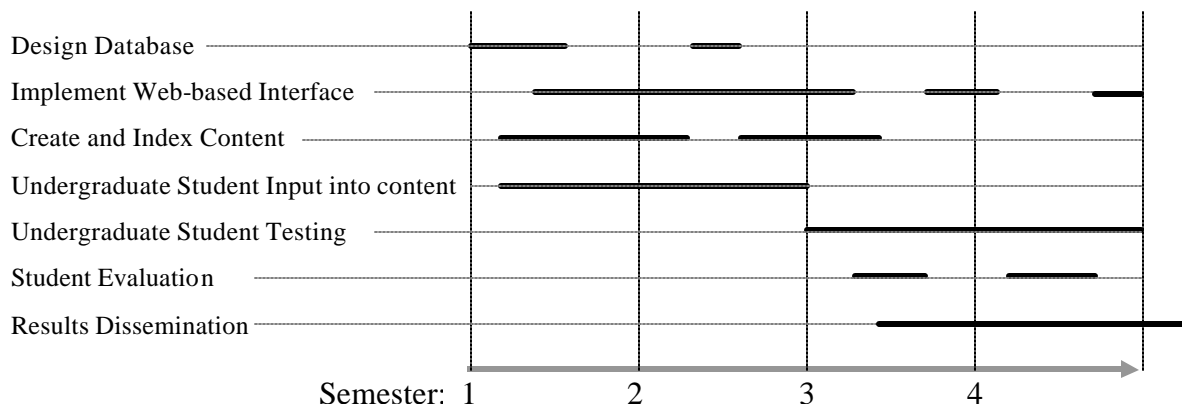


Figure 5

The undergraduate students will spend the first two semesters giving feedback as to the questions they have and whether or not they can find the answers using the examples supplied by the system. During the second two semesters the two students using the system each semester will give feedback on the usability of the system, and will do a formal evaluation at the end of each semester. During the last two semesters the PI will write up a description of the project, as well as the project results once the project is over. These results will be submitted for publication in Computer Education Conferences and newsgroups

Facilities and resources available

This project will be housed at the University of Illinois at Chicago (UIC) Computer Science Department. The test bed for the project will be the CS 102 course, which will have approximately 480 students over the course of 4 semesters. The department is committed to finding more effective learning environments, including web-enhanced courses and virtual environments. Current introductory Computer Science courses use the web extensively for synchronous and asynchronous presentation of material, student research, and turning in programs. The Computer Science Department is part of the College of Engineering, which has **55% minority students**. Housed in an urban setting, the College is committed to ethnic and gender diversity.

Experience and Capability of Principal Investigator

The PI has taught undergraduate introductory programming courses to roughly 1,700 undergraduate students over the last 12 years and has a demonstrated track record of commitment to technology in education.

From 1991 to 1996 the PI was the Principal Investigator for a bilingual Summer Science Camp (SSC) for 134 middle-school students, funded by NSF and DOE. This program was six-week full-time study and lab introduction to Computer Science hardware and software for 134 inner-city Latino middle school students. It included material normally not encountered until University-level Engineering courses and was an incentive for students to enter math & science based careers, resulting in *increased interest* in science, engineering and math careers *for 90% of the participants*. From 1998 to 2001 the PI was (is) a NSF Post-Doctoral fellow in SMET Education (NSF PFSMETE), working for the first two years with an NSF-funded Center for Learning Technologies in Urban Schools.

In the last two years teaching introductory programming at UIC the PI has pioneered the use of hands-on labs and hands-on testing, implementing this for a large lecture-format classes of 90 students. Hands-on testing in particular has been a challenge, designing tests that evaluate student's programming skill as opposed to just their programming knowledge. Additionally the PI has been developing a knowledge base of 130 online examples to help answer student questions, which can be seen at <http://logos.eecs.uic.edu/171>, under "Notes/Functions" and also under "Samples".

Evaluation Plan

The project evaluation has three components: 1. Gathering usage statistics of the online examples, 2. Detailed interface evaluation forms filled out by 4 undergraduate students during semesters 3 and 4, and 3. General evaluation forms filled out by undergraduate students using the system during semesters 3 and 4 of the project.

The usage statistics will give frequency of use of each example. These statistics will dynamically affect the behavior of the system as it progresses, since they will determine the ordering of presented examples when users select a *Display Type* of “Popularity” (see screen shots above). These statistics will show which examples are used, how many times they are used, and an approximation as to how many different students use them.

The detailed evaluation forms will be filled out by 4 undergraduate students who will spend 10 hours each per week using the system during semesters 3 and 4. They will report on:

1. A description of the question they had, and
2. What percentage of the time the answers to their questions were found in the knowledge base, as opposed to in their textbook or elsewhere.

The general evaluation forms will be filled out by the 300 undergraduate students who use this system as part of the CS 102 course. These evaluations will be filled out using “bubble forms”, where the students will be asked to answer the following questions using a rating of 1 to 5:

1. How often did you use the examples system to answer a question? (1 rarely - 5 often)
2. How often did you use your textbook to answer a question? (1 rarely - 5 often)
3. How well do you understand programming in C? (1 poorly – 5 thoroughly)
4. How confident are you that you could find the information you need to solve a programming problem? (1 not confident – 5 very confident)
5. How good of a programmer are you? (1 poor – 5 excellent)
6. How easy was it to learn programming in this course? (1 difficult – 5 easy)

The first semester these questions are asked (semester 2) the system will not yet have been deployed. The answers to these questions will then be compared to those of the students in semesters 3 and 4 to see if the system made a difference. The attitudes of the end users of the system will accurately reflect its usefulness.

An eventual measurement of success nation-wide could be measured by the number of links to the site, as reported by Google or other search engines. An additional metric would be the number of examples submissions to the generativity component of the system.

Dissemination of Results

The target audience for this project consists of 3 groups: 1. Computer programming Teachers, 2. Students, and 3. Computer science textbook publisher editors. Teachers and by extension their students can be reached through publications in computer education conferences and newsgroups (e.g. FIE, EDUCAUSE, ACM-SIGCSE, ASEE, ISECON, ICTE, and comp.edu). A description of the project could be submitted to one of the above conferences while the project is in progress, with the results submitted at the end of the 4 semesters. The PI has already had a conversation with McGraw-Hill regarding turning this into a commercial product, though the feasibility of this could more accurately be gauged at the end of the two-year project.

A moderate number of examples submissions could be handled by instructors teaching the introductory course here at UIC. There is also the possibility of an “open-source-like” shared responsibility of moderating submissions. If the number of examples submissions grows quickly, further funding would be required to successfully incorporate the examples.