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 should have broader implications for teaching and assessment within the Scholarship of Teaching and Learning.
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/), ITiCSE (Conference on Innovation and Technology in Computer Science Education, http://www.cs.kent.ac.uk/events/iticse2013/), or FiE (IEEE Frontiers in Education, 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.
Investigator CVs

On the following pages we present CVs for the three investigators, in the order listed on the cover sheet.
John F. Duncan

3500 W Indian Creek Dr
Bloomington, IN 47403
Phone: 812-340-4161
Email: johfdunc@indiana.edu

Academic History:
- **Indiana University at Bloomington** 2011
  - PhD in Computer Science (Security), PhD minor in Cognitive Science
- **Indiana University at Bloomington** 2003
  - Masters in Computer Science
- **Washington and Lee University**, Lexington, Virginia 2001
  - B.S. in Computer Science, B.A.-equiv. major in Mathematics

Academic Publications:
- *What’s in a Session: Tracking Individual Behavior on the Web*, HyperText 2009
  - With Mark Meiss, Bruno Goncalves, Jose Ramasco and Filippo Menczer
  - With L. Jean Camp and William R. Hazlewood
  - With L. Jean Camp and William R. Hazlewood
- *Homophily In Web Browsing Behavior*, (submission pending)
  - With Mark Meiss and L. Jean Camp
- *Conducting an Ethical Study of Web Traffic*, Cyber Security Experimentation and Test (CSET) 2012
  - With L. Jean Camp
- *Ethical Design for Security and Privacy* (dissertation)

Personal Information
- Birth Date: May 25, 1980
- Citizenship & Permanent Residence: USA

Awards and Honors:
- Fellow, 2009-2010, Indiana University Center for Applied Cybersecurity Research (CACR)
- Inaugural Member, Alpha Chapter of Iota Nu Phi, Informatics Honor Society

Professional History:
*Indiana University at Bloomington*
- Lecturer, School of Informatics and Computer Science 9/2012 – current
  - I201 – Mathematical Foundations of Informatics - 2 semesters
  - I130 – Introduction to Cybersecurity - 2 semesters
  - I210 – Information Infrastructure I - 2 semesters
- Visiting Lecturer, School of Informatics and Computer Science 9/2010 – 8/2012
  - C241 – Discrete Structures for Computer Science - 3 semesters
- I231 – Math of Cybersecurity - 2 semester
- I130 – Introduction to Cybersecurity - 4 semesters
- A201/A597 – Introduction to Programming I - 1 semester
- I201 – Mathematical Foundations of Informatics - 1 semester
- I210 – Information Infrastructure I - 2 semesters
- I211 – Information Infrastructure II - 1 semester

- Associate Instructor, Computer Science 8/2001 - 5/2010
  - Course Instructor
    - A111 – Survey of Computers and Computing - 6 semesters
    - A112 – Programming Concepts - 6 semesters
    - A113 – Data Analysis - 1 semester
    - A114 – Introduction to Databases - 1 semester
    - I130 – Introduction to Cybersecurity - 2 semesters
    - I231 – Computational Foundations of Cybersecurity - 1 semester
  - Lab Instructor
    - A111 – Survey of Computers and Computing - 4 semesters
    - A112 – Programming Concepts - 4 semesters
    - A201 – Introduction to Programming I - 1 semester
    - A216 – Digital Multimedia Concepts and Technologies - 1 semester
    - I399 – Health, Technology and Aging - 1 semester
    - P535 – Pervasive Computing - 1 semester

- Research Assistant, E.T.H.O.S., IU Informatics Fall 2008
  - Duties: Web programming, database design, web application design, application maintenance, application testing, data migration, user testing
  - UITS, Data Management Services - Database Analyst 6/2001 – 10/2005

Washington and Lee University, Lexington, Virginia

Technology Skills:
- ColdFusion 7 years experience
- JavaScript 7 years experience
- SQL/TSQL 7 years experience
- PHP 1 year experience
- ASP 1 year experience
- Access 7 years experience
- SQL Server 5 years experience
- Oracle 1 year experience
- Python 3 years experience
- Java 3 years experience
- C++ 2 years experience
- C 2 years experience
- C# 1 year experience
ADRIAN GERMAN
821 East 10th, SoIC Undergrad Annex Bloomington, IN | 812 855-7860 | dgerman@indiana.edu

EDUCATION
Indiana University Bloomington
MS in Computer Science 1994

Bucharest Polytechnical Institute
MS in Control Engineering and Computer Science 1987

Lic. Matei Basarab
Baccalaureate in Mathematics and Informatics 1981

AWARDS
Trustees Teaching Award 2007 – 2008
CSCI Department Teaching Excellence Award 2002 – 2003
Teaching Excellence Recognition Award (TERA) 1999 – 2000

In addition I have been nominated by third-parties (School, Department or my students) several times for the

- Trustees Teaching Award (twice) and the
- Student Choice Student Award (twice).

BOOKS
A Computable Universe 2012
Co-edited with Hector Zenil.

Randomness Through Computation 2011
Co-edited with Hector Zenil

The Fundamentals of Web Programming 2006
Pearson Education

Soliloquy (in Java) 2005
A programmatic instruction handbook developed with John Wiley and Sons.


CONFERENCES
SIGCSE 2013 March 2013
Chair of the Peer Instruction paper session.

SIGCSE 2012 March 2012
Chair of the Collaborative Learning paper session.

SIGCSE 2010 March 2010
Chair of the Assessment paper session.

2008 Midwest NKS Conference October-November
2008

Midwest Theory Day March-April 2006
Chair and Organizer, with Fan Chung-Graham a Special Conference Guest.

2005 Midwest NKS
Co-chair and Organizer.
Guests: Ray J Solomonoff (in person) and Stephen Wolfram (videoconference from Boston)

I have also been an active member of several other SOTL events on campus including the Course Development Institute.

PUBLICATIONS AND PAPERS

*Extreme Scaffolding in the Teaching and Learning of Programming Languages*

*SIGCSE 2008 (Portland, OR): BOF Session on Multiplayer Games in CS1/CS2*
Session prepared and facilitated along with student Bjorn Ottesen 2008

*The Net Worth of an Object-Oriented Pattern*
International Conference on Parallel and Distributed Systems, Los Angeles 2004

*RMI: Observing the Distributed Pattern*
Frontiers in Education (FIE) Conference, Savannah, GA 2004

I have had other presentations and publications at national and/or local conferences from 1997-2003.

EVENTS ORGANIZED

Collegium Video Presentation with Prof. John Hattie from Melbourne, Australia 2012

Collegium Video Presentation with Author Geoff Petty from London, England 2011

I have also helped organize and spoke at several NKS conferences:

- 2004 in Boston,
- 2006 in Washington DC,
- 2007 in Burlington, VT at the University of Vermont

MEMBERSHIPS

SIGCSE, SIGPLAN, SIGOPS, ACM SIGCSE and ITiCSE Reviewer
Suzanne Menzel

Contact Information
Lindley Hall 215 4965 Inverness Woods Road
150 South Woodlawn Avenue Bloomington, IN 47401
Bloomington, IN 47405-7104
(812) 855-0778 (812) 360-406
menzel@indiana.edu

Education
1985 Completed all course work and qualifying examinations for Ph.D. in Computer Science, Rutgers University.
1983 M.S. in Computer Science, Rutgers University.
1981 B.S. in Computer Science and Mathematics, Douglass College, Rutgers University.
Graduated with Honors in Computer Science.

Appointments
2004 - present Senior Lecturer in Computer Science, Indiana University, Bloomington.
2010 - present Co-Editor of the ACM-W Newsletter
1992 - 2004 Lecturer in Computer Science, Indiana University, Bloomington.
1990 - 1991 Lecturer in EECS, University of Illinois, Chicago.
1989 - 1990 Visiting Lecturer in Computer Science, Indiana University, Bloomington.
1985 - 1989 Lecturer in EECS, University of Illinois, Chicago.
1984 - 1985 Instructor in Computer Science, Rutgers University, New Jersey.
1981 - 1984 Teaching Assistant in Computer Science, Rutgers University, New Jersey.

Publications
Awards

2012  Inducted into the Faculty Colloquium on Excellence in Teaching (FACET).
2012, 2011, 2010  Received the Women in Computing (WIC) Inspirational Teacher Award.
2011, 2004  Received the Trustees Teaching Award for Lecturers, Indiana University.
2011  Nominated for TechPoint Mira Award, Contribution to Education.
2007  Nominated for TechPoint Mira Award, Education Contribution to Technology - Individual.
2009, 2006, 2004  Received the Computer Science Departmental Teaching Excellence Award, Indiana University.
2011, 2006, 2005, 1998  Nominated for Student Choice Award for Outstanding Faculty
(from Student Alumni Association), Indiana University.
2003  Received IU Professional Development Award to attend SIGCSE 2003.
1999, 1998  Received departmental Teaching Excellence Recognition Award (TERA), Indiana University.
1989  Nominated for Silver Circle Teaching Award, University of Illinois.
1983  IBM Research Fellow, Rutgers University.

Grants

2008  Received $15,000 NCWIT award to implement hands-on activities for Just Be roadshow presentation (with Kay Connelly).
2008  Received $8,000 IPRE award to purchase Scribbler robots, and develop CS1 robotics curriculum (with Kay Connelly and Kent Dybvig).
2007  Received Avon Hello Tomorrow award for week of Dec 7, a $5,000 cash award to fund initiatives to empower women.

Conferences/Workshops Organized

- 2/9 - 2/10 2012, Indiana Celebration of Women in Computing (InWIC) co-organizer of a statewide conference, attended by undergraduate and graduate student, faculty, and industry representatives. Sponsored by the Anita Borg Institute, Microsoft, and the NSF. (http://www.cs.indiana.edu/inwic)
- 2010 and 2011, Indiana Award for Aspirations in Computing, co-organizer and Selection Committee Chair for a regional award to identify and reward high school girls for their accomplishments and aspirations in computing.
- 2/5 - 2/6 2010, Indiana Celebration of Women in Computing (InWIC)
- 5/29 - 5/30 2008, CSTA Roadshow Workshop for over 40 participants from across the nation who are interested in developing a K-12 Roadshow program at their school. Hosted by Google in Mountain View, CA.
- 4/12 2008, Bring IT On! reunion for students and faculty from ten Indiana schools with fledgling outreach programs, who had taken part in the 2007 workshop on the IU campus, to report on their progress and learn about next steps. (http://www.cs.indiana.edu/bringiton)
- 10/26 - 10/28 2007, Bring IT On!, a three-day diversity workshop for students and faculty from schools across Indiana, co-organizer; 32 participants visited IU to learn how to create their own K-12 outreach program for their schools and to explore post-graduate opportunities for themselves.
- 10/20 - 10/22 2006, Bring IT On!, a three-day diversity workshop for students from HBCUs, co-organizer.
- 20 from 10 different HBCUs participated
- 9/29 - 9/30 2006, Midwest Celebration of Women in Computing, conference co-chair, Greencastle, IN, held in conjunction with CCSC-MW, this was a prototype for a second model of a regional women in computing conference. (http://www.cs.indiana.edu/midwic)
- 4/8/06, Computing Outreach in Indiana (COIN) Workshop co-organizer. Participants learned to create a K-12 outreach program using Just Be as a model.
• 2/3 - 2/4 2006, Indiana Celebration of Women in Computing (InWIC) co-organizer of a statewide conference, based on CICWIC 2004, but with broad participation from over 100 women from many small schools across Indiana as well as large research institutions (UI and Purdue) and nationally known corporations such as Google, Microsoft and Lilly. (http://www.cs.indiana.edu/inwic)

• 11/11 - 11/12 2005, Java Education for Teacher Training (JETT), for high school teachers of AP computer science in the midwest to learn about object-oriented programming and Java. Other objectives include (1) the free exchange of teaching strategies, ideas, experiences, and materials, (2) building a regional community of computer science educators, and (3) attracting young students from underserved groups, such as girls and minorities, into computing.

• 11/5 - 11/6 2004, Java Education for Teacher Training (JETT)

• 10/31 - 11/1 2003, Java Education for Teacher Training (JETT)

• 2/20 - 2/21 2004, Central Indiana Celebration of Women in Computing (CICWIC) co-organizer of a low cost, regionally tailored supplement to the nationally known Grace Hopper Celebration of Women in Computing conference. The conference involved students and faculty from Butler University, DePauw University, Rose-Hulman Institute of Technology, Indiana University, and Purdue University together with industry leaders for a local gathering at McCormick’s Creek State Park to focus on issues of women in computing.

Teaching

CS0: Introduction to Programming for Non-majors
CS1: Introduction to Computer Science (various versions using Scheme, C, C++, and Java)
CS2: Introduction to Software Systems
Foundations of Computer Science
Digital Design
Computability

Talks, BoFs and Workshop Presentations

• Recursive Thinkers and Doers in CS, with Joseph Cottam, Poster Presentation, SIGCSE 2012.

• Recursive Thinkers and Doers in Introductory Computer Science, with Joseph Cottam, Poster Presentation, CITL, Indiana University, 2011.


• Introducing the NCWIT Academic Alliance, CCSC, Chicago, Oct 2009.

• Leveling the CS1 Playing Field, with Gloria Townsend and Katie Siek, SIGCSE 2007.

• Demystifying and Degeekifying Computing Careers through K-12 Outreach, with Katie Siek, Jennifer Franko, Samantha Foley, and Laura Hopkins, SIGCSE 2007 workshop.

• Peer Tutoring Works, with Sarah Loos, I600 Pedagogy and Professionalism course.


• Doors to Diversity, TECS workshop, Washington University, July 2006.

• Demystifying and Degeekifying Computing Careers, with Katie A. Siek and Jennifer Franko, Paul Munger Conference for Youth-Serving Professionals, June 21, 2006.
  • Creating a Regional Celebration of Women in Computing (R-CWIC) conference, with Gloria Townsend and Benita Bair, March 3, 2006, SIGCSE. We shared our experiences in organizing the Indiana regional conferences, InWIC 2006 and CICWIC 2004, in an effort to inspire Women in Computing organizations and also individuals to seriously consider holding a regional celebration in their area.
• *Binary Trees and Recursion*, JETT, Indiana University, November 11, 2005.

• *The Under-representation of Women in IU Computer Science Undergraduate Degree Program*, with Beth Plale, April 8, 2005, presented the final report of the Task Force on Undergraduate Curriculum and Gender to the IU faculty. The report is available online at http://www.cs.indiana.edu/~pplale/documents/G-UGreportFinalv4.pdf.

• *Java Workshop for APCS High School Teachers*, led one-day event at Stonehill College, Massachusetts, May 16, 2005.

• *Java Collections*, JETT, Indiana University, November 5, 2004.


• *Central Indiana Celebration of Women in Computing (CICWIC): a Successful Pilot for a Small Regional Conference (SRC)*, with Gloria Townsend, DePauw University, Beth Plale, Indiana University, and Barbara Clark, Purdue University. *Birds of a Feather* session at Grace Hopper Celebration of Women in Computing conference in Chicago, Oct 6 - 9, 2004.

• *Basic Java and Timers, Threads, and GUIs*, JETT, Indiana University, October 31, 2003.

**Professional Memberships**

ACM-W  Leadership Council Member since 2008

Member of the WIC@IU Executive Committee since 2006

Lifetime member of the Association for Computing Machinery (ACM),
ACM Special Interest Group in Computer Science Education (SIGCSE), and
Computer Science Teachers Association (CSTA)
February 15, 2013

Dear Members of the SOTL Grant Review Committee,

I am pleased to provide this letter of support for the SOTL Grant application from John Duncan, Adrian German, and Suzanne Menzel. The proposed work on Decoding Computer Science and Informatics is of great interest and importance to the computing education field of study, as well as taking an important step forward in understanding and advancing student success.

The School of Informatics and Computing is committed to excellence in teaching, and has invested great efforts in establishing the Faculty Collegium on Student Success, which provides an environment for faculty to develop a community of reflective practitioners. The Faculty Collegium has benefited from the support of CITL for many years, especially in the early stages of group formation with the input of Joan Middendorf.

The great idea behind the bottle-neck approach is that by addressing these issues in classes, retention and success of students in majors also increases. The courses aimed at in this project are “bottle-necks” for our curriculum, in the sense that students that do not succeed in these courses are increasingly unlikely to continue in the major. I have confidence in the people proposing this project, and optimism for the project’s success.

Regards,

Dennis P. Groth
Associate Dean for Undergraduate Studies
1. Learning is Hard Work. Teaching is Hard Work

Two are the essential prerequisites for successful learning: (a) the knowledge (skill, information) needs to be accessible to the learner throughout (in the sense that learning occurs in the zone of proximal development) and (b) we need a motivated learner (in other words the learner has to want to acquire the knowledge, or skill). In an institutional setting (such as a university, or college, where students’ schedules are more flexible) class prerequisites, when enforced, aim to place students of similar ability in the same class. This is meant to simplify the task for (in the sense of maximizing the effectiveness of) the instructor. Though the range of professional duties performed by teachers is wide and extensive, at the heart a teacher’s role is the promotion of learning for all students: using the most effective teaching and learning strategies, and making assessment an integral part of their teaching while maintaining high expectations for all students, they need to find ways to motivate their students (and keep them motivated). Let’s state here our assumption that most students start the semester eager to learn. But in spite of everybody’s efforts (instructors and students alike) this genuine (initial) desire to learn soon fades away\(^2\) for some, when confronted with the harsh reality of learning: learning, turns out, is hard work [1]. This is not to say that the entire burden of proof should be placed on the learner’s shoulders; indeed we emphasize the central, privileged role of the learner in the process. The responsibility for success is shared equally by both teacher and student, because if learning is hard, teaching can’t be easy. Thus we get to a thesis that we will revisit again and again later: a good teacher is, in fact, so much more than just a (text)book, no matter how well written and organized that text may be.

2. The School Syndrome. A Paradox

As authors and historians point out (most notably in [2], [3]) our educational system is highly accessible, radically unequal and organizationally fragmented. Its purpose seems to be to offer equal opportunity\(^3\) to education. School reformers have acted as agents for the society, seeking to use school to create capable citizens and productive workers and to cure our social ills. In contrast to reformers, individual consumers of education have seen schools less as a way to pursue grand social designs than as a way to pursue intensely personal dreams of a good job and a good life. The choices of educational consumers have always overwhelmed top-down efforts at school reform. Individual families seek to use schools for their own purposes—to pursue social opportunity, if they need it, and to preserve social advantage, if they have it. In principle, we want the best for all children. In practice, we want the best for our own. As Labaree puts it: “Americans may not be well educated, but we are well-schooled. [...] At heart, this is a story grounded in paradox [and largely the result of the] tension between our social goals and our personal hopes. We ask schools to promote equality while preserving privilege, so we perpetuate a

---

1 SOTL Phase II Grant Report (August 15, 2014 version, curator Adrian German)
2 Much like New Year resolutions: a 2007 study found that 88% of those fail, in spite of 52% of the participants being certain of success at the beginning of the study (Blame it on the brain, Jonah Lehrer, WSJ 2009).
3 But so do any type of playoffs – where half the teams actually don’t make it past the first round.
system that is too busy balancing opposites to promote student learning. We focus on making the system inclusive at one level and exclusive at the next, in order to make sure that it meets demands for both access and advantage. In the end, the system does what we want as consumers, even if it doesn’t do what we ask as reformers [...]. As a result, we find ourselves trapped on an educational treadmill of our own making, running hard just to stay in place.”

3. Teaching and Learning. In the Classroom

One reason that school reformers struggle to have an impact on student learning is that the organization of the school system makes it hard for reforms to get past the classroom door. But another reason is that, even inside the classroom itself, teachers themselves struggle to get their students to learn. For student learning to take place, teachers must first establish a special kind of personal relationship with the individual students in the class. Without this kind of relationship, students will not learn what schools want them to learn. And teachers can only establish such pedagogically effective relationships if they are allowed the discretionary space to do so (see [4], [5]). They need this latitude to figure out a way of doing things that works best for the individual students in the class and for the special situations of time and place. But intruding on teachers in this way threatens to undermine the degree of teacher discretion that is necessary to foster effective learning. We’re not saying that this is the reality of our current classroom (either in general or in our specific individual cases) but if education is indeed (or turns into) an equal opportunity zero-sum game it is important for learners to become aware of it so they can better plan for that into their efforts. In other words we’re saying that the learner is not and cannot be a passive element of teaching and learning but plays a central, active role in it. As such no reform will ever succeed if all parts don’t feel invested and that automatically includes the learners.

There is another aspect that compounds the difficulty of learning beyond the potential zero-sum game aspect of public schooling and the specifics of the “street-level bureaucracy” mentioned earlier: and that has to do with the peculiarities of the learning process itself. It is a well documented fact [6] that educators across the country are frustrated with the challenge of how to motivate the ever increasing number of freshmen students entering college who are psychologically, socially, and academically unprepared for the demands of college life. But college life is comprised of many college courses and if

4 Teachers fit the occupational category that Michael Lipsky calls “street-level bureaucrats”. These are public service workers whose clients are non-voluntary, who function under conditions of crushing demand and inadequate resources, where goals are ambiguous or conflicted and where performance in relation to goals is hard to measure. In cases like these (police officers, social workers, teachers), the bureaucracy has no choice but to allow the front-line agent substantial discretion to decide how to apply general policies to the myriad peculiarities of the cases at hand. From this perspective, then, school reform at the classroom level may not only be difficult; it may be counterproductive. And a key reason that teachers often resist reform efforts may be that they are trying to preserve a form of teaching and learning that seems to work and to fend off an alternative approach that might not (As Michael Fullan has noted, it is just as helpful to schooling to block a harmful reform as it is to implement a beneficial reform.)

5 Such students often exhibit maladaptive behavior such as tardiness, hostility towards authority, and unrealistic aspirations. The standard approach is to address the problem as an academic issue through remedial or developmental instruction. Developmental education programs however do not address the whole problem. Lack of motivation is not limited to the academically weak student. When students have both a lack of academic skills and lack motivation, the greater problem is motivation. Faculty often have neither the time or inclination to
students come unprepared for the demands for college life they might just be unprepared for the demands of the college classroom as well. Are they supposed to know how to behave in college already? Perhaps not. Then they must learn. And who better suited to teach them if not their teachers. It follows that teachers are (or should be) more than interactive textbooks: beyond the subject matter that they teach they need to help their students succeed in the classroom and that includes (a) learning but also (b) learning how to deal with the rigors of a college course and eventually (c) learning how to learn.

4. Significant Learning. Powerful Learning

It is so natural for adjectives like “significant” or “powerful” to be used to describe the kind of learning we’re aiming for that one might not realize that, in time, these adjectives have come to acquire very specific meanings. In particular author Dee Fink [8] has created a new taxonomy of learning, called “A Taxonomy of Significant Learning” (see also [9], [11]). As Fink puts it: “It was with this thought in mind that I worked on and eventually put together a new taxonomy of learning, one that is called a Taxonomy of Significant Learning [...] My own view is that this new taxonomy can be seen as a successor to the classic taxonomy, the well-known taxonomy of educational objectives formulated by Benjamin Bloom and his associates in the 1950’s.” One of the six kinds in his taxonomy is Learning How to Learn. We have already advocated for making the learners aware of the specifics of the learning process earlier.

Another author, Ronald S. Brandt [10] gives more concrete characteristics of the contexts in which people learn best. He lists ten conditions for “powerful” learning to occur. Here are the first four:

1. What they learn is personally meaningful
2. What they learn is challenging and they accept the challenge
3. What they learn is appropriate to their developmental level
4. They can learn in their own way, have choices and feel in control

address difficult motivational issues in the classroom, consequently, the task of trying to effectively motivate such students often falls to academic advisors. Opinions about the role of motivation in academic achievement and what can be done about it vary widely among college faculty, administrators, and student services professionals. Consideration about unmotivated students opens a Pandora’s box of questions: Can anything be done about these students? Can motivation be taught? What kind of strategies can be used to influence motivation? Is this time wasted that might better be used on those students who are already motivated? These are very important questions.

One of the first tasks teachers face when designing a course is deciding what they want students to learn or get out of their course. Students will always learn something, but good teachers want their students to learn something important or significant, rather than something relatively insignificant. This leads to a question that is key to the whole teaching enterprise: What are the ways in which learning can be significant? If we have or can develop a language and a conceptual framework for identifying the multiple ways in which learning can be significant, then teachers can decide which of various kinds of significant learning they want to support and promote in a given course or learning experience.

This occurs when students learn something about the process of learning itself. They may be learning how to be a better student, how to engage in a particular kind of inquiry (e.g., the scientific method), or how to become self-directing learners. All of these constitute important forms of learning how to learn.
The point we’re trying to make is that it is impossible to separate the learner from these conditions: without a learner these conditions can’t even be stated, or expressed.

In summary thus far: students want to succeed but they need to be careful, patient and resilient as they walk the path of learning. What can be done to ensure (or increase) the success of this enterprise? Teachers need to aim to teach more than just the subject matter of their course but it is only reasonable that efforts to increase learning should in fact start there.

5. The Scholarship of Teaching and Learning

After thirty years or more\(^8\) of the scholarship of teaching and learning (SoTL) many students still struggle to learn at the university level, while ever more techniques are being developed to help students learn and measure their success. In practice, however, any efforts to reshape classes begin with questions such as: “How can I make use of this new technique?” Or, “How can I increase my students’ critical thinking?” Such questions are often too broad to provide a clear focal point for designing efficient strategies. They sometimes draw attention to parts of the course not in great need of reform, and generally focus the process on the means (teaching) rather than the end (student learning). Decoding the Disciplines [12] is an evidence-based, learner-centered methodology that has been shown to conclusively improve teaching by prioritizing what really matters in making meaning in a discipline, and boosting learning outcomes in the classroom.

6. Decoding of Computer Science and Informatics

The “Decoding the Disciplines” model [13] takes advantage of the differences in thinking among academic fields in order to decode each individual discipline. Following the model, faculty start by answering a series of questions to define crucial bottlenecks to learning, dissect the ways an expert deals with the issues that causes the bottleneck, and invent ways to model this thinking for students.

\(^8\) As Middendorf, Pace, Diaz and Shopkow put it in Decoding of Disciplines, An Overarching Strategy: “There was a time when educators had to scramble to find conceptual tools that could be used to help in the classroom. Now, after several decades of intensive work by educational researchers and scholars of teaching and learning, we have at hand an extraordinary collection of strategies for increasing learning. Collaborative learning, problem-based learning, just-in-time teaching, team-based learning, and myriad other approaches have brought faculty developers into a new realm in which the wealth of learning ‘therapies’ begins to rival those available to their counterparts in health care. [...] This abundance of strategies and concepts, however, raises new challenges. Whereas instructors once had to operate like the proverbial carpenter, whose only tool was a hammer, they now find themselves in the midst of a great superstore filled with a mind-numbing variety of implements. Faced with so many possibilities, we need a system for deciding which tools are most appropriate for a particular task. And we need a means to connect this ever more complex literature on teaching and learning with the practical decisions to be made by instructors who have limited time to invest in grand theoretical explorations of teaching and learning. We need a meta-strategy, a broader context in which to mediate between the learning needs of a particular group of students and the growing set of possible interventions. We argue that Decoding the Disciplines can provide such a meta-strategy. It can provide both the faculty developer and the instructor with a framework to define the task at hand and a means to determine which tools would be most useful in accomplishing it. Decoding can, thus, serve as a bridge between theory and practice and allow us to make maximum use of the educational riches that lay before us.”
After giving students an opportunity to practice these skills and receive feedback, each professor assesses student performance on these basic operations.

Decoding the Disciplines is a powerful technique that goes far beyond the (somewhat obsessive) observational learning that normally represents the bulk of a traditional lecture. Though it may take time to fully appreciate the technique the results are often impressive: (a) wanting to develop a certain type of thinking, recognizing when it’s not there and giving more examples for students to emulate is not the same as (b) truly locating that type of reasoning and presenting it to the students like in a dissection – as a separate, distinct cognitive process. At first Decoding of Disciplines may appear as an obvious, somewhat self-evident technique. But what its creators have developed is in fact a very sophisticated introspection technique similar to psychological analysis because just like in that area the breakthrough is achieved by bringing the unconscious (i.e., some ingrained expert technique) into the consciousness (via e.g. skilled guidance). However we need to acknowledge that with every cohort the challenges are different, even though as they still remain the same on some meta level. New students come with new (implicit) narratives and different backgrounds. As a result what we did in the previous semester to achieve effective teaching may not be entirely and directly applicable, in the exact same form, during the next. What remains the same though is the process of questioning ourselves and our students while we try to establish the coordinates of the specific strategy for that cohort and semester – to determine student narratives and to convey the expertise to them by taking into account their implicit assumptions and specific backgrounds.

Decoding then starts anew with every cohort and we are true believers in the method. The one additional aspect we want to emphasize is this: in Decoding of Disciplines introspection is only limited to the expert. We think that it’s important for the learner to participate and contribute to the introspection process as well. We will explain why and how we reached this conclusion. If this aspect were not as important as we think it is then (as said before) the instructor could easily be replaced by a textbook.

7. Recursion, Proofs, Testing and Debugging. Test-Driven Development

The purpose of our study was to investigate three related and pivotal bottlenecks to learning in Computer Science and Informatics: students find it difficult to understand (a) recursion (or recursive thinking), (b) logical reasoning (or the specifics and level of detail needed for proofs, along with when a proof is in fact needed), and testing and debugging (alone or as part of program development), but this understanding is critical to success in these programs. We decided to collaborate in experiments to uncover our own tacit knowledge so that we can help our students operate successfully in the discipline, by applying concrete strategies of Decoding the Disciplines. We envisioned that our study would 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 our interventions. These experiments and assessments would, in turn, provide a better understanding of the epistemology that comprises Computer Science and Informatics and should have broader implications for teaching and assessment within the Scholarship of Teaching and Learning.

9 Or any street level bureaucrat for that matter.
7.1 Recursion. A Technique. How

Drawing on previous work [14] we developed a strategy for explaining recursion to students. Preliminary analysis of the data indicates that our method was successful. The emphasis here should be on the type of bottleneck we wanted to solve: students could not master a specific technique. Using all the steps in the Decoding process we have made it easier for students to acquire this technique.


The second bottleneck has a slightly ambivalent nature: the “how” is still present in that students indeed need to acquire techniques of proving statements and programs. At the same time there is a certain amount of “why” involved: if the statement seems obvious it may not be clear to the student why a proof is necessary in the first place. This bring us to the third bottleneck.

7.3. Debugging. Test-Driven Development. Why/How

A program is essentially a theorem that states (or ensures) a certain relationship between inputs and outputs. Because of its constructive nature there’s little motivation on the part of the student to build the program while checking for, or attempting to preserve, its correctness. Writing a program in that sense is very much like writing an essay. As a result the outcome is often in drastic disagreement with the intentions: the program does not work, the student starts to panic, identifying and removing the errors takes the aspect of a frantic race against the unknown and ends in tears, anger, frustration, humiliation and shame.

Debugging is a very systematic activity that can be very successful but like medical intervention takes time and patience. Students usually have little of any of these when the assignment deadline is rapidly approaching. Furthermore they are not willing to accept that their “carefully” built program does not work since their mindset is on what they wanted the program to do not what the program is in fact doing. Debugging is necessary when maintaining or dealing with legacy programs (programs written a long time ago by someone else). When developing a new program debugging needs to be integral to the development process from the very beginning; in a very real way debugging as part of test-driven development has, in fact, a true prophylactic nature.

Using the seven steps of the Decoding of Disciplines process we set up a solution for this bottleneck. We demonstrated this in class (during lecture and lab) consistently and encouraged students to practice it when developing their assignments. To our surprise very few students chose to follow our instructions. The major hurdle seemed to be this: students saw this systematic development as too slow, boring and ultimately unnecessary. They didn’t think that they needed to go through all these steps. Most were sure that they’d be able to do things faster if they approached it in their own way. So, as it turned out, approaching this bottleneck revealed another one of a slightly different nature: students preferred a fuzzy (unknown) future cost to a known, immediate cost when developing a program. Though we were giving them a manual of instructions on how to design programs they wouldn’t even open it. We started
to come up with metaphors for this kind of thing: it immediately brought to mind the old RTFM [15]. This is what happens when people start to assemble furniture or toys without looking at the instructions shipped with the product: usually it takes longer and the result is very poor. It is not clear if this is a consequence of being lazy or not trusting the instructions or is due to some other reason. Another similar situation that comes to mind is the established stereotype that men don’t ask for directions. In that regard some of the hypotheses advanced were: men prefer to learn by doing, not by being told what to do; men want to win (aggressive); men want to be strong (insecurity). If that too was the case with our bottleneck it was clear that we first had to change the culture. It was equally clear to us that students were not perceiving the development process as a process similar with rock climbing in which every step needs to be secured before moving on to the next step. Instead their perception was that writing the program was more like writing an essay in some foreign (natural) language. Students would prefer writing the entire program on the computer and compiling at the end then debugging until it worked (or, as was the case more often then not, until it didn’t).

7.4. The Market Value of an Unfinished Assignment

There are conflicting opinions (even among the three of us) on how to properly address (i.e., grade) an unfinished assignment. One view is that a program that does not compile immediately deserves a zero. The alternative view takes into account that one can take a perfect program and change one character in it thus making it syntactically invalid: does the resulting (not working) program deserve a zero even if the fix is trivial? This second perspective also takes into account the possibility that the program submitted by a student might be readable and quite brilliant when read by a human but unusable on a computer due to compilation errors. Submissions of this kind would be eliminated (and perhaps a lot of the potential learning subsequently rejected) if we’d make the explicit requirement that submitted programs need to compile. So we thought that first it was necessary to establish an operational measure for an unfinished assignment. We succeeded and the resulting measure was both very operational and concrete although it left room for some subjective interpretation; this later proved to be a very useful feature. Having established a way to grade an unfinished assignment we then gave the students the possibility to appeal their grades. During appeals (always in person) students had to come with a line of reasoning that showed that a better grade was in fact deserved. These discussions proved

---

10 The RTFM comment is usually expressed when the speaker is irritated by another person’s question or lack of knowledge. It refers to either that person’s inability to read a technical manual, or to his perceived laziness in not doing so first, before asking the question. This initialism appeared in print in 1979 on the Table of Contents page of the LINPACK Users’ Guide in the form “R.T.F.M.” -- Anonymous, suggesting that it was already well established.

11 From http://www.multicians.org/thvv/realprogs.html Real Programmers never use comments or write documentation: “If it was hard to write”, says the Real Programmer, “it should be hard to understand.”

12 “Days and weeks of programming will save us hours of planning” we use to say in those situations.

13 But that was indeed our ultimate goal and the list of instructions on test-driven development we had developed earlier, if followed, would have guaranteed it. So the ultimate goal was the same. Dealing with unfinished assignments was a just the result of having to deal with the cultural resistance to following instructions, not a goal in itself.

14 This has been described elsewhere, in presentations to the Collegium and various workshops and could be included here if necessary.
to be very effective instruments of learning\textsuperscript{15} even though they were very time consuming. We decided that we had to find a way to make them part of the assignment itself.

7.5 Self-Assessment.

Having determined that inviting students to have an opinion about their grade (after the fact, as part of the appeal process) was useful, we decided to move this interaction earlier in the process. We asked students to collect this evidence proactively, as they were building the assignment. We told them that this would act as insurance for when they had to come and argue that their grade needs to be higher and present evidence for that. Students didn’t mind, and most actually enjoyed, negotiating their grade this way. It was a polite, casual (semi-formal) and deliberate meeting of minds. It seemed natural that they might also consider it useful to collect this evidence and make a case for their grade in advance: it gave them more time, and a chance to make their cases stronger. Little by little, slowly but surely the tide started to turn: those students that would normally come for an appeal would now act proactively and show their peers what they were doing. More students were doing test-driven development (rock climbing) than writing free prose (essays) in their programs. The seed had been planted. There were still assignments that were turned in unfinished but they all had to be accompanied by a short document describing their worth – an itemized list of reasons describing that precisely. While the self-assessment part was mandatory students (a) had the opportunity to redo any assignment that had been turned in on time and (b) were encouraged to include in their self-assessment write-up anything and everything they wanted to include, all types of comments were accepted. Several things happened:

- Without knowing students had overcome the barrier from the second bottleneck (proofs). Instead of asking “why” it is necessary to prove anything that seems obvious, they were now happy to do it simply because someone else was questioning/wanted the evidence.
- The quality of the assignments went up, the number of unfinished assignments went down, the type and quality of tests (evidence) started to range from the very simple, to the more general to even tests that exhibited meta-cognition. We give some examples in an appendix.
- Students who couldn’t finish the assignment started writing about all sorts of things in their self-assessment pieces. Eventually valuable information about their own narratives that we hadn’t even hoped to collect started to appear in these documents (first from students who were very frustrated, then from those that were more successful now than they had ever been before).
- Some students continued to submit short and inconclusive self-assessments (and a very small fraction actually resisted it). Far from being a disappointment it helped us quickly identify those types of students that needed additional assistance/help with this.

Critical thinking is thinking that assesses itself. To the extent that our students need us to tell them how well they are doing, they are not thinking critically. Didactic instruction makes students overly dependent on the teacher. In such instruction, students rarely develop any perceptible intellectual independence and typically have no intellectual standards to assess their thinking with. Instruction that fosters a disciplined, thinking mind, on the other hand, is 180 degrees in the opposite direction. Each

\textsuperscript{15} At this point the situation strongly evoked Ronald S Brandt’s list of conditions for “powerful” learning.
step in the process of thinking critically is tied to a self-reflexive step of self-assessment. As a critical thinker, we do not simply state the problem; we state it and assess it for its clarity. We do not simply gather information; we gather it and check it for its relevance and significance. We do not simply form an interpretation; we check our interpretation to see what it is based on and whether that basis is adequate.

Because of the importance of self-assessment to critical thinking, it is important to bring it into the structural design of the course and not just leave it to episodic tactics. Virtually every day, for example, students should be giving (to other students) and receiving (from other students) feedback on the quality of their work. They should be regularly using intellectual standards in an explicit way. *This should be designed into instruction as a regular feature of it.*

There are two kinds of criteria that students need to assess their learning of content. They need universal criteria that apply to all of their thinking, irrespective of the particular task. For example, they should always be striving for clarity, accuracy, and significance. Of course, they also need to adjust their thinking to the precise demands of the question or task before them. If there are three parts to the task, they need to attend to all three parts. If the question requires that they find specialized information, then they need to do just that. One simple structure to use in attending to this dual need is to provide students a set of performance criteria that apply to all of their work, criteria that they will be using over and over. Then, make specific provision for encouraging students to think in a focused way about the particular demands of any given task or question before them [17].

We already knew [18] that self-assessment contributes to higher student achievement and improved behaviour. It was also known that while self-grading (n.b., specifically, not self-assessment in general) appears to result in increased student learning, peer-grading does not [19]. It was now clear to us that self-assessment was a necessary part of instruction. We had been lucky enough to identify a variety that was easy to implement while also acting as a catalyst in removing our original bottleneck. But for all its benefits [16] it was not clear that self-assessment is easy to teach (in general).

8. The Secret of Self-Regulated Learning

Self-regulated learning is like your own little secret. It stirs from within you, and is the voice in your head that asks you questions about your learning [20]. More formally, self-regulated learning is the conscious planning, monitoring, evaluation, and ultimately control of one’s learning in order to maximize it. It’s an ordered process that experts and seasoned learners like us practice automatically. It means being mindful, intentional, reflective, introspective, self-aware, self-controlled, and self-disciplined about learning, and it leads to becoming self-directed. Just because we may practice self-regulated learning doesn’t mean our students do. Most of us were among the best students, especially in college, and the best students can become the worst teachers because we quickly knew how to master the material. In fact, few of our students demonstrate self-regulation – not even those in professional schools. When asked to identify the factors they considered important in their learning, 132 veterinary students most commonly cited the quality of their faculty’s instruction, not their own effort or learning skills [21]. Not

---

16 Presumably opening up the door to self-regulated learning mentioned in Fink’s taxonomy.
surprisingly, younger, undergraduate students have the same mind set. They see learning as something that is “happening” to them, and our job is to make it happen and make it easy. After all, learning was easy in elementary and high school, so why should it require much time and hard work now?

Another secret about self-regulated learning is its strong positive impact on student achievement. Just the cognitive facet of it, metacognition, has an effect that’s almost as large as teacher clarity, getting feedback, and spaced practice and even larger than mastery learning, cooperative learning, time on task, and computer-assisted instruction [22].

How do you get students to practice self-regulated learning? First, you explain to them what it is and how it will benefit them and then have students do self-regulated learning activities in class and as homework. Then you wait for them to see the good results. We decided to integrate self-regulated learning in our courses [23] and to identify places where we may have been doing that already.

9. Fall 2013 Celebration of Teaching

Our poster [24] “Three Metaphors for Three Related (and Pivotal) Bottlenecks in Computer Science” was presented at the Fall 2013 Celebration of Teaching in the Decoding of Disciplines Section.

10. SIGCSE 2014 (Atlanta)

Our presentation “How to Decode Student Bottlenecks to Learning in Computer Science” was accepted as part of the 2014 SIGCSE Technical Symposium in Atlanta, GA [25] (also [26] [27]).

11. Budget. Unfinished Business

We spent most of the money on preparations, travel and presentations to conferences. There’s still some money left that were originally saved for additional research, for example Brian Rak has designed and partially implemented a Java plug-in editor for Eclipse that would record (and later play back) all keystrokes that a programmer would type while developing the program. Brian was not paid, instead his development was part of an Independent Study Course he took with one of the authors (AG). The instrument, when finished, would be able to reveal some qualitative information on how students go about writing programs during a test.

12. The Future

The new direction (including self-assessment as part of a larger goal to integrate self-regulated learning) was discussed at a meeting on Wed Jun 11, 2014. We discussed adding student portfolios as a running self-assessment assignment.
A. Appendix

This appendix collects a few examples of the kind of tests we have seen in our classes since we introduced self-assessment as mentioned in section 7.5 above. Assume the goal is to design a function that calculates the sum of the first $n$ integers where $n$ is a positive integer. Such a program could be defined as such:

```
(define (sum n)
  (apply + (build-list (add1 n) values)))
```

Student would write three categories of tests.

1. Elementary tests.

In this category we have simple one to one associations between input and output (base cases):

```
(check-expect (sum 10) 55)
(check-expect (sum 0) 0)
```

Notice this makes no statement about how the function works.

2. Proof-level tests.

In this category the student would provide considerable evidence of program’s correctness.

```
(define n (random 100))
(check-expect (sum n) (/ (* n (+ n 1)) 2))
```

Note this is a more powerful statement that ties into the second bottleneck (proofs).

3. Meta-cognitive tests.

This category includes one-step expectations that lead to the discovery of recursion solutions:

```
(define n (random 100))
(check-expect (sum n) (+ n (sum (- n 1))))
```

Note how this now ties into the first bottleneck directly (recursion).

This last category of tests is the most valuable of them all. One step-expectations immediately lead to recursive proofs and subsequently into proofs by induction tying them all together into one valuable learning experience.
References

[1] Why don’t students like school? (Daniel T Willingham)
[2] Someone has to fail: the zero-sum game of public schooling (David F Labaree)
[3] Treating education as a zero-sum game (Michael Brooks, Peter DeWitt; Education Week)
[14] Recursive Thinkers and Doers in CS1 (Joe Cottam and Suzanne Menzel, SIGCSE 2011)
[18] Writing Effective Assignments (Adrian German, Collegium Presentation, March 20, 2013).
[22] Visible Learning, Hattie