Decoding Computer Science and Informatics

John Duncan  
Informatics, School of Informatics and Computing  
johfdunc@indiana.edu

Adrian German  
Computer Science, School of Informatics and Computing  
dgerman@indiana.edu

Suzanne Menzel  
Computer Science, School of Informatics and Computing  
menzel@indiana.edu

**Funding Level Requested:** Phase II

**Duration of Funding Period:** 1 year
Decoding Computer Science and Informatics Project

John Duncan, Adrian German, Suzanne Menzel

The purpose of this study is to investigate three related and pivotal bottlenecks to learning in Computer Science and Informatics. Students find it difficult to understand recursion, logical reasoning, and debugging, but this understanding is critical to success in these programs. Three instructors, Duncan, German, and Menzel, will collaborate in experiments to uncover their own tacit expert knowledge so that they can help students operate successfully in the discipline, applying the concrete strategies of Decoding the Disciplines. This study will use quantitative and qualitative student data, including classroom assessment techniques, pre- and post-tests, attitudinal surveys, and time and motion records to assess the value of their interventions. These experiments and assessments will, in turn, provide a better understanding of the epistemology that comprises Computer Science and Informatics and has broader implications for teaching and assessment in the field as well.

All concepts in a discipline are not equal. Some are threshold concepts—they require almost an intellectual conversion; failure to grasp them prevents students from functioning well within the discipline (Meyers & Land, 2006). Decoding the Disciplines provides a systematic framework for addressing such difficulties (Shopkow et al., 2012). Since 2010, faculty members in the Faculty Collegium on Student Success in the School of Informatics and Computing have undertaken the Decoding of their courses using this methodology. Each of the investigators will describe a source of conceptual confusion and our individual attempts to overcome it through exercises inside and outside of the classroom. Because it is difficult to see one’s own tacit knowledge and where one’s instruction is not clear, together we will share ideas and methods and serve as sounding boards as we each enter unknown territory, purposefully experimenting with some of the most difficult concepts in introductory computing. Our goal is to improve student learning outcomes in our separate courses, but also to develop and investigate assessment methods that can be used across an entire curriculum. These methods and the publications we propose will enhance the reputation of the entire School of Informatics and Computing.

We will report on the transformations of three early-curriculum courses, provide results of our experiments to date, and describe plans for future work with this funding.

1 Suzanne Menzel and Recursion in C211

The first course for computer science majors has, for over two decades, introduced the discipline using a functional model (Abelson & Sussman G, 1985). Students systematically learned the minutiae of programming (arithmetic, Boolean logic, if-expressions, list manipulation, etc.) in a progressive fashion. A different approach was attempted recently that placed the focus on algorithm construction; teaching students how to efficiently implement modern (and therefore relevant), groundbreaking algorithms by leveraging the power of recursion in a natural way. Recursion is central to all of computer science, and is the concept we quickly identified as a bottleneck.

1.1 Why is Recursion Hard?

To students, recursion is indistinguishable magic—some kind of circular reasoning. The sequence in which a recursive program executes its steps can also be hard to grasp at first. Finally, recognizing which variables are in scope (i.e., visible) at a given point in the execution process is key to understanding recursion. Recursion may also make use of nested data structures, which requires students to understand information storage concepts as well as the program’s execution.
1.2 Pedagogical Solutions
Much work has been done to design introductory programming projects that use “cool data” and we incorporated many of these ideas years ago into C211 (Simon et al., 2010). In most cases, however, the actual computational tasks involved in processing the data are disappointingly simple. Our new approach shifts the focus to “cool algorithms” which involve multiple stages and interesting data structures.

In spring 2012, we adopted a new text about algorithms that have “changed the future” (MacCormick, 2012). At the same time, we incorporated Team-Based Learning (TBL) (Michaelsen, Knight, & Fink, 2004), to promote accountability and foster a better sense of community in the classroom (Lasserre, 2011). Lecture, lab, and homework assignments were carefully recast to enable students to write programs implementing Google’s PageRank algorithm, a handwriting recognition algorithm, and a feature-preserving image resizing algorithm, just to name a few.

PageRank is the motivating example for introducing recursion in the third week of the course. The crux of the algorithm involves inverting the data structure representing the web so that it is keyed on the words rather than the addresses. We introduce the necessary programming constructs to accomplish the task “as needed”. This differs from the traditional approach to teaching programming, where the language governs the curriculum and instructors try to invent entertaining examples to illustrate fundamentally boring language details. In our approach, the algorithm shines as the star of the show and the introduction of language issues serves merely to support the implementation of the algorithm. Recursion pervades the problem, immediately heightening students’ curiosity and motivation to understand the concept.

1.3 Proposed Work
We believe we have improved student learning outcomes with this course redesign and our new focus on the crucial programming bottleneck. Having collected data from students in the fall 2012 and previous semesters, it remains to analyze the data and report results to the wider Computer Science Education community.

A common criticism of active-learning pedagogies such as TBL is that the course will be diluted and students will be less prepared for future courses. We plan to categorize the questions on the final exams (using Bloom’s scale and/or the SOLO taxonomy) over the past several years to demonstrate that the opposite is the case. Expectations have risen and students are more sophisticated thinkers at the end of the course.

We plan to examine the course drop rate for the years before and after the intervention to determine if an improvement can be seen. Additionally, our hypothesis is that students are simply more comfortable with programming at the end of the course. It seems they are less likely to leave a programming question unanswered on the examinations, as judged by the length of time it takes us to score the exams. Comparing the attempt rates of similar questions on the final exams of different semesters would help to answer the question.

2 Adrian German and Debugging in C212
Computer programming is the art of giving precise instructions to a computer; problem-solving in programming is notoriously non-linear. When it comes to programming one must not only find a solution, one also needs to be able to express that solution in the confines of a programming language. Writing a program generally starts with analysis and design, then an implementation stage, testing, and debugging. Debugging represents the stage where errors detected by the testing process are removed from the program.
Debugging can appear inscrutable to students, and is thus a bottleneck. Testing finds errors, while debugging localizes and repairs them. Just like proofreading, debugging is directly affected by the phenomenon of perceptual blindness: it’s hard for students, as authors of the program, to find their own errors in the program because one usually “sees” what one intended to write, not what one actually wrote. The differences can sometimes be agonizingly subtle. Students need to learn methodical testing habits in order to be able to debug effectively.

2.1 Why is Debugging Hard?
Debugging is hard for a variety of specific and objective reasons. For example, there may be no obvious relationship between the external manifestation(s) of an error and its internal cause(s); symptom and cause may be in remote parts of the program; changes (new features, bug fixes) in the program may mask (or modify) bugs. For students/novice programmers, these are huge hurdles, even though the programs written by novice programmers are fairly simple by expert standards. Students who are unable to master debugging will find difficulty progressing as computer scientists.

2.2 Pedagogical Solutions
The difficulties reviewed above are objective, and affect experts and student programmers alike. To address the debugging bottleneck, we will use Test Driven Development (TDD), a software development process pioneered by Kent Beck that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines the desired improvement or new function, then produces the minimal amount of code to pass that test, and finally refactors the new code to acceptable standards. Initial experiments (Erdogmus & Morisio, 2004) find that test-first students write more tests on average and, in turn, students who write more tests tend to be more productive and that quality increases linearly with the number of programmer tests.

2.3 Proposed Work
We plan to use pre- and post-tests in our second programming course to measure changes in attitude and efficacy in students using the TDD strategy. Both qualitative and quantitative assessments will be used. Anonymous responses describing the level of frustration perceived during development of testing/debugging code before and after will be collected. Furthermore, we also plan a controlled experiment in a closed lab in which students will be given a debugging task to solve that they haven’t seen before. Keystrokes from all workstations will be recorded and the normalized time per keystroke per student will be used to detect improvements in productivity.

3 John Duncan and Proofs in I201
Students in computer science and informatics are required to take a course in discrete mathematics. For computer science, the discrete math course (C241) focuses heavily on algorithmic complexity and proving program correctness. For informatics, the equivalent course (I201) highlights ways in which mathematics provides tools for solving practical problems. In both courses, proofs are used to teach students logical thinking and establish the means by which solution correctness can be assured. Even for students pursuing mathematics as a major, proofs present a challenging topic that requires explanation (Weber, 2001) (Goldberg, 2002). In areas such as informatics where the direct applications of proofs may escape students, contextualizing their purpose is even more important. Motivating students to excel in math can have profound consequences—students’ performance in math classes correlates with college graduation and increased lifetime earnings (Rose, 2001).

3.1 Why are Proofs Hard?
When confronted with many topics (including proofs) in a discrete math course, students may experience an emotional bottleneck where they have been conditioned by their experiences with the educational system to feel that non-applied mathematics are inherently less useful to them and, at the same time,
inherently more difficult than other material. Emotional bottlenecks are places where students have a strong emotional attachment to a particular way of thinking that may cause them difficulty with a learning concept (Middendorf et al., 2013). Some of this might result from the typical exposure to mathematics at the high school level in America, which focuses heavily on the trajectory leading up to calculus.

### 3.2 Pedagogical Solutions

Proofs serve two primary purposes to students in computer science and informatics. First, they are meta-tools. That is to say, they are tools with which we construct or understand other tools. When a theorem is presented to students, this is the equivalent of handing them a tool with which they can solve future problems. Proving the theorem establishes that this tool can be trusted to function correctly. Secondly, proofs establish the concept of justified reasoning, an important component to logic. The ability to present a clear chain of reasoning with solidly grounded justifications is an important skill in any career where decisions must be made or explanations given.

Finally, students in computer science and informatics can draw parallels between theorem proving and programming. Proofs use the same style of logically-connected statements that compose programs in most languages. In particular, proofs using techniques such as the principle of mathematical induction have a direct relationship to recursive programs. Proof techniques can even be used to demonstrate that programmatic solutions operate correctly. Concepts such as debugging apply to proofs as well as programs, as students often introduce “bugs” by taking unjustified or unsupportable steps while attempting a proof, such as dividing by zero.

### 3.3 Proposed Work

Currently, we are using tools such as minute papers to evaluate how student attitudes towards proofs change throughout the course. Student assessments will be used to design course materials that highlight the usefulness of logical thinking and theorem proving as a tool to solve other problems.

Numerous problems in discrete mathematics require logical reasoning to solve. Examples include the Knights and Knaves puzzle, the Cannibals and Missionaries puzzle, and the game of Sprouts. By having students apply the concepts they learn from proofs, they may be able to overcome their emotional and cognitive bottlenecks to better master the underlying concepts.

We propose to design and conduct experiments in which students must use logical reasoning to present a problem solution that they can prove is correct. This will include pre- and post-tests. Student attitudinal surveys from the beginning and end of the course will provide data about the changes in student attitudes towards proofs.
4 References


Budget Narrative

Phase II funding is $5,000. This money will be allocated in equal portions among the three investigators. We anticipate using the money for the following purposes:

- **$3,000**—These funds will be used to pay for the time we will spend compiling and processing our data, during which we will not be teaching.

- **$2,000**—These funds will be used to help pay travel expenses to attend and participate in a scholarly conference, such as SIGCSE (Special Interest Group in Computer Science Education, [http://www.sigcse.org/](http://www.sigcse.org/)), ITiCSE (Conference on Innovation and Technology in Computer Science Education, [http://www.cs.kent.ac.uk/events/iticse2013/](http://www.cs.kent.ac.uk/events/iticse2013/)), or FiE (IEEE Frontiers in Education, [http://fie2013.org/](http://fie2013.org/)). We plan to submit articles to academic journals based on this research.

If more detailed information is needed on fund distribution, please contact us.
Research Plan and Timeline

Spring 2013

The investigators are at different stages of this work. Menzel has conducted a complete redesign of her course and collected much data about this new version, so she is ready to move to analysis. Duncan and German will be developing new assessments and collecting their initial data, which includes planning and conducting the proposed testing. This is an important time period for the research team to meet regularly (current meetings are biweekly) for planning of innovative lessons and assessments. We will meet IRB requirements in relation to this research.

Summer 2013

With quite a few different kinds of data collected, Summer 2013 will be the time to dig into the analysis of our results. We intend to prepare a paper (or poster) submission for SIGCSE 2014. We are also considering designing a workshop or “birds of a feather” session to bring the Decoding idea to more educators.

Fall 2013

Having analyzed the data, we will prepare presentations for audiences such as the Collegium, the IUB SOTL program, and possibly the IU FACET conference. We will submit proposals to any applicable national conferences will later deadlines (or earlier ones occurring before summer). Having made such presentations, we will prepare publications to academic journals.