WCCCE 2009

Proceedings of the 14th Western Canadian Conference on Computing Education

Simon Fraser University,
Burnaby, B.C. , Canada
May 1-2, 2009
http://www.cs.sfu.ca/wccce2009

In cooperation with:
ACM SIGCSE

Edited by:
Roelof Brouwer, *Thompson Rivers University*
Diana Cukierman, *Simon Fraser University*
George Tsiknis, *University of British Columbia, Vancouver*

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York New York 10121-0701

ACM COPYRIGHT NOTICE. Copyright © 2009 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept., ACM, Inc., fax +1 (212) 869- 0481, or <permissions@acm.org>.

For other copying of articles that carry a code at the bottom of the first or last page, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, +1-978-750-8400, +1-978-750-4470 (fax).

Notice to Past Authors of ACM-Published Articles
ACM intends to create a complete electronic archive of all articles and/or other material previously published by ACM. If you have written a work that was previously published by ACM in any journal or conference proceedings prior to 1978, or any SIG Newsletter at any time, and you do NOT want this work to appear in the ACM Digital Library, please inform <permissions@acm.org>, stating the title of the work, the author(s), and where and when published.

ISBN: 978-1-60558-415-7

Cover artwork by Danyu Zhao.

A Word from the WCCCE Steering Committee Chair

Welcome to the 14th Western Canadian Conference on Computing Education (**WCCCE 2009**)!

Thanks to **Simon Fraser University** for hosting the Western Canadian Conference on Computing Education for the first time.

I would like to thank all members of the **WCCCE 2009** Steering Committee, all members of the Conference Committee, all members of the Local Organizing Committee, as well as all authors and panelists, reviewers, session chairs, exhibitors, workshop organizers, sponsors, keynote speakers, and the invited speakers for their constant support in organizing this conference.

It was very encouraging to receive a large number of high quality submissions for papers, panel discussions, and workshops from all over Canada, USA, and the rest of the world. The Conference Committee members and the reviewers faced quite a challenge deciding what to accept/reject for presentation.

It gives me a lot of pleasure that we will be joined by four speakers of national and international fame. I am sure that all delegates will enjoy the very informative presentations by our two keynote speakers (**Dr. Bjarne Stroustrup & Dr. Gary Poole**) and two invited speakers (**Dr. Yan Xu and Mr. Paul Denny**) on Friday and Saturday. Thanks to all speakers, once again!

We are delighted that for the **first time** in the history of this conference, WCCCE will run in cooperation with **ACM** and **SIGCSE**. As part of this **in cooperation status**, the proceedings will be included in the **ACM Digital Library**.

In addition to members of all the committees, **WCCCE 2009** would not have been possible without the untiring efforts of a number of other volunteers. On behalf of the Steering Committee, I would like to express my gratitude to all the volunteers for attending to all the minute details, spending many hours orchestrating the conference website, art design, and publicity, putting together the conference proceedings, and performing an enormous number of other tasks, in spite of their many other commitments.

Finally, I am *personally* very thankful to **Dr. Diana Cukierman** for her excellent work (in particular, in obtaining the **in cooperation status**) and professional leadership throughout this conference. She has been a dynamic and continuous source of inspiration for all committees. In fact, she deserves much more than this small note of thankfulness.

Welcome to Simon Fraser University to attend the 14th Western Canadian Conference on Computing Education (**WCCCE 2009**)! Once again, thanks to all the authors for sharing their interesting works with us.

The **15th** Western Canadian Conference on Computing Education (**WCCCE 2010**) will be held in May 2010 at the **UBC Okanagan, Kelowna, BC**. I hope to see you all next year!

Sincerely,

Surinder Dhanjal, P. Eng.
(Chair, WCCCE Steering Committee)
Department of Computing Science
School of Advanced Technologies & Mathematics
Thompson Rivers University
Kamloops, BC Canada
V2C 5N3

Welcome from the Conference Committee

It is with great pleasure that we welcome you to the 14th annual Western Canadian Conference on Computing Education. This is the first time that the conference is offered in cooperation with the Association of Computing Machinery (ACM) and the ACM Special Interest Group in Computing Science Education (SIGCSE). The cooperation allows us to publish the proceedings in the ACM digital library and has strengthened the international appeal of the conference.

We hereby thank our keynote speakers, the invited speakers and all the authors for sharing their findings with us. The conference received a large number of high quality papers on a variety of interesting and important topics. To accommodate the large number of papers that were submitted, the committee decided to hold parallel sessions and to extend the conference by one day. The program also includes a number of special events like high quality workshops and panel discussions.

We extend our appreciation and gratitude to our reviewers for their excellent reviews, constructive comments and their patience with the review process. Also a number of volunteers from SFU provided much needed help at various stages during the past year. The committee expresses its gratitude and appreciation to each of them for attending to all of the details.

Finally, we extend our gratitude to our hosts, SFU, and the School of Computing Science, to our academic and corporate sponsors for their continuing support, and to all the conference participants for their valuable discussions and interactions.

Welcome to Burnaby!
Sincerely,

Conference Committee:

Roelof K. Brouwer, *Thompson Rivers University*
Diana Cukierman (Chair), *Simon Fraser University*
George Tsiknis*, University of British Columbia, Vancouver*

Table of Contents

WCCCE 2009 Committees

Steering Committee

Diana Cukierman
Simon Fraser University

Surinder Dhanjal, Chair,
Thompson Rivers University

Paul Franklin
University of the Fraser Valley

Rick Gee
Okanagan College

Michael Zastre
University of Victoria

Conference and Local Organizing Committee

Greg Baker
Simon Fraser University

Roelof Brouwer
Thompson Rivers University

Diana Cukierman, Chair
Simon Fraser University

Anne Lavergne
Simon Fraser University

Steven Pearce
Simon Fraser University

Craig Scratchley
Simon Fraser University

George Tsiknis
University of British Columbia, Vancouver

Acknowledgements

Wendy Addison
Simon Fraser University

Pam Borghardt
Simon Fraser University

Toby Donaldson
Simon Fraser University

Philippe Giabbanelli
Simon Fraser University

Yudong Liu
Simon Fraser University

Steve Wolfman
University of British Columbia, Vancouver

Danyu Zhao
Simon Fraser University

WCCCE 2009 Sponsors

Sponsoring institutions at Simon Fraser University

SFU COMPUTING SCIENCE

SFU FACULTY OF APPLIED SCIENCES

SFU VICE PRESIDENT ACADEMIC

irmacs

In Cooperation with

acm

SIG CSE

Corporate Supporters

Gold

RIM®

NOKIA

Microsoft® Research

Silver

PEARSON

WILEY

Mc Graw Hill McGraw-Hill Ryerson

WCCCE 2009 Reviewers

Don Acton
University of British Columbia, Vancouver

Wayne Babinchuk
Thompson Rivers University

Patrice Belleville
University of British Columbia, Vancouver

Roelof Brouwer
Thompson Rivers University

Charles Brown
University of Northern British Columbia

Diana Cukierman
Simon Fraser University

Surinder Dhanjal
Thompson Rivers University

Rick Gee
Okanagan College

Ted Kirkpatrick
Simon Fraser University

Ed Knorr
University of British Columbia, Vancouver

Patricia Lasserre
University of British Columbia, Okanagan

Donna McGee Thompson
Simon Fraser University

Kevin O'Neill
Simon Fraser University

Janice Regan
Simon Fraser University

Craig Scratchley
Simon Fraser University

Ora Steyn
University of the Fraser Valley

George Tsiknis
University of British Columbia, Vancouver

Steve Wolfman
University of British Columbia, Vancouver

Keynote Talk

Teaching in a Windstorm: Setting educational goals in a rapidly changing discipline*

Gary Poole
Centre for Teaching and Academic Growth
University of British-Columbia
Vancouver, BC, Canada
gary.poole@ubc.ca

Abstract

In this session, we will look at the implications of teaching in a discipline in which some of the knowledge changes rapidly. It has been said that some of the information presented in the first month of a computing science course is obsolete by the fourth month. This means that an important set of educational goals must address the development of lifelong learning skills and the ability to stay up on a rapidly changing field.

Categories & Subject Descriptors: K.3.2 **[Computers and Education]**: Computer and Information Science Education.

General Terms: Experimentation.

Bio

Gary has spent a considerable amount of time working with people who are dedicated to students' learning, first at Simon Fraser University, then as President of the Society for Teaching and Learning in Higher Education, as a 3M Teaching Fellow, and at the Centre for Teaching and Academic Growth of which he is currently the director. He is also a member of the Department of Health Care and Epidemiology, where he teaches medical and dental students, and conduct research investigating factors that affect people's ability to cope with cancer.

* Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
ACM 978-1-60558-415-7

Special Session

PeerWise – students sharing and evaluating their MCQs

Paul Denny
Department of Computer Science
The University of Auckland
paul@cs.auckland.ac.nz

Abstract

PeerWise is a web-based learning tool that supports students in the creation of MCQs with model answers and associated explanations. The questions are contributed to a shared repository, where they can be answered, rated and discussed by other students. The repository serves not only as a drill-and-test library for students, but also as a creative medium for engaging students in critical reflection and deep learning. This talk will present a summary of student utilization of PeerWise in several Computer Science courses and highlight potential benefits of its use to students and instructors.

Categories & Subject Descriptors: K.3.1 **[Computers and Education]**: Computer Uses in Education.

General Terms: Human factors

Keywords: MCQ, peer assessement, question test bank, PeerWise

Bio

Paul Denny teaches introductory computer science and computer programming in the Faculties of Science and Engineering at the University of Auckland in New Zealand, since 2000. He is a member of the Computer Science Department's Education Research group and of the ACM Special Interest Group on CS Education. He is a recipient of the 2004 Faculty of Science Dean's Award for Distinguished Teaching and the 2008 University of Auckland Teaching Excellence Award for Innovation in Teaching. He was also nominated for the national Tertiary Teaching Excellence Awards for 2009.

Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
ACM 978-1-60558-415-7

Pedagogical Transformations in the UBC CS Science Education Initiative

Donald Acton, Kimberly Voll, Steven Wolfman, Benjamin Yu
University of British Columbia
Department of Computer Science
2366 Main Mall
Vancouver BC V6T 1Z4
{acton, kvoll, wolf, benyu}@cs.ubc.ca

ABSTRACT

The UBC CS Science Education Initiative (CSSEI) has resulted in a number of research projects. New teaching methods and student assessment instruments are introduced to engage student learning and evaluations of their understanding. In this paper, we report four of these recent initiatives and their initial findings.

Categories and Subject Descriptors

K.3.2 [**Computers and Information Science Education**]: Pedagogy, education research – *just-in-time, BRACElet, in-class activities, attitudinal survey, problem-based learning, analysis.*

General Terms

Experimentation, Measurement.

Keywords

Computer education, transformation, code understanding, survey analysis, grade analysis.

1. Introduction

The UBC CS Science Education Initiative (CSSEI) is part of the five-year, $12 million project at the University of British Columbia under the Carl Wieman Science Education Initiative (CWSEI), which began in 2007. The project aims to dramatically improve undergraduate science education by taking a four-step, scientific approach to teaching: 1) establishing what students should learn through the identification of learning goals, 2) scientifically measuring what students are actually learning, 3) adapting instructional methods and curriculum, and incorporating effective use of technology and pedagogical research to achieve desired learning goals, and 4) disseminating and adopting what works for student learning. In the UBC CS department, we have developed learning goals for five of our core courses [3]. We have also identified some of the difficulties students face in achieving these learning goals. This paper describes some of our ongoing research projects to help students better achieve these learning goals, and our initial results.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

2. Just-in-Time Teaching

Previous studies of our CPSC 121: Models of Computation course have shown that students have trouble connecting topics and retaining core ideas from term to term, and from lectures to labs. Many students found the hands-on labs (often circuit-based) disconnected from the more theoretical material discussed in class. Downstream courses also tend to extensively review (re-teach) material that students learn in the course. These problems remain despite significant development efforts on the course by many faculty over several years.

In two sections of this course offered in the January to April 2009 term, we have shifted towards a heavily problem-based lecture style enabled by Just-in-Time-Teaching (JITT) [5].

The core tenets of JITT are two-fold. Before a class, instructors assign reading and other preparatory work to students and expect them to complete it. During the class, instructors avoid re-covering the textbook material and instead use exercises to engage students in higher-level discourse and learning of the material. The key mechanism that feeds both elements of this process is some kind of assessment (e.g. a quiz) just before class. This quiz motivates students to do their preparation and enables instructors to assess how well students learned the preparatory material and where to focus attention during the classroom work.

In CPSC 121, each (possibly multi-day) lecture ends with the preparatory assignment for the next lecture. This assignment comes in three parts: a list of "pre-class learning goals" that students should achieve before the next lecture, a list of readings that students can use to achieve those goals, and suggested exercises to assess their progress. In addition, before the next lecture, students are required to complete an online, low-credit quiz that assesses the pre-class learning goals. Ideally, these quizzes would be drawn randomly from an immense bank of questions all targeted at the relevant learning goals. For now, the quizzes are fixed. As a result, rather than resubmitting the quiz again and again with instant feedback, students are instead encouraged to look over the quiz as they begin their preparation (to focus their reading) and complete it at their leisure. Feedback comes only after the deadline, however. Beyond the automatically graded questions targeting pre-class learning goals, each quiz also includes 1-2 open-ended questions that get students thinking about the next lecture's in-class learning goals. These are graded purely for completeness.

Just before the lecture (i.e. "just in time"), the instructor aggregates the results on the automatically graded questions and summarizes results on a sample of the open-ended questions. Where students did poorly on the automatically graded questions, the instructor dynamically introduces new discussion and problems to reinforce those concepts. The instructor usually summarizes students' responses on the open-ended question into a strategy guide that the class discusses before diving into closely related problems for in-class work.

In-class work is weighted heavily toward working and discussing practical problems, often with (anonymous, ungraded) clicker questions used to gauge students' progress and summarize their results. In the larger section (with ~90 students) two undergraduate TAs attend lecture to help the instructor facilitate problem solving sessions.

As an example, the pre-class learning goals for the first lecture on predicate logic are stated as follows:

By the start of class, the students are expected to be able to:

- Evaluate the truth of predicates applied to particular values.
- Show predicate logic statements are true by enumerating examples (i.e., all examples in the domain for a universal or one for an existential).
- Show predicate logic statements are false by enumerating counterexamples (i.e., one counterexample for universals or all in the domain for existentials).
- Translate between statements in formal predicate logic notation and equivalent statements in closely matching informal language (i.e. informal statements with clear and explicitly stated quantifiers).

The online quiz asked 11 questions such as, "What can you say about the truth of the following statement: $\exists x \in Z, \exists y \in Z, \exists z \in Z, x^2 + y^2 = z^2$?" Students averaged over 80% on all but three questions; so, the instructor reviewed those three questions to begin the class.

The in-class learning goal was for students to be able to build statements about the relationships between properties of various objects using predicate logic. Problems may be drawn from real-world examples like "every candidate got votes from at least two people in every province" or computing examples like "on the ith repetition of this algorithm, the variable min contains the smallest element in the list between element 0 and element i". These will then segue to open-ended predicate logic representation problems, including one asked on the quiz (defining precisely what it means for a list to be sorted).

Since students already understood the basic terminology, as indicated by their quiz results, lecture focused on higher-level topics, including summarization and further development of the open-ended quiz question to precisely define what it means for a list to be sorted. Lecture culminated with students defining a predicate GenerallyFaster(a1, a2) that indicates when one algorithm is "generally faster" than another based on a predicate Faster(a1, a2, n) that compares the algorithms for particular problem sizes. Students invented several of their own definitions, including one that corresponds to the standard comparison of asymptotic performance for algorithms, and we discussed the strengths and weaknesses of each one.

Anecdotally, the instructor feels he has a clearer sense of when students are able to apply core course concepts successfully. In particular, rather than only discovering that students cannot solve real problems when they fail at assignments or exams, he can now discover and address students' difficulties early in the learning process. An anonymous clicker survey shows that students generally now see the relevance of labs to the course material, possibly due to the increased connection between hands-on problems and theory in lecture.

Finally, assessment results (e.g., the midterm) indicate that both sections that use a problem-based style in class are performing at least as well as the section using a more traditional lecture approach.

3. Problem-Based Peer Learning

In our Data Structures and Algorithms course (CPSC 221), we are faced with a wide range of topics that require students to transition smoothly between abstraction and implementation. This relationship can be difficult for students to understand without high-level problem solving practice paired with implementation practice. Traditionally, CPSC 221 has seen implementation techniques taught during labs in the form of an algorithm and a partially-implemented framework, which students complete under the guidance of a lab assistant. Higher level abstract concepts are then presented in a more traditional, lecture-style environment. Repeatedly, however, interactions with students (both exam-based and in-person) have shown a distinct lack of comprehension regarding the nature of abstraction versus implementation. In particular, students make inappropriate choices when deciding to address a problem at an abstract versus implementational level.

To address this deficiency, during our two-section January to April 2009 offering of CPSC 221, an experiment was conducted to test the efficacy of a non-standard lecture model. In our experiment, one section of the spring term was offered in the traditional format where classroom contact hours were spent presenting electronic slides (made available prior to the class), and clicker-style questions to help students review the material and assess their progress. The second section was delivered using an alternate model of presentation presenting new course concepts in two components: an "offline" component where students were expected to read and learn the prescribed concepts before class; and an "introduction via problem" component where students were expected to solve a moderately sized problem, or series of problems, in groups during class.

The offline-learning component was sometimes supplemented with a clicker-style quiz or short discussion administered at the start of the class to ensure students have done their preparatory readings and to identify any areas of confusion among the students (not unlike the JITT model described for the CPSC 121 above). After a brief discussion, the remaining class time was spent on the "introduction via problem" component using group exercises, which required the students to directly apply the concepts they were learning while the instructor monitored the progress of each group, offering hints and discussion points when appropriate. Some of the exercises were presented in the form of a series of tasks that progressively led the students from newly acquired foundations toward more in-depth applications, or a large-scale problem in which the students had to determine and apply the relevant concepts to the solution.

It should be noted that in the "introduction-via-problem" component, core concepts were introduced "cold", that was without any prerequisite reading and a minimal introduction by the instructor. The students relied on their problem-solving skills and prior knowledge to solve the problem. As an example, students were given two unidentified sorting algorithms at the start of the complexity unit for which they were to answer the questions, “What do these algorithms do?” and “Which algorithm is better?” The ensuing group discussion served as a segue into complexity theory and how and why we wanted to compare algorithms.

Aside from different presentation styles, both sections shared the same instructor, course material, labs, assignments, exams, and learning goals. In both sections the course material was presented via three-hours of classroom-contact time, and two hours of lab time. Furthermore, both sections used learning goals as concept “book ends” presented at the start and close of each unit.

As of this writing, the final exam has not been administered yet. The final exam is composed of a question derived from a shared exercise that was run in both sections, and two questions for which the material explicitly appeared on an experimental exercise. Each of these questions had high-level and low-level sub-questions to test performance between the two sections given the differing delivery of content. Any significant differences between the sections will result in the lower section being scaled up to meet the higher section.

An end-of-term informal survey has now been completed in the experimental section, indicating a highly favourable response to the experimental classroom setting. While not conclusive, this nonetheless motivates this approach from the perspective of student engagement and enjoyment. And although the midterms have thus far been statistically insignificant in their differences (though the average was higher in the section delivered using the alternate learning model), the true efficacy of this approach is more likely to be felt in long-term retention. This effectiveness may already start to appear in the final exam as students are forced to revisit the material from the first half of the term. Plans are underway for a follow-up study in approximately five months to test retention between both the experimental and control sections.

The final outcome of this experiment will be analyzed on the basis of assessment outcomes between the two sections. Although the first midterm did not report a statistically significant result between the sections, the instructor reports that students did respond favourably to the learning environment, and there has been no push-back on the front-loading of course material. In addition, there are fewer students during office hours from the alternate learning model section.

4. BRACElet

BRACElet [1] is a multi-institutional investigation into the reading and comprehension skills of novice programmers. The basic idea is that as a student or beginning programmer transitions from novice to expert, their explanations of what a piece of code does also change.

For example when experts are presented with the following code sample:

```
bool bValid = true;
for (int i = 0; i < iMAX - 1; i++) {
  if (iNumbers[i] > iNumbers[i+1]) {
    bValid = false;
  }
}
```

and asked to explain what it does in plain English experts typically say the code checks to see if the numbers in an array are sorted [2]. Non-experts will typically engage in a line-by-line description of what the code does without relating what the body of code as a whole does.

Based on this idea we wanted to explore how student explanations changed as they progressed through our various first and second year programming courses. Assuming these questions, in some dimension, express a student’s mastery of the concepts and techniques of a particular course, then these questions may serve as a baseline to measure the success, or lack thereof, of curriculum changes. In addition to tracking the progression of students from novice to expert, we also wanted to see if there was any relationship between a student’s performance on these questions and their final course grade. As the first step in this study, we administered several "explain in plain English" questions to students at various levels of computing at UBC. Our initial data was gathered from:

1. Students who had just completed our primary introductory course in Computer Science (about 420 students).
2. Non computer science students who had just completed a second programming course, but whose first programming course was not as comprehensive as the course in (1) (about 140 students)
3. Students just starting the course in (2) (about 210 students)

In each case we provided our students with the same three code samples expressed in the primary programming language of that course. The samples were presented in order of increasing complexity. The first code sample swapped the value of two variables, the second computed the average of a collection of numbers in an array, and the third looked for the last occurrence of a character in a string. The first two questions were graded out of two. One mark was given for answers that provided a line-by-line description of the code with two marks awarded for answers that described what the code as a whole did. No half marks were awarded. Full marks were given for answers that weren’t quite correct but were trying to explain what the code as a whole did as opposed to providing a line-by-line summary. The third question was graded out of three, with, again, one mark for a line-by-line description, two marks for a description of the code as a whole, and a third mark for indicating that the code was looking for the last occurrence of a character[1]. The questions were administered as part of the final exam for the first two student populations and as an anonymous quiz on the first day of class for the third group.

Although we have collected a substantial amount of data across the three identified student populations, the initial analysis has

[1] Students often realized the code was looking for a particular character but failed to realize it was the last occurrence being looked for.

focused on the students who had just completed their second programming course and whether or not these questions were good predictors of student course grade. Our hope was that the scores on the three "explain in plain English" (i.e. BRACElet) questions would either collectively, or individually, exhibit a strong correlation with either or both the final exam and/or course grade.

The approach we are using is to compute the correlation between the BRACElet question scores and the final grades in the course.

The data corresponding to questions B1, B2, B3 in Table 1 shows the results of this computation. The correlation for the three BRACElet questions ranges from 0.35 to 0.47. In addition the correlation of the sum of the three questions, 0.56, is provided. Although these questions all exhibit a positive correlation with the course grade they are only moderate predictors.

Since the BRACElet questions were not strong predictors (with correlation of 0.7 and greater) of a student's performance in this course we decided to expand our analysis to see if other questions on the final exam were better predictors. The rest of Table 1 lists all the questions on the final exam, their correlation to the student's final grade, and the number of marks the question was out of.

Question	Correlation to Final Grade	Maximum Score	Average Mark
Q1	0.64	10	7.16
Q2	0.7	20	10.52
B1	0.45	2	1.64
B2	0.35	2	1.76
B3	0.47	3	1.8
Q4	0.66	4	2.72
Q5	0.37	4	3.18
Q6	0.56	17	9.27
Q7	0.67	7	3.8
Q8	0.47	6	3.24
Q9	0.29	7	4.17
Q10	0.7	12	8.43
Q11	0.49	12	10.54
Q12	0.62	9	5.8
Q13	0.79	20	9.94
Sum of BRACElet Questions	0.56	7	5.2

(B1, B2, B3 are BRACElet questions.)

Table 1: Correlation between Questions on Final and Final Grade

4.1 Analysis of the Data

Our goal in looking at this data is to try to get a better understanding of the sorts of questions that correlate highly with a student's grade and the underlying skills required to be successful at these sorts of questions. The hope would be that if we can identify these skills we can then provide guidance to student's on how to improve these skills and hence their grade in the course. Similarly, if we find questions with a low correlation they need to be examined to see if there is something fundamentally wrong with the question, be it in the question's wording or in what we are testing for. In both scenarios the documented set of learning goals, which are given to students, are also used as part of the analysis. Students are also told that exam questions will always be mappable to one or more learning goals. In other words, the exam is in some sense a checklist of the level of knowledge and understanding of a subset of the course's learning goals. (There are usually too many learning goals to test all of them on a final exam.)

To illustrate the use of this data consider two scenarios:

1. Questions that have a high correlation to a student's final grade.
2. Questions that have negative or little correlation to the final grade.

Given these criteria three questions with a high correlation to the final grade (Q13, Q2, and Q10) and one with a low correlation (Q9) to the final grade are of particular interest. (The course instructor was pleased to see that no questions had a negative correlation.)

Question 13 had the highest correlation at 0.79. This question leads students through the process of developing a piece of code to add a new node to a linked list. The students are asked to write comments about what some code does, draw pictures of how the structure of the list changes, write a five-line function to add a new node, analyze the time complexity of the code they wrote, and finally compare this implementation to one that uses a different type of linked list. In essence this question touches on multiple aspects of the course material and different parts of the question require different levels of understanding of the material. It also has the property that later parts of the question can be completed without getting earlier parts correct.

In question 10, students are given snippets of C++ code and asked to label a picture of program memory. For example students might be expected to indicate where variable X is located and what value X has. If a pointer is involved, students would be expected to indicate the memory location the pointer points to. In total there were 6 snippets of code of varying degrees of complexity with respect to the resulting memory layout. Again like question 13 this question touches on multiple aspects of the course material. Some diagrams are relatively simple while others are fairly complex. To get full marks students would need to have a good understanding of the relationship between C++ code and how memory is used.

Finally, question 2 also shows strong correlation. Question 2 was comprised of a series of sub-questions requiring at most a short sentence to answer. The questions cover the breadth of the course material and typically focus on a key piece of information about a particular topic. An example of this would be a question that asks under what conditions would Quicksort exhibit the same running time complexity as a simple bubble sort.

What do these three questions have in common? Not surprisingly they are all multi-part questions. In addition the sub-questions are not tightly tied to each other. That is, the inability to do one sub-question does not preclude being able to answer other sub-questions. Since both questions 10 and 13 explore one area at several levels of detail, one might refer to this as a breadth of understanding about a topic area. Question 2 has this same property of breadth but from the perspective of key ideas throughout the course. This observation raises the question as to the role of breadth play in a student's success and whether the ability to exhibit breadth of understanding in one area implies a breadth of understanding in another area? We might be able to gain some insights into this by looking for correlation between questions. For example is there a strong correlation between these three questions? We have not looked into this in detail, but our preliminary analysis suggests there is not. Taking this further, we also intend to take subsets of questions and explore the correlation between subsets of questions and final grades.

As indicated earlier, another area of interest is questions that have a low correlation to the course grade. From this exam, question 9 is a candidate. This question focuses on merge sort and asks a number of questions about how the sort works. Although this is a multipart question it has the property that if a student doesn't understand one part of the question it is likely that other parts of the question can't be completed. In some sense one can say that the question lacks breadth. Also, given the relatively low average mark for this question it is important to check the learning goals to ensure that the question is indeed testing students at the appropriate level for this material. In this case they were. Armed with this information the course instructor has a number of options:

1. Re-work the question so that it tests for a broader understanding of the topic.
2. Change the course learning goals to better reflect the expected learning outcomes and revise the question as necessary.
3. Change or augment the way the material is taught.
4. Do nothing and treat this as a question that only the top students in the class are expected to get.

Note that having a low correlation to a final grade is not necessarily a bad thing. For example Q5 has a low correlation (.37) but the average is relatively high (3.18/4). In examining the question and the learning goals it was determined that this was an important question and the material was not tested anywhere else. It just happened that most students did quite well on this question, even if they didn't have a good course grade.

Going forward, the questions with high correlation to final course grade (Q2, Q10, and Q13), or very similar ones, will be administered on the final exam for this course in April. From this we hope to see if these questions continue to be strong predictors given the difference student backgrounds in this offering of the course. Although the BRACElet questions were not strong predictors of a student's performance in this course, the questions provide useful metric for ranking students across the different courses. As a result, we will again be administering the BRACElet questions and we will also have the opportunity to compare the class's results with the same questions administered at the start of term.

As we gather and analyze more of the data we will be able to make changes to our teaching and course delivery methods in an effort to improve outcomes. The work we have described here will provide a way to measure our progress. In addition, we can also provide guidance to our students with respect to the things they can do to aid their learning. For example in this course we exhort our students to always draw pictures of data structures and their in-memory representations. Based on the results of our analysis (Q10) we can tell students in this class that the ability to draw out memory representations seems to be an important skill for course success. We know that students who can't draw memory representations do poorly. Unfortunately we don't know if drawing memory pictures causes students to have a better grasp of the course material or if it is the case that a better understanding of the material makes picture drawing easier. Knowing the answer to this and other similar questions would allow us to tell students how to be successful in this course and we hope to perform additional studies to answer these sorts of questions.

5. Attitudinal Surveys

Attitudinal surveys provide valuable information on the student perception of the course and their learning. This includes their perception of the level of difficulty of the course, their progress in the course, the way the material has been presented, the forms of assessment, their expectations, their self-efficacy, etc. While attitudes about the course can change in a very short time, attitudes about the students learning and expectations can provide useful insights to the instructors. We have conducted attitudinal surveys of students in each of the four levels of the CS program at UBC. Some of the questions asked include:

- What are your three biggest expectations from this course?
- How do you study for this course?
- How have you been using the learning goals for this course?
- Which of the following do you feel are most important to you when you work on a particular difficult assignment:
    - Regular meetings with TA / Prof to discuss about the assignment.
    - Bouncing off ideas with friends.
    - Sample code examples.
    - Regular feedback on progress.
    - Access to web resources.
    - Lecture / Assignment material.
    - Advice from students who have completed the course.
    - Help in the lab while working on the assignment.
    - Drop-in help at TA office hours.
- How do you know when you have learned something?

Responses from these questions allow instructors to have greater insights in the corresponding areas:

- Whether students are intrinsically (e.g. want to learn something) or extrinsically (e.g. taking a course primarily to satisfy program requirement) motivated in this course.
- Whether the students are spending appropriate and reasonable amount of time for this course.
- Whether the learning goals are well defined to guide the students in their learning.

- Whether appropriate support is made available to the students in their learning.
- Whether the students are developing appropriate meta-cognitive skills.

As we have developed learning goals for a number of courses in our program, we were interested in finding out whether the students are using these learning goals in their study, and if so, whether this has any correlation with their course grades. We were also interested whether students who are intrinsically motivated score higher in midterms / final than those who are extrinsically motivated. In both cases, there is a slight correlation. Students, however, are not usually good predictors of their own performance. When asked what final grade they expected to get, almost none indicated that they would fail the course, although, many of them had failed the first midterm. Also, when students were asked how they studied for their course, most answered that by reading the textbook or notes. This may not be the most effective ways in studying, especially for long term retention, where research shows that repeat testing is more effective [4].

While attitudinal surveys provide valuable insights on student attitudes and perceptions in a number of areas, insights into their unique background and circumstances can be gained through follow-up one-on-one interviews. These interviews are best conducted by an education researcher rather than by the instructor so that the students may freely express their opinions and point of views without concern of any impact on their grades. One of the questions that can provide particular insights for the instructors is how the students approach each question on the midterm or exam. The responses may reveal what strategies or the sources of material they use and this may provide insights for the instructors on what components of the course the students deemed to be useful for their learning. As an example, in one of the courses, students find that the labs are helpful but not so much for the assignments. A follow up investigation reveals that the assignments are too big and are worth too little towards the final course grade. In another course, clicker questions are found to be helpful in particular types of questions on the midterm. Such information is useful for the instructors for future offerings of the same course.

6. Conclusion and Future Work

In this paper, we have described a number of education research activities in UBC CS courses to improve student engagement and learning. We have collected student performance and attitudinal data in these courses and made some initial conclusions. We plan to correlate this data with similar data to be collected in the future terms for a longtitudinal study, especially in the area of long term retention. We will also correlate the data with students' prior education background and experience to ensure students success in our CS program.

7. REFERENCES

[1] BRACElet. 2006. BRACElet. Retrieved on February 19, 2009 from http://online.aut.ac.nz/Bracelet/repository2.nsf/HomePage?OpenPage.

[2] Lister, R., Simon, B., Thompson, E., Whalley, J., 2006. Novice Programmers and the SOLO Taxonomy. 11th Conference on Innovation and Technology in Computer Science Education (ITiCSE).

[3] Simon, B. 2008. Computer Science Learning Goals. Carl Wieman Science Education Initiative at the University of Britist Columbia. Retrieved on February 19, 2009 from http://www.cwsei.ubc.ca/departments/computer_learning_goals.htm.

[4] Roediger, H.. Karpicke, J. 2006. Test-Enhanced Learning, Taking Memory Tests Improves Long-Term Retention. *Psychological Science*. 17(3), pp249 – 255

[5] Novak, G., Gavrin, A., Wolfgang, C., Patterson, E. 1999. *Just-in-Time Teaching: Blending Active Learning with Web Technology*. Upper Saddle river, NJ: Prentice Hall.

Making University Education more like Middle School Computer Club: Facilitating the Flow of Inspiration

Alexander Repenning
University of Colorado Boulder
Department of Computer Science
Boulder, CO 80303
(303) 492-1349, 001
ralex@cs.colorado.edu

Ashok Basawapatna
University of Colorado Boulder
Department of Computer Science
Boulder, CO 80303
(720) 838-5838, 001
basawapa@colorado.edu

Kyu Han Koh
University of Colorado Boulder
Department of Computer Science
Boulder, CO 80303
(303) 495-0357, 001
kyu.koh@colorado.edu

ABSTRACT

The way programming is currently taught at the University level provides little incentive and tends to discourage student peer-to-peer interaction. These practices effectively stifle any notion of a 'learning community' developing among students enrolled in university level programming classes. This approach to programming education stands in stark contrast to the 'middle school computer club' approach; As part of 10 years of research projects aiming to teach programming to middle school children, it is observed that middle school students in computer clubs freely share programming ideas, code, and often query one another and provide solutions to the various programming problems encountered. To enable these interactions at the university level, a novel online infrastructure has been developed over the past 6 years through use in the Educational Game Design Class at the University of Colorado Boulder. The culmination of the submission system, entitled the Scalable Game Design Arcade (SGDA), seems to foster the flow of ideas among students yielding an effective open classroom approach to programming education.

Categories and Subject Descriptors

K.3.2 Computer and Information Science Education

General Terms

Algorithms, Design, Human Factors,

Keywords

University Programming Education, Middle School Programming Education, Scalable Game Design, Open Classroom, peer-to-peer interaction, flow of inspiration, Computational Thinking.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

1. INTRODUCTION

The current structure in most computer science classes at the university level resembles the so-called "Sage on the Stage" approach to learning [1]. A single lecturer in the front of the class talks to a group of students who are taking notes. Currently, the emergence of remote and on-demand class viewing obviates the need to physically be in a class wherein a teacher takes this 'Sage on the Stage' approach to teaching. This approach is further indicted by a recent study suggesting that students who on-demand remotely view these types of lectures actually retain more information and get a deeper understanding of the material as compared to students who physically attend the class [8]. Thus, the value of physically attending class is decreased when a teacher takes this approach. Moreover, the 'Sage on the Stage' approach to teaching makes no attempt to use the inherent characteristics of a student-filled classroom to enable a better learning experience. One could argue that with this teaching approach, a lecture wherein one student attends would be identical to a lecture wherein 50 students attend. Given that this is the case, it begs the question, why even have a physical classroom environment? This line of reasoning is unfortunate since the physical student-filled classroom environment lends itself nicely to teaching strategies that foster peer-to-peer student learning and the creation of a learning community within the classroom. Unfortunately, most undergraduate computer science classes make no attempt to create any kind of learning community.

In computer science classes, for example, save for a possible "computer lab", assignments are completed outside of class with little or no motivation or incentive for peer-to-peer interaction [2]. Furthermore, collaborations, sharing of ideas, and looking at fellow students' code is actually frowned upon and often considered cheating; for example, it is not uncommon for teachers to run automatic checks on assignments to ensure everyone's code is strictly their own. This has the unfortunate side effect of inhibiting the proliferation of ideas among students drastically reducing the opportunity for students to learn from one another. Since students are actively discouraged from looking at fellow classmates' work, they are not pushed to do more based on what their peers have accomplished on a given assignment [3]. This has a negative effect not only on individual student achievement, but also, reduces the level of work produced by the class as a whole over the duration of a given course [4]. Some classes try to allow for peer-to-peer interactions through group projects. However, in our experience and as documented by others, group projects often do not yield true group collaborations as students divide the work up and only work together to reassemble the project when everybody's individual part is complete [7]. Thus, most group projects only have limited peer-to-peer interaction and learning. The above issues motivate the following question: what changes can be made such that students effectively learn from and inspire one another in undergraduate computer science classes?

Surprisingly enough, an interesting solution approach to this problem presents itself in the seemingly chaotic structure of middle school computer clubs. For the last 10 years, as part of several NSF funded projects, middle school students were taught the fundamentals of programming using a rapid game prototyping environment called AgentSheets [5]. In our most recent project called Scalable Game Design[1], we are exploring computer science education through game design starting at middle school and moving along all the way to graduate school. In this context we have found that there are interesting educational phenomena taking place at the middle school level that are perhaps worth exploring for more advanced levels [6].

A typical AgentSheets lesson for a given day involves students in front of a computer creating a game. However, because of the relaxed non-classroom atmosphere of an after-school club, students are able to run around to the computers of their peers. Often, upon a student viewing a fellow classmate's game, the following sequence takes place. First, the student notices the classmate's interesting idea or concept and asks that classmate "How did you do that?" The classmate then explains to the student how to accomplish the concept even displaying the "code" used to implement the idea. Social scaffolding, wherein more advanced kids help others, happens naturally in this environment. The two students converse providing feedback on the idea; then, the wandering student would go back to the computer integrating and updating the new code for her/his own purposes. The ability to traverse the class and openly share concepts between classmates enables a network capable of carrying inspiration and influence from one student to the next. All of this appears to work with little, if any, teacher input.

Moreover, this classroom infrastructure allows for viral ideas, wherein one student discovers a concept at first, and that concept soon spreads throughout the class [6]. This flow of inspiration is not limited to simple ideas; A highly sophisticated example of this flow appeared when a student was explained the concept of a collaborative diffusion approach to help him give agents in his game better AI. Though the equations associated with diffusion involve more complex math than is learned at this student's grade level, by the next time the group met, one week later, a group of students had started using diffusion equations in their own games and were explaining the concepts to others.

These middle school experiences motivate the implementation of a university level programming class that allows for all the same communication modes among students. Over the past 6 years an online cyber-infrastructure, through use in the Educational Game Design Class at the University of Colorado Boulder, has evolved to allow for these interactions. The culmination of this evolution is the newly developed Scalable Game Design Arcade (SGDA) and, thus far, the SGDA seems to have successfully allowed for many of the same types of interactions visible in middle school computer clubs.

[1] scalablegamedesign.cs.colorado.edu

Section 2 explains the Flow of Inspiration Principles derived from the middle school computer club experiences. Section 3 explains previous approaches to this problem finally arriving at the SGDA implementation. Section 4 provides initial findings of the SGDA, along with a brief discussion, including the results of an in-class questionnaire used to gauge the extent to which the SGDA enables student peer-to-peer learning.

2. FLOW OF INSPIRATION PRINCIPLES

In the limited time, infrastructure and physical space allotted for university programming classes, to integrate the benefits of middle school computer club learning, a different approach must be taken. Thus, the methods employed in the Educational Game Design Class involve transferring the interactions present in the computer club environment to one that works within the limitations of the classroom. Specifically, a successful implementation of this environment should support the following five Flow of Inspiration Principles.

Flow Of Inspiration Principles

These Principles should allow students to:

1) Display projects in a public forum

2) View and run fellow students' projects

3) Provide feedback on fellow students' projects

4) Download and view code for any project

5) Provide motivation for students to view, download, and give feedback on fellow classmates' projects

The first four specifications describe general infrastructure characteristics that enable in-class peer-to-peer interactions. The final point, wherein the class provides incentive for these interactions, is crucial for attaining the emerging behaviors present in the middle school computer club environment. This incentive is provided through an initial highly-scaffolded curriculum that gradually gets relaxed as the semester proceeds.

2.1 Educational Game Design Class

Each week, the Educational Game Design Class consists of a theoretical and practical part. The theoretical part discusses gamelet creation strategies including adding educational value to games, making games engaging, and reviewing computational thinking patterns [12,13]. The practical part allows students to apply this knowledge by creating gamelets. The Educational Game Design Class has an aggressive schedule assigning students to create one gamelet a week for 8 weeks. The remaining semester time is spent on a final project. The first 4 weeklong assignments are the same among all students. The next 4 weeklong assignments comprise a period known as "Gamelet Madness." Gamelet Madness forces students to think of and implement their own original simple educational game idea each week. Furthermore, a given student must create a gamelet that has no relation to a prior week's Gamelet Madness game she/he created. Generally, during Gamelet Madness, the students' submissions for a given week have very little relation to one-another. The final project involves students creating an educational gamelet and playtesting it at the local middle schools for feedback. Students then use this feedback to improve their gamelet and present their findings to the class. With approximately 32 students in the class creating 9 games over the course of the semester, the Educational Game Design Class as a whole yields around 290 games by semester's end.

As mentioned above, the first four homework assignments of the Educational Game Design Class are weeklong wherein all students create the same games: Frogger, Sokoban, Centipede and The Sims. These games start simple but get increasingly complex as students learn more sophisticated computational thinking patterns [6]. Though individually-created student games may look different, the main programming patterns used to solve various in-game problems are the same. Thus, in Frogger, for example, if a student wants to program the truck colliding with the frog, the implementation structure is virtually identical among all students in the class. This situation provides the motivation to look at fellow students' projects to see how they went about solving a particular problem. Furthermore, since everyone is working on the same project, students want to look at other projects to compare with their own assignment and integrate interesting things fellow classmates may have done into their own project. As the class progresses, this scaffolding is removed as students work on the more individualized assignments of Gamelet Madness and the final project; By this point, the precedent of viewing other students' projects and using their interesting ideas has hopefully been established.

3. EVOLUTION OF CYBERLEARNING INFRASTRUCTURE

One possible way to replicate the Flow of Inspiration witnessed at middle school computer clubs within a university classroom is to employ a cyberlearning infrastructure. But what is Cyberlearning? In the recent "Fostering Learning in the Networked World: The Cyberlearning Opportunity and Challenge. A 21st Century Agenda for the National Science Foundation, June 2008" report [10] the National Science Foundation defines Cyberlearning to be

> "[...] networked computing and communications technologies to support learning. Cyberlearning has the potential to transform education throughout a lifetime, enabling customized interaction with diverse learning materials on any topic—from anthropology to biochemistry to civil engineering to zoology. Learning does not stop with K–12 or higher education; cyberlearning supports continuous education at any age."

Cyberlearning infrastructures take many forms. In the classroom cyberlearning infrastructures, like Poogle for example, have been used to add social elements and publicly share solutions to homework assignments [11]. Google Code Search[2], on the other hand, is a cyberlearning infrastructure that helps developers seek out specific programming code for use in their projects. For our purposes, the cyberlearning infrastructure should enable the middle school computer club interactions. The evolution of a cyberlearning infrastructure that accomplishes the Flow of Inspiration Principles is described in the systems that follow. The shortcomings of prior systems with respect to the Flow of Inspiration Principles motivate the creation of the Scalable Game Design Arcade.

[2] http://www.google.com/codesearch

3.1 Individual Homework Email Submissions

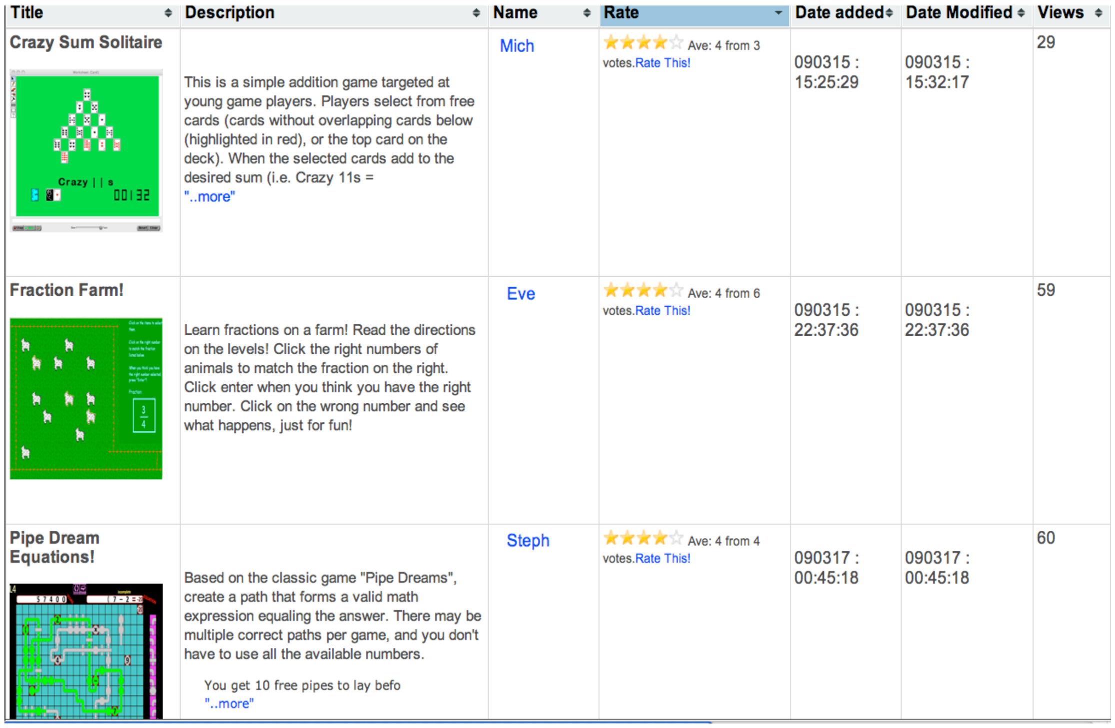
Initially, in 2003, before attempting to recreate the Flow of Inspiration Principles, individual students in the Educational Game Design Class would email the grader their assignments. This is the classic structure of most university level programming classes; none of the Inspiration Principles are met. The only project feedback is given by the class grader who probably only reviews each assignment briefly. Students do not view one another's projects and cannot see how other students went about solving problems. Nor can students give feedback on a fellow classmate's project. One advantage to this system is that cheating is more easily discovered as similarities between student-code should not occur if students are not viewing each other's code or garnering illegal input from fellow students.

3.2 Group Projects

Allowing for increased student peer-to-peer interactions necessitate a shift from the email submission system. To this end, in 2004 and 2005, the Educational Game Design Class employed group projects as a way to encourage student interaction. It was thought that group projects would foster all the specifications above and create an open classroom environment among all the group members. However, this did not occur; instead, students would break up projects and delegate tasks. Interaction, for the most part, was restricted to when the group members met back up to reassemble the project. Within this divide and conquer approach the student goals are disparate; thus, there is very little advantage in peer interaction. In short, the group project system was found to be a poor peer learning model because there exists little intrinsic motivation to learn from other students. From this experience it became clear that to get the emergent interaction advantages inherent in the middle school computer club, students must be working on individual projects but with facilitated and encouraged community interaction. One way to accomplish this within the limitations of a university class involves creating a cyber-infrastructure.

3.3 Gallery Organizer Repository of Projects

The Gallery Organizer Repository of Projects (GORP, 2006) infrastructure used an online project posting submission system; this system allowed students of the Educational Game Design Class to put their projects online and freely view fellow student projects [9]. Furthermore, students could comment on and easily download other students' projects. This cyber-infrastructure obtained limited success in each of the above Principles laid out. Students in theory could display, view, provide feedback and download other students' projects. However, students could not rate projects in a simple manner. Furthermore, students in general would not leave comment feedback upon viewing a project; it was discovered that the main reason for this was that the overhead of leaving a simple comment, logging in, meant that the student had to go through more trouble than it was worth yielding the practice nonexistent for more casual comments. There was also existed no method of tracking who downloaded and viewed a project so there was no way to evaluate how well or even if the peer-to-peer interaction was taking place, though there was informal evidence to suggest that it was. The shortcomings of GORP motivated the current cyber-infrastructure, the SGDA.



Title	Description	Name	Rate	Date added	Date Modified	Views
Crazy Sum Solitaire	This is a simple addition game targeted at young game players. Players select from free cards (cards without overlapping cards below (highlighted in red), or the top card on the deck). When the selected cards add to the desired sum (i.e. Crazy 11s = "..more"	Mich	★★★★☆ Ave: 4 from 3 votes.Rate This!	090315 : 15:25:29	090315 : 15:32:17	29
Fraction Farm!	Learn fractions on a farm! Read the directions on the levels! Click the right numbers of animals to match the fraction on the right. Click enter when you think you have the right number. Click on the wrong number and see what happens, just for fun!	Eve	★★★★☆ Ave: 4 from 6 votes.Rate This!	090315 : 22:37:36	090315 : 22:37:36	59
Pipe Dream Equations!	Based on the classic game "Pipe Dreams", create a path that forms a valid math expression equaling the answer. There may be multiple correct paths per game, and you don't have to use all the available numbers. You get 10 free pipes to lay befo "..more"	Steph	★★★★☆ Ave: 4 from 4 votes.Rate This!	090317 : 00:45:18	090317 : 00:45:18	60

Figure 1: Scalable Game Design Arcade with thumbnail, author description, rating, and submission time. Clicking on the Author's name link for any of these projects allows one to play the game, download code, rate, and add comments.

3.4 Scalable Game Design Arcade

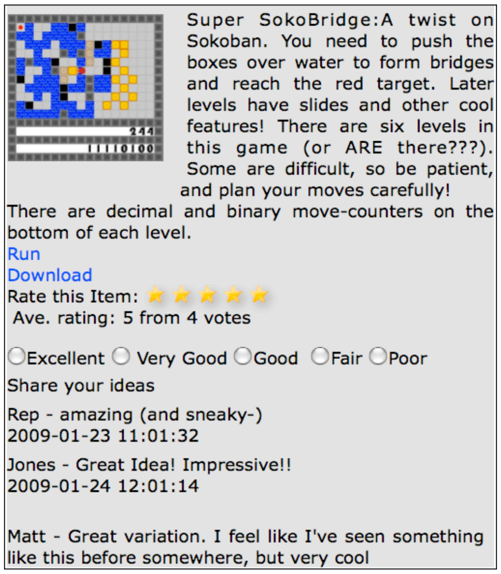
The Scalable Game Design Arcade (2009) extends the GORP online submission interface to facilitate user feedback. The user does not have to log in to leave feedback on given student's submission. Furthermore, a one to five rating system provides an easy way to appraise projects and organize by projects that got the best user feedback. This, in theory, makes it easy for students to look and be inspired by the best projects and allows a simple way to provide feedback. Embedded in the rating system and the ability to organize by ratings is the Youtube/Facebook mentality which allows students to gravitate towards classmate's work who they individually find interesting as well as work that the class as a whole finds interesting. Furthermore, just like with Youtube, students can sort according to many different criteria including views, star rating, and name. Students are encouraged to submit their work early and often so that the feedback received can be used in the creation of subsequent game iterations. The user does, however, have to login to download a program enabling the tracked flow of ideas from one user's game to another. It also discourages cheating as directly copying a fellow student's work can easily be detected.

4. RESULTS

Figure 1 depicts a screenshot of the Scalable Game Design Arcade. The SGDA Website consists of a list of game descriptions including a thumbnail depicting the game, descriptive text describing the game along with some additional meta-information such as number of views. On the left of Figure 1 is the "Title" column which contains a thumbnail screenshot of each project along with a title. Next to the thumbnail, in the "Description" column, is a brief synopsis of the game, game storyline, and game-play instructions written by the student. To the right of the description, in the "Rate" column, is the ratings fellow students give the game out of 5 stars along with how many students star rated the game; in the column to the right of the rating, is the "Date Added" column which shows the time the project was first submitted. The next column to the right, the "Date Modified" column, displays the time of the most recent game submission. Finally, the "Views" column furthest to the right displays the number of times the project has been viewed by people.

If a student clicks on the Author's name link in the 'Name' column, it allows the student to run the game, leave a comment, and download the code; Figure 2 depicts this. The projects are run as a java applet in the browser itself. However, as mentioned above, to download another student's code, the student must first log in.

In terms of accomplishing the five Flow Of Inspiration Principles, initial indications from students using the SGDA are promising. In order to get a better understanding of how students use the SGDA and to what extent, if any, the SGDA enables the Flow of Inspiration Principles, a questionnaire was given to the class. The following sections will analyze and discuss the results of this questionnaire.



Super SokoBridge:A twist on Sokoban. You need to push the boxes over water to form bridges and reach the red target. Later levels have slides and other cool features! There are six levels in this game (or ARE there???). Some are difficult, so be patient, and plan your moves carefully!
There are decimal and binary move-counters on the bottom of each level.

Run

Download

Rate this Item:

Ave. rating: 5 from 4 votes

Excellent Very Good Good Fair Poor

Share your ideas

Rep - amazing (and sneaky-)
2009-01-23 11:01:32

Jones - Great Idea! Impressive!!
2009-01-24 12:01:14

Matt - Great variation. I feel like I've seen something like this before somewhere, but very cool

Figure 2: A detailed game description window wherein a student can run the project, download the project code, star rate, and comment on the project.

4.1 Student Questionnaire

The online questionnaire administered to the class consisted of 23 questions, 15 open-ended and 7 multiple choice. Replies were anonymous, voluntary, and students could skip any question they did not want to answer. Responses from 20 students were collected, and the open-ended answers were categorized to discover any trends among students. On all open-ended questions students could give multiple answers, thus, the percentages of all categorized answers to open-ended questions can exceed 100%.

The questionnaire is meant to answer two questions. The first question is to what extent does the SGDA accomplish point five of the Flow of Inspiration Principles; namely, given that students can view, play, download and comment on classmates' projects, how often does this online peer-to-peer interaction actually occur. The second question investigates the following: to the extent that the Flow of Inspiration Principles are being integrated into the class, is there indications that the quality of work is increasing? It should be noted that we are not proving a causal relationship between the quality of work and the Flow of Inspiration Principles based on the questionnaire; however, we are trying to establish if there is an attempt on the student's part to improve their work because of the SGDA. The following result sections are organized in terms of Flow of Inspiration Principles 2-4 in order to see how well the SGDA is accomplishing motivating these principles (essentially how well the SGDA is accomplishing Flow of Inspiration Principle 5; it should be noted that Flow of Inspiration Principle 1 is essentially accomplished by the existence of the SGDA online submission system). In each section we attempt to answer the two questions presented above for a given Flow of Inspiration Principle.

4.2 Viewing and Running Fellow Students' Projects

As shown in Figure 2, running a fellow classmate's project can be done in the browser via clicking the "Run" link on the project page. Students were asked to estimate how many times per week they played a classmates' game on their own volition (not as part of an assignment). 95% (19/20 students) of students answered that they played at least one classmate's game a week and 85% (17/20) played 2 or more classmate's games a week with most students, 45% (9/20), playing exactly 2 games a week. Figure 3 is a graph that shows what benefits students hoped to gain when they played classmates' games on the SGDA.

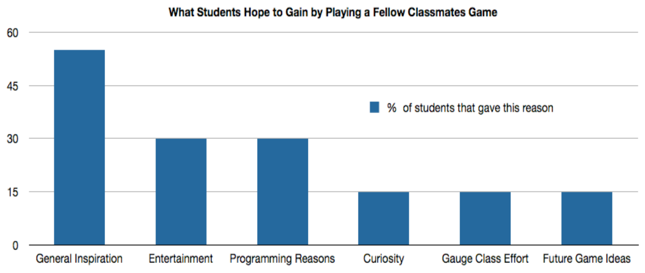


Figure 3: Graph that shows what benefits students hoped to gain from playing SGDA games.

Most students, 55% (11/20), mentioned general inspiration as the reason for playing classmates' projects. 30% (7/20) of the responses mentioned specific programming reasons for playing a fellow student's project. These include, among other things, looking for in-game graphics ideas, game play ideas and in-game agent interactions. Furthermore, 30% of students played games for pure entertainment purposes. Finally, 15% (3/20) of students mentioned they played games out of curiosity for what their fellow classmates were turning in and to gauge class effort for a given assignment. The following quote is representative of how many students answered this question:

"When I play a game created by someone else, I look for interesting ideas and design, and hope to find some inspiration for my own games. I also hope to be entertained for a while by their game."

Given that so many students hoped to be inspired by a classmate's game, it begs the question what effect, if any, does playing classmates' games on the SGDA have on a given student's project submissions. Asked if they had ever changed their project after playing a classmate's version of the game, 52.6% of the students said 'yes' (10/19 with 1 person abstaining). Of those 52.6%, when asked the reason for changing their game, 100% (10/10) replied that they wanted to improve their own game based on what they saw in their classmate's game. The following is a typical student quote:

"On the centipede gamelet i did notice other games made the centipede smarter , so i changed so it wouldn't get stuck in certain situations."

Furthermore, students were asked if a classmate's game they played from a previous week's assignment had ever inspired the current week's assignment; 57.9% (11/19) said yes. One gave the following answer:

"Yes, my Digital Logic gamelet was inspired by one of my classmates who had done a digital logic gamelet the week before. I got the idea of using logic circuits from their game and molded my own game out of it."

The questionnaire results indicate that students played other students' projects on a weekly basis, and thus, the SGDA is effective in enabling and motivating students to view and run, fellow student's projects. Moreover, after playing classmates' games, students often use the experience to try to improve their own game or as inspiration for a future game. The data shows strong evidence that the SGDA helps to increase the quality of work submitted as students are trying to improve upon what they turn-in based on what their peers have done.

4.3 Providing Feedback on Fellow Students' Projects

The SGDA provides two formal mechanisms for leaving feedback. The quickest way is to star rate a given project. The hope is that over time, the amalgamation of star ratings should yield a consensus as to what particular projects were liked and disliked by the class as a whole. Star ratings can be made anonymously.

Commenting is the other formal way students can leave feedback. Comments allow students to give a more in-depth critique, or a better description of things they liked. Students must give a "handle" to leave a comment which could be the student's real name but did not have to be.

The final way students can provide feedback is through in-class verbal feedback on a classmate's game. One could argue that this is not explicitly an SGDA feedback mechanism as students could provide verbal feedback in any class regardless of the existence of an online infrastructure. However, students must be able to play a classmate's project before having the ability to give verbal feedback, and the characteristic of playing any classmate's game is explicitly enabled by the SGDA. The fact that students have to submit their assignments publicly to the class enables in-class verbal feedback among students to exist. The questionnaire queried students about all three types of feedback.

When asked how many times this semester they had star rated a classmate's game on their own volition, 40% replied seven or more times (8/20); 80% of the class said 3 or more times (16/20). These results seem to indicate that there is motivation to use the star-rating functionality in the SGDA. When asked to give the reasons for star rating a classmate's game, the overwhelming majority replied that they star rated because they liked the game. Figure 4 is a graph that shows all the reasons students said they star rated games. 61.11% (11/18 with 2 abstaining) said they star rated when they liked the game;

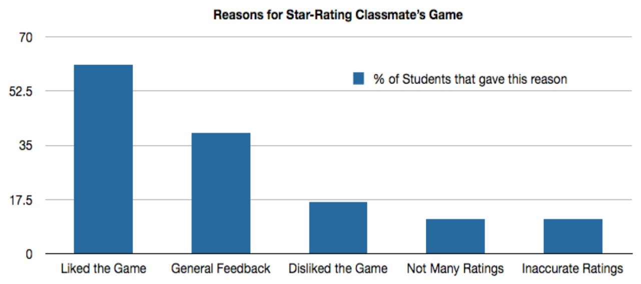


Figure 4: Reasons students gave for star-rating a classmate's game. Most people star rated when they liked a game.

38.89% (7/18) said they star rated as a general feedback mechanism. Only 16.67% (3/18) said that they star rated when they disliked a game. In the current state, most star ratings are only given by the subsection of the class which view the game positively. Therefore, if true, the SGDA star rating does not reflect a class consensus as to whether a given game is good or bad because the whole class is not involved in rating a particular game. One student wrote the following:

"Usually I leave a rating if it's a great game... not quite as often if I didn't think it was very good."

This seems to indicate an area where the SGDA needs to be modified such that students are more willing to honestly rate games they find good and bad. A few students suggested that star ratings be updateable, meaning, that if a student initially gives a project a low star rating, if the project improves, the student could go back and replace the initial star rating. Currently, a student can only give one star rating for a given project, and it cannot be changed. Furthermore, other students suggested that it should be possible to tie a star rating explicitly to a comment such that if you give a low star rating you could couple it with an explanation as to why the star rating was so low; this is backed by the SGDA comment feedback data that will be presented next. This indicates that students would not mind being critical with star ratings as long as there exists a way to make the low star rating constructive criticism. The reluctance to leave a bad star rating is at least partly tied to the lack of an SGDA mechanism to explain the criticism and reward an improved project.

When students were asked how many times this semester they left a comment on a classmate's project, 25% (5/20) responded seven or more times, 75% (15/20) said they left a comment 2 or more times, and 85% (17/20) left at least 1 comment. Though there is room for improvement, this data seems to show that students are leaving comments on classmates' projects. Figure 5 shows the student responses when asked the main reasons for leaving a comment. Again, as with the star rating, students left comments because they liked a given game (35.29%, 6/17 with 3 abstaining); however, in contrast to the star rating, equally as many students left comments to suggest improvements (35.29% 6/17). As mentioned above, it appears that students are less reluctant to criticize as long as they can explain their criticism.

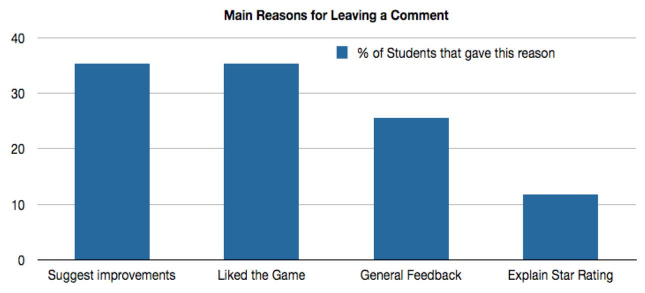


Figure 5: Reasons students left comments. Mostly students commented because they liked the game or to suggest improvements.

When students were asked whether they had ever updated a project based on comments received, 36.85% said 'yes' (7/19, 1 abstaining). The people that said 'yes' replied that usually the comment pointed to a specific problem with their project that they corrected. The following quote is typical of the 'yes' responses received:

"YES! On some of the earlier projects I submitted early, and the feedback told me it was too hard, or the interface was awkward. So I switched it up, went back and resubmitted."

On the other hand, 63.16% (12/19) answered 'no'; however, it should be noted that of the 'no' responses, at least 4 said that they were intending to use comments made on a previous game for a future game including game-remakes they were planning on doing for their final projects. Many of the 'no' responses were because students either commented after the project was due or the author of the project only looked at the comments post due-date.

When asked if comments left on previous projects influenced their current projects 38.89% said yes (7/18, 2 abstaining); 61.11% (11/18) said 'no'. Many of the 'no' responses refer to the fact that since the assignments are so different, it is hard to apply previous project comments to current projects. The following is a typical quote from a student who gave the 'no' response:

"No, since the gamelets are so different, the comments usually only apply to that project."

Given that this is the case, it makes sense that the students who are remaking old games for their final project plan on taking older comments into consideration. Students are commenting and, to a certain extent, using the comments to improve their projects. However, it seems like the SGDA has to increase motivation for pre-deadline commenting on projects. Moreover, it might be helpful to send an email to the author such that if a comment is made before the deadline, the project author sees it before turning the project in rather than after the deadline. Furthermore, 57.14% of students (8/14, 6 abstaining) said they would comment more if they had more time. The following is a typical quote from these answers:

"If I had more time!. . . When I spend 15-20 hours (from start to finish) putting a game together and submitting it, and am faced with another game due in 5-6 days, it's tough to motivate myself to go play a bunch of other folks' games. . ."

Since students feel the workload is so large, it seems promising that comments are occurring using the SGDA, and students are using these comments to update their own games. A workload change or phased development wherein students can leave comments after an initial project phase might allow more time for comments to be made and an opportunity for comments to readily be used in project development.

When asked if other students had provided verbal in-class feedback on a project (under their own volition, not as part of an assigned discussion), 75% said 'yes' (15/20). Furthermore, when asked if they had ever changed their project because of in-class feedback received 53.5% said 'yes' (8/15, 5 said 'no' to the last question). Interestingly, it seems like the SGDA has increased in-class feedback, and this, in turn, has led to students attempting to improve their project based on verbal feedback received. The above quote about how more time would allow for increased SGDA comments, provides insight into why many students might prefer to instead give feedback verbally in-class. Since students come to class anyways, playing a fellow classmate's game and giving comments in-class does not take away from the time the student spends outside of class on the projects. Finally, it should be noted that this in-class interaction mirrors the interactions that actually happen in middle school computer clubs, and these interactions are the very ones we want to promote regardless of the feedback mechanism used.

4.4 Downloading and Viewing Project Code

In Figure 2 we see that in addition to running a classmate's game for a given assignment, students have the option of downloading the project as a .zip file. This allows students to look at other students' code to figure out how they accomplished different interactions. When downloading code, the student must login. The reasons the SGDA requires student login for downloading is to discourage outright cheating (identically copying someone else's code) and to track influence among students. Students are encouraged to investigate other students' code and even take parts of their classmate's code as long as they make it their own. When asked if they had ever downloaded a classmate's project code, 57.89% replied yes (11/19, 1 abstaining). Furthermore, of those who downloaded code, 75% replied that the code was helpful (6/8, 3 abstaining). The general consensus among those who were helped by downloading was they were trying to figure out something specific, and by downloading the code, it put them on the right track to accomplishing this. The following is a quote that is typical of what most students who found downloading helpful said:

"Yes, I downloaded code a few times. Usually they had a cool feature in their game and I wanted to see how they did it. Yes it did help, it showed me how the feature was implemented."

During the course of the semester, many students expressed reluctance to download code even though it was encouraged from the first day of class. When asked if they felt downloading a classmate's code for a given assignment before turning in the assignment was wrong, 25% (5/20) answered 'yes'. Furthermore, 15% (3/20) felt that it was a gray area. Many of these people thought that it was all right after the assignment was due or later in the semester when students submitted individual projects (as opposed to the first four weeks wherein everyone submitted similar games like 'Frogger' or 'Sokoban' etc.)

From the questionnaire answers it looks like the SGDA along with the Educational Game Design Class is effective in motivating students to download code allowing ideas to easily flow from one student to the next. It also seems like students want more guidelines as to when downloading and using someone else's code should be allowed and when it should be prohibited. A possible solution to this would be to first explain the middle school computer club motivation for allowing students to download a classmate's code. Furthermore, as was suggested with the star ratings, maybe if we could link the downloads with comments, such that students could openly provide feedback on the downloaded code, it might highlight the middle school computer club context. Overall, student-downloading seems to be occurring using the SGDA, and students who download often attempt to use the information garnered to improve their project.

5. CONCLUSIONS

The results of the questionnaire indicate that the Scalable Game Design Arcade is effective in implementing the Flow of Inspiration Principles in the Educational Game Design class. Students use the SGDA to view other students' projects. To varying degrees students also use one of three feedback mechanisms enabled by the SGDA to appraise classmates' projects. A sizeable portion of students download and view classmates' code. Evidence strongly suggests that students are attempting to increase the quality of their work after viewing classmates' projects, receiving feedback on their own projects, and downloading classmates' code. Informally, this is further verified by our prior experiences with the Educational Game Design Class; the current SGDA class implementation coincides with a marked increase in the quality of submitted projects. The SGDA seems to enable and motivate the emergent middle school computer club peer-to-peer interactions in the Educational Game Design Class.

Future SGDA iterations will focus on the shortcomings highlighted by the questionnaire. Creating a class with phased gamelet development such that students have more time to comment on fellow students' code might allow for increased numbers of students to use the SGDA online feedback mechanism more often. Allowing students to explain their critical star ratings could enable more honest evaluations of classmates' projects. Finally, making the download policy more explicit might allow students to be less tentative downloading and using classmates' projects for their own purposes.

This questionnaire was meant to analyze the cyberlearning infrastructure and its effects on the Educational Game Design Class. However, the questionnaire data not only indicates the extent to which the Scalable Game Design Arcade accomplishes the Flow of Inspiration Principles, but also, possibly gives insight into how the more successful SGDA characteristics are actually enabled by the physical classroom. For example, students preferred to give in-class verbal feedback of classmates' games; furthermore, in-class feedback led to many students altering their games based on this feedback. More investigation is necessary to see the reasons why in-class feedback was popular, but one possibility is that face-to-face interaction might be preferred over online feedback mechanisms. Or possibly, being in the vicinity of others lends itself to feedback that might have not otherwise been voiced. Some quotes from the questionnaire support the idea that when students were programming their games in-class often people would randomly stop by and give unsolicited feedback. Furthermore, a few students said they asked fellow classmates' in-class for game feedback presumably as a quick and easy way to see what was working and what needed improving. Though it might be possible to provide and receive this classmate interaction online, it seems more efficient to do it in class. Regardless of the reason, the point still remains that the questionnaire pointed to classroom interactions being a very important part of accomplishing the Flow of Inspiration Principles. Thus, the "Sage on the Stage" teaching approach might have little need for a physical classroom anymore; however, these initial findings look like a promising argument for the essential role of the physical classroom within teaching strategies that employ peer-to-peer student learning.

6. ACKNOWLEDGMENTS

We would like to thank all the students of the Educational Game Design Class at the University of Colorado Boulder. Our thanks to Centennial Middle School and the Boulder Valley School District. A special thanks to David Webb for helping with the evaluation methodology. This material is based in part upon work supported by the National Science Foundation under Grant Number No. DRL-0833612. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] Fischer, G., Rohde, M., Wulf, V. 2007: Community-based learning: The core competency of residential, research-based universities. International Journal on Computer-Supported Collaborative Learning (iJCSCL), Vol. 2, No. 1., pp. 9-40.

[2] Williams, L., Layman, L., Slaten, K. M., Berenson, S. B., and Seaman, C. 2007. On the Impact of a Collaborative Pedagogy on African American Millennial Students in Software Engineering. In Proceedings of the 29th international Conference on Software Engineering (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 677-687.

[3] Pollard, S. and Duvall, R. C. 2006. Everything I needed to know about teaching I learned in kindergarten: bringing elementary education techniques to undergraduate computer science classes. In Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM, New York, NY, 224-228.

[4] Hamer, J., Cutts, Q., Jackova, J., Luxton-Reilly, A., McCartney, R., Purchase, H., Riedesel, C., Saeli, M., Sanders, K., and Sheard, J. 2008. Contributing student pedagogy. SIGCSE Bull. 40, 4 (Nov. 2008), 194-212

[5] Repenning, A. and Ioannidou, A. 2004. Agent-based end-user development. Commun. ACM 47, 9 (Sep. 2004), 43-46.

[6] Repenning, A. and Ioannidou, A. 2008. Broadening participation through scalable game design. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 305-309.

[7] Waite, W. M., Jackson, M. H., Diwan, A., and Leonardi, P. M. 2004. Student culture vs group work in computer science. SIGCSE Bull. 36, 1 (Mar. 2004), 12-16.

[8] McKinney,D., Dyck, J., Luber, E. 2009. ITunes University and the classroom: Can podcasts replace Professors? Computers and Education. Vol 52, 617-623.

[9] Chorost, M. (2008). Educating Learning Technology Designers: Guiding and Inspiring Creators of Innovative Educational Tools . New York: LEA. 357

[10] Christine, H. Abelson, L. Dirks, R. Johnson, K. R. Koedinger, M. C. Linn, C. A. Lynch, D. G. Oblinger, R. D. Pea, K. Salen, M. S. Smith, and A. Szalay, "Fostering learning in the networked world: The cyberlearning opportunity and challenge, a 21st century agenda for the National Science Foundation," NSF Report, June 2008.

[11] Head, C. C. and Wolfman, S. A. 2008. Poogle and the unknown-answer assignment: open-ended, sharable cs1 assignments. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 133-137.

[12] Wing, J. M. 2006. Computational thinking. Communications of the ACM 49, 3 (Mar. 2006), 33-35.

[13] Papert, S. 1996. An Exploration in the Space of Mathematics Education. Journal of Computers for Mathematical Learning. Vol. 1, No. 1, pp. 95-123

Re-Thinking Computer Literacy in Post-Secondary Education

Satindranath Banerjee
Department of Computer Science
University of Calgary
2500 University Dr. NW, Calgary, T2N 1N4
+1 403 472 7531
smbanerj@ucalgary.ca

Jalal Kawash
Department of Computer Science
University of Calgary
2500 University Dr. NW, Calgary, T2N 1N4
+ 1 403 220 6619
jkawash@ucalgary.ca

ABSTRACT
Computer literacy can no longer be reduced to knowing how to operate a computer. Nonetheless, it must embrace making use of computers to solve problems. This paper describes a post-secondary computer literacy course with this objective, focusing on problem solving using computers, and encouraging methodological thinking. The course includes "real" computer science topics that are tailored to fit the backgrounds of non-computing related majors. The paper also shares our preliminary observations from running the course.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education, Curriculum, literacy.*

General Terms
Design, Human Factors, Theory

Keywords
Computer literacy, problem solving

1. INTRODUCTION
It is no longer acceptable to consider computer literacy in post-secondary education as merely knowing how to use a computer at a basic level. Graduates from different non-computing disciplines often find themselves obliged to use computer software to solve problems that require more than just knowing how to use the software itself. They find themselves involved in issues related to problem solving using computers, something that a basic computer literacy course often ignores. Furthermore, many of these graduates end up working in multi-disciplinary teams, which include, among other members, computing professionals. Knowing what computer science is all about, even at a very basic level, becomes indispensable to allow these multi-disciplinary teams to efficiently and effectively function.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

In this paper, we describe a computer literacy course that we have designed and delivered at the University of Calgary. The course is geared towards training the students in problem solving using computers. It gives the students, who are not majoring in computing-related fields, a peek into computer science. There are many difficulties stemming from this objective. First, the course has to be general enough to embrace a diverse target audience with different backgrounds, majors, and objectives. Second, it has to be specific enough to have some meaningful content. Third, we have to make minimal assumptions about the background required to study the topics of the course. Hence, we have to convey core problem solving concepts and skills from within computer science practice that will be useful across a range of disciplines.

As Bain [1] notes, the goal is not merely to convey a set of facts about computers, but to develop a style of thinking characteristic of the computer science discipline. This forces us to ask ([1], pp 85): "What reasoning abilities will students need to possess or develop to answer questions that the discipline raises? How can we cultivate the habits of mind that will lead to constant use of those intellectual skills? "

A look at computer literacy textbooks that are available or adopted in post-secondary institutions (for example see [3-7]), quickly reveals the absence of a textbook that is suitable to achieve our objective. This leads us to conclude that our attempt may be one of very few (we are not aware of other similar attempts, but this does not preclude their existence) so that the development of a textbook is justified from a business point of view.

We describe the course structure and topics in Section 2. Section 3 gives some specific multidisciplinary examples used to augment the core topics. Section 4 gives some preliminary observations that we collected while involved in delivering the course, and Section 5 concludes the paper.

2. THE COURSE

CPSC 203 is a first year computer literacy course for non-majors. The bulk of our students are from the management and social sciences schools. Yet, students from natural sciences, communications, and other disciplines also register in the course. Hence, there is a wide variety of students who take the course. It is a required course for students taking the University of Calgary Haskaynes Business School's 3rd year information systems management courses, and is an elective course for other disciplines.

2.1 Course Format

The course is a 3-credit hour course delivered in a combination of lectures and tutorials: 150 minute lectures and 100 minute tutorials every week in a 13-week semester. The course's intake of students is at least 700 per semester, with lectures ranging from 100 to 175 students and a cap of 25 students for each tutorial. Staffing requirements are from 2-4 instructors and at least 20 teaching assistants (TAs) per semester.

2.2 Course Content

The course's objective is achieved via a set of concepts and a set of skills. While these are not completely independent, the concepts are covered in the lectures and the skills are covered in the tutorials. The scheduling takes into consideration the relationships between the concepts and skills so that they are run in parallel between lectures and tutorials. The concept topics include:

FUNDAMENTALS

- **Logic, naïve set theory, and algorithms:** propositional logic, truth tables, inference, quantifiers, sets, relations, and functions, algorithms, correctness and efficiency of algorithms, statistical analysis.
- **Graphs and trees:** graphs (undirected, directed, and labeled), Euler tours and circuits, example graph algorithms such as graph coloring, applications to scheduling, trees, binary trees, coding, Huffman's coding algorithm, finite state machines.

ARCHITECTURE

- **Computer Organization:** modern computer components, memory hierarchy, multi-core machines, magnetic and optical disk operation, data paths, binary numbers, logic design, adders, decoders, CPU and memory design.
- **Computer Networks:** client-server organization, the Internet, TCP/IP, UDP, WWW, HTTP, naming, cryptosystems, mutual authentication, firewalls.

APPLIED PROBLEM SOLVING

- **Databases:** Entity-Relationship (ER) modeling and ER diagrams, database schema, translating an ER diagram to database schema, first, second, and third normal forms, SQL, set operations in SQL, natural joins in SQL, aggregate functions in SQL.
- **Programming:** Python programming, variables, functions, conditionals, lists, loops, searching and sorting, top-down and bottom-up designs.

The fundamentals provide a basic framework of concepts in which problem solving skills can be developed through clear logical reasoning and the construction of simple graph-based models. The fundamentals are often illustrated with "classic" examples from computer science. These fundamentals are then augmented with an understanding of modern systems. Finally, we focus on solving real-world problems taken from various domains (business, sciences, and arts) first through developing database applications, and finally via developing simple programs. In the last section, we repeatedly emphasize the use of fundamentals to design an appropriate solution.

The resulting lectures cover a smaller set of concepts than typical of many survey courses – but cover them in greater depth than most survey courses. Students are introduced to fundamental concepts in computer science with sufficient grounding that they could pursue higher level (2nd or 3rd year) courses in the same subjects. Judicious choice of problems and examples is used to appeal to the diverse backgrounds of the students.

TUTORIAL MODULES

- **MODULE 1 - Problem Solving via Spreadsheets:** basics of statistical analysis in spreadsheets, spreadsheet design, visual display of quantitative information.
- **MODULE 2 - Problem solving via Databases:** data organization, query driven data analysis, practice using the relational model.
- **MODULE 3 - Problem solving via Programming:** emphasis on image processing, practicing the programming concepts developed in lectures.
- **MODULE 4 - Problem Solving via Design:** emphasis on integrating the previous skills in the realistic design of a volunteer information system, with a focus on team design. Practice at the use of top-down design, use cases, ER modeling in constructing information system designs.

Each module runs approximately for 3 weeks, and the first three modules conclude with a time-boxed (50 minutes), in-tutorial assignment where students demonstrate their "hands-on" mastery of the skills developed. The final module in tutorial assignment was co-developed with the Haskaynes business school. In the course of two tutorial sessions, students work in teams to design a volunteer management system, based on the actual workings and data originating from the University of Calgary's Olympic speed skating oval.

TAKE-HOME ASSIGNMENTS

- **Assignment 1 - Developing an information dashboard in spreadsheets:** At the beginning of the term, students fill in a "Who are we" survey that captures information on their technological interests, website usage, and simple demography information. Their task is then to analyze this data and reduce it to a visual display (information dashboard) that summarizes this data.

- **Assignment 2 - Database System Development:** Students are asked to re-organize the information in Assignment 1 into a small, properly designed relational database – and within that context are asked to develop further analyses using multi-table queries that would be difficult to achieve in a spreadsheet. Finally, they are asked to develop further insights on "Who are We" via query driven analysis.
- **Assignment 3 (Term Project) - Technology Issues:** This assignment allows us to leverage the diverse backgrounds of our students. Students form groups, research a specific technology (usually Internet based) and corresponding issues (business, economic, social, ethical) arising from the technology, and develop a focused Wikipedia-like entry, summarizing their research. This exercise gives the students an opportunity to explore the ways in which technology relates to broader issues in society and it exposes the students to working in teams.

The take home assignments are designed to provide a framework for students to integrate lecture concepts with tutorial skills, and explore issues in greater depth, by developing their own knowledge and solutions. A number of these projects with relevant topics have taken on a life of their own, becoming among some of the most frequently viewed Web pages in the University of Calgary Wiki [2].

In addition to the take-home assignments, focusing on real-world problems, four time-boxed assignments were conducted in the form of sit-in quizzes. These sit-in assignments have more tightly scoped problems that could be solved in 50-minute sessions.

TIME-BOXED ASSIGNMENTS

- **Quiz 1 – Data analysis with spreadsheets:** The quiz consists of a simple data set and a series of questions aiming at testing the students on using spreadsheets and deriving conclusions.
- **Quiz 2 – Databases:** This quiz consists of a data set which the students must convert to a database. A series of queries must be formulated on the database.
- **Quiz 3 – Programming:** Students are given a simple programming exercise, such as editing an image via modifying code and calling functions within a Python program.
- **Quiz 4 (Group Quiz) – System Design:** Unlike the previous three quizzes, this assignment is conducted in groups of 5 students. It spans two tutorial sessions. In the first meeting students are shown a short documentary on the operation of the University of Calgary's Olympic Oval during international skating events. After which, they have to start designing a computer system that administrates volunteering during such events. Students continue working on the design outside the tutorial and complete and submit their work in the following tutorial (tutorials are one day apart). The objective of the quiz is to design use-case scenarios, a database using entity-relationship modeling, and user interface components. No special software is used and the design can be completed on paper.

3. SOME SPECIFIC EXAMPLES

In this section, we provide some specific examples on how to tie some of the core computer science subjects to examples from different disciplines. Not all core subjects can be tied with such examples. After all, this is a computer science course that is primarily concerned with teaching computer science.

- **Arts**: image manipulation in the context of programming; visual information design principles.
- **Natural Sciences**: illustrating data analysis for scientific examples, such as the effect of DDT on bald eagle populations; distinguishing hybrid swarms from their parent species in Iris plants.
- **Business:** data analysis of male-female life insurance premiums for smokers and non-smokers; scheduling through graph coloring.
- **Law:** logic of text guides and valid and invalid arguments; analysis of voting bias in the U.S. supreme court.
- **Social Sciences:** analysis of survey data (such as the "Who are we" survey conducted at the beginning of the semester); analysis of the relationship between economic indicators and incumbent-President advantage in U.S. elections.
- **Media Technology:** Each semester new examples are developed by students in the course of their term projects. A few examples from recent semesters listed among the top 50 most frequent views on the U of C Wiki are:
    - Photoshop and body-image ("Maybe She's born with It, Maybe It's Photoshop").
    - Illegal media sharing ("Illegal Media Sharing").
    - Health issues and virtual games ("Guitar Hero or Guitar Zero").
    - Human co-evolution with Ipods ("Ipod: Man's New Best Friend").

4. OBSERVATIONS

Overall, the feedback we have received from students has been very encouraging. The students' feedback is collected from three different sources: Confidential course and instruction evaluation surveys collected at the end of each semester, assessment and analysis of exams and assignments, and verbal discussions with students inside and outside the class room. In addition to the students' feedback, we will also discuss our own observations as instructors of the course.

It is worth mentioning the exams are formulated such that at most 40% of the questions are multiple choice, and the remaining questions consist of problems to be solved using written answers.

The majority of students found the course *challenging but very interesting*. Many reported that after the course they have a *different perspective on how to approach problems* and that the course *improved their ability to solve problems*, in a conscious way. Some students were disappointed with the complaints: "it is not what I have expected" or "this was supposed to be an easy course".

The most difficult topics for the students were: programming, logic, queries that require joining tables, and some of the complex logic circuits. The most favorable topics where the students demonstrated competency were: database design, basic SQL, graphs, and trees, including finite state machines and Huffman's coding.

We had to concretize the fundamentals with every-day, real-world examples to allow students to grasp the concepts. We noticed that it takes a few weeks for students to abstract databases using set theory and work properly with them.

Table 1 shows the average and standard deviation of grades for the student's written responses to long-answer questions in the midterm and final exams measured for almost 320 students in two sections in one semester (during this semester there were no long-answer questions from the subject of logic and naïve set theory).

A summary of the testing conducted in the tutorials is shown in Table 2 (for take-home assignments) and Table 3 (time-boxed assignments)

There was some anecdotal evidence from TAs that during the programming modules, students fell into two groups: those who intuitively grasped the programming concepts and those who struggled, and tended to imitate the provided examples. Note the lower average grade and higher standard deviations associated with programming topics in Tables 1 and 3.

The addition of homework problems as part of lecture notes may help students who need additional practice, particularly with the topics that were identified as difficult.

In the tutorials, the diverse backgrounds of the students required the TA's to be able to adjust their pacing to match the capabilities of students in their particular tutorial. Finally, the best term projects showed a surprising degree of subtlety and depth on the part of students in relating technology to a larger world view.

Table 1. Analyzing student's exam responses by subject

Subject	**Average Grade**	**Standard Deviation**
Database design	74%	27
Single-table SQL queries	82%	32
Multi-table SQL queries	42%	25
Finite State Machines	81%	27
Coding	70%	36
Logic circuits	72%	32
Tracing Python programs	54%	29
Constructing Python programs	28%	35

Table 2. Analyzing student's take-home assignments

Subject	**Average Grade**	**Standard Deviation**
Assignment 1 Data analysis with Spread Sheets	79%	20
Assignment 2 Databases	81%	24
Assignment 3 Term project	82%	22

Table 3. Analyzing student's time-boxed assignments

Subject	**Average Grade**	**Standard Deviation**
Quiz 1 Spreadsheets Data analysis with Spread Sheets	69%	22
Quiz 2 Databases	80%	19
Quiz 3 Programming	63%	31
Quiz 4 System Design	84%	15

5. CONCLUSION

We believe that computer literacy in the telecommunications age cannot be simply reduced to knowing how to operate a computer. It must be geared towards problem solving using computers. Making use of computers to solve problems requires methodological thinking. We have shown that it is possible to train students, from non-computing domains, to develop a style of thinking that is appropriate for this objective. We hope that our experience with CPSC 203 will inspire and encourage others to adopt similar approaches.

Finally, the absence of a suitable textbook that covers all the material covered in this course has lead one of the authors to develop a new textbook for this purpose [8].

6. REFERENCES

[1] Bain, K. 2004. What the Best College Teachers Do. Harvard University Press.

[2] http://wiki.ucalgary.ca/page/Courses/Computer_Science/CPSC_203/CPSC_203_Template/CPSC203_Term_Projects

[3] Beekman, G. and Quinn, M. J. 2005. Computer Confluence: Tomorrow's Technology and You. Prentice-Hall.

[4] Snyder, L. 2007. Fluency with Information Technology: Skills, Concepts, and Capabilities (3rd Edition). Addison Wesley.

[5] Evans, A, Poatsy M .A., Martin, K. 2005. Technology in Action (5th Edition). Prentice Hall

[6] Anderson, G., Ferro, D., Hilton, R. 2005. Connecting with Computer Science. Thompson.

[7] Brookshear. J.G. 2009. Computer Science. An Overview. Addison Wesley.

[8] J. Kawash. 2009. Peeking into Computer Science (1st custom edition for the University of Calgary), scheduled for September 2009.

Teaching Page Replacement Algorithms with a Java-Based VM Simulator

Fadi N. Sibai
College of Information Technology
UAE University
Al Ain, UAE
+(971) 3 7135589
fadi.sibai@uaeu.ac.ae

Maria Ma
Fidelity
Investments
Toronto, Canada
ruifangma@hotmail.com

David A. Lill
Motorola Inc.
Schaumburg, IL
USA
david.lill@motorola.com

ABSTRACT

Computer system courses have long benefited from simulators in conveying important concepts to students. We have modified the Java source code of the MOSS virtual memory simulator to allow users to easily switch between different page replacement algorithms including FIFO, LRU, and Optimal replacement algorithms. The simulator clearly demonstrates the behavior of the page replacement algorithms in a virtual memory system, and provides a convenient way to illustrate page faults and their corresponding page fault costs. Equipped with a GUI for control and page table visualization, it allows the student to visually see how page tables operate and which pages page replacement algorithms evict in case of a page fault. Moreover, class projects may be assigned requiring operating system students to code new page replacement algorithms which they want to simulate and integrate them into the MOSS VM simulator code files thus enhancing the students' Java coding skills. By running various simulations, students can collect page replacement statistics thus comparing the performance of various replacement algorithms.

Categories and Subject Descriptors

D.3.2 [Java]

General Terms

Algorithms, Performance

Keywords

Simulation, Page Replacement, MOSS virtual memory simulator

1. INTRODUCTION

Demand paging in modern operating systems ensures that disk pages are brought to memory when needed. It greatly helps in allocating the memory's limited resources to serve running code. A page table contains page entries with virtual and physical addresses as well as page protection and access rights and other information, When a page containing a needed datum or instruction is searched in a page table --or its cached TLB version-- and found to be missing from main memory, a page fault is said to occur. As the size of main memory is limited and is much smaller than the size of permanent storage, the role of page re-placement is to identify the best page to evict from main memory as a result of a page fault and replace it by the a new page from disk that contains the requested datum or instruction. The problem is very similar to the block replacement in cache memories except that it may argued that the page replacement is more critical as page transfers from disk to memory are orders of magnitudes slower than block transfers from main memory to the cache memory or between two levels of cache memories. Many page replacement algorithms [1, 4-5] exist including but not limited to the Not Re-cently Used (NRU), Second Chance, First In First Out (FIFO), Least Recently Used (LRU), Optimal, and Aging.

The Modern Operating Systems Simulator (MOSS) [2-3] is a memory management simulator developed by Alex Reeder and is used with [4]. In a virtual memory environment where main mem-ory is divided into equally sized pages, the MOSS simulator facilitates the visualization of page replacements and the generation of page faults. During simulation runs, the simulator displays the status of pages, and displays the physical pages in memory to which virtual pages are mapped to, and the status of the last page accessed including the following attributes

i. *page fault* which indicates whether that access resulted in a page fault or not;

ii. *virtual page* which indicates the last accessed page's virtual page number;

iii. *physical page* which indicates its corresponding physical page number;

iv. *R* which indicates whether this page has been read or not;

v. *M* which indicates whether this page has been modified or not;

vi. *inMemTime* which indicates how many nanoseconds ago that the physical page was allocated to this virtual page; and

vii. *lastTouchTime* which indicates how many nanoseconds ago that the physical page was last modified.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

After the initial page table is configured, the simulator reads and runs a series of virtual mem-ory instructions, either being a read or a write, one instruction at a time, and after each simulation step, displays the above page attributes in a win-dow. As such, it serves as a great educational tool as the apprentice can observe how page replace-ment algorithms operate in practice. The source code for the simulator is written in the Java programming language and is available for download. The availability of the simulator's source code al-lows one to modify specific modules in order to implement specific page replacement algorithms, graphically visualize how they replace pages and operate, and collect performance statistics to evaluate their goodness. Typically, the goodness of page replacement algorithms is reflected by the number of page faults that occur for specific se-quences of virtual memory operations. The better the algorithm, the lower is the number of page faults that are generated during the simulation runs.

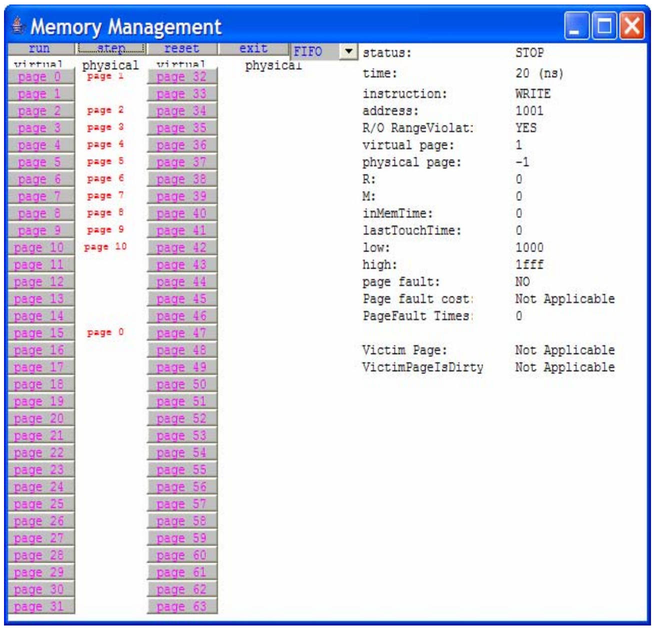
By modifying and extending the MOSS memory management simulator, we developed a useful page replacement tutor in the Java language which allows the user to easily switch between different page replacement algorithms including FIFO, LRU and Optimal replacement algorithms. The simulator clearly demonstrates the behavior of the page replacement algorithms in a virtual sys-tem and provides a convenient way to illustrate page faults and their corresponding page fault costs. We describe the changes and code additions we made to the MOSS simulator to turn it into a page replacement tutor. We also describe some experiments conducted by the operating system class students who used this enhanced simulator. The students integrated new Java code into the MOSS simulator source code corresponding to 2 scheduling algorithms (Optimal and LRU) complementing the default FIFO replacement algo-rithm already implemented by MOSS.They then made simulations with the various page replace-ment algorithms and various configuration files corresponding to predetermined and ran-domly-generated memory traces to step through the simulations and watch each page replacement algorithm "in action", and watch how each algorithm differs in its selection of victim pages to evict or replace. The simulator's GUI also allowed the students to constantly view the virtual page to physical frame mappings and the fullness of the page table explaining why page faults are raised and their timing. The students collected statistical results from the simulation runs under various re-placement algorithms, in order to compare their performance.

The following sections detail the changes we made to the MOSS memory management simula-tor, which will be followed by a discussion of the uses of this education tool and examples of simulation experiments which operating systems stu-dents can conduct with this tool.

2. New Simulator Features

2.1 Page Replacement Algorithm Options

In virtual memory management systems, the key algorithm that impacts the overall performance of the virtual memory system is the page re-placement algorithm. A good replacement scheme can reduce the page fault cost resulting in higher performance, since the more page faults the op-erating system encounters, the more resources are wasted on paging in/out instead of doing useful work, resulting ultimately in serious thrashing problems with ping-pong effects and with serious performance degradations. Page replacement is still a hot topic [7-10] and has ventured into new memory technologies such as Flash memory [11]. In Fig. 1, we show the modified MOSS simulator graphic user interface (GUI). The 4th button (rightmost) with FIFO label is a pull-down button that we added to allow the easy selection between FIFO (default), Optimal, and LRU replacement schemes. On the right side of the GUI, we also added the following new attributes which we will explain later: *R/O RangeViolat, Page Fault Cost, Victim Page,* and *VictimPageIsDirty.*



```
status:             STOP
time:               20 (ns)
instruction:        WRITE
address:            1001
R/O RangeViolat:    YES
virtual page:       1
physical page:      -1
R:                  0
M:                  0
inMemTime:          0
lastTouchTime:      0
low:                1000
high:               1fff
page fault:         NO
Page fault cost:    Not Applicable
PageFault Times:    0

Victim Page:        Not Applicable
VictimPageIsDirty   Not Applicable
```

Figure 1. Virtual Memory Management Simulator Graphic User Interface

2.2 Page Fault Calculation and Presentation

The number of page faults and the page fault cost that result from the execution of the program are the main factors in determining whether an ap-plied page replacement algorithm is good or not. The fewer the number of page faults and the lower the page fault cost, the better is the performance, and the better is the replacement algorithm. This is because as the good page replacement algorithm reduces the page fault overhead, it reduces the amount of I/O actions in the virtual memory man-agement system.

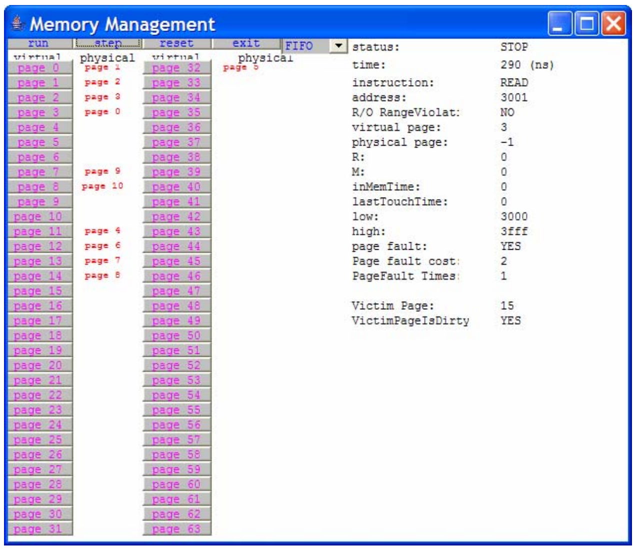
A page fault requires I/O to swap in/out pages. An I/O read operation is required to read the new page to be placed in main memory from disk. In addition, one I/O write operation is required to write out the victim page to disk in the event that the victim page has been modified, in other words, if the page is dirty. We therefore have the following I/O cost for each occurring page fault:

- 1 I/O - Read in replacement from disk to main memory
- 1 I/O - write out victim page from memory to disk only if the victim page is dirty

Thus if the victim page is dirty the total cost of a page fault is 2 I/O transfers, while the cost of a clean (unmodified) victim page is 1 I/O transfer.

Concerning the page fault, we add five labels to the simulator GUI to show the page fault, page fault cost and *victim page*, as shown in Fig. 2. In the situation where page fault has occurred.

- If *page fault* is YES, then a page fault occurred in the last simulation step.
- *Page fault cost* is defined as above.
- *PageFault times* accumulates the number of page faults which occurred during the execution of a batch of command instruc-tions. When stepping one instruction, *Pagefault times* is either 1 or 0 depending on whether a page fault occurred or not.
- *Victim page* indicates the victim page number, and has the value "Not Applicable" if the *page fault* attribute is "NO".
- *VictimPageIsDirty* takes the values of "YES" or "NO" if *page fault* has the value"YES", and"Not Applicable" if *page fault* has the value "NO".



```
status:               STOP
time:                 290 (ns)
instruction:          READ
address:              3001
R/O RangeViolat:      NO
virtual page:         3
physical page:        -1
R:                    0
M:                    0
inMemTime:            0
lastTouchTime:        0
low:                  3000
high:                 3fff
page fault:           YES
Page fault cost:      2
PageFault Times:      1

Victim Page:          15
VictimPageIsDirty     YES
```

Figure 2. New Page Fault Attributes

2.3 Code and Data Alignment

We describe here another change to the simulator. By defining certain pages to be *Code* pages and others to be *Data* pages, we align the Code and Data pages such that there is no mixed code and data in the same page. This definition refinement and distinction can contribute to reducing the cost of the page faults. In fact, if there is a need to page out a code page, since the code page is not allowed to change, we can simply page out the code page without the need to write it out to disk. This will result in a savings of 1 I/O transfer thereby reducing the cost of the page fault. If we did not intro-duce the code/data attribute and did not distinguish between code pages and data pages, all dirty page faults cost 2 I/O transfers.

The *R/O RangeViolation* attribute is added to the simulator to indicate whether the instruction violates the Read-only range, and takes the value "YES" only when a instruction is written into text virtual page range defined in memory configuration file, otherwise it takes the value "NO."

3. Code Implementation of the New Features

The following sections detail the changes made to the MOSS simulator's Java source code in order to implement the above new features and attributes which are needed to implement and evaluate the goodness of the three investigated page replacement algorithms.

3.1 Page Replacement Algorithm Switch Options

To add a pull-down button for selecting a replacement, we made code modifications to the following MOSS simulator files: *Kernel.java* and *Control.Panel.java.*

In the *Kernel.java* file, we added the new attribute *pageRepAlg*, which indicates the page replacement algorithm selected by the user, and which is FIFO by default. Another modification we made in this file, when reset is required, we reset the choice to 0 which is for the FIFO algorithm.

In the *ControlPanel.java* file, we added code related to the algorithm choice. When the user clicks over *run/step*, the program now checks the selection value to figure out the current replace-ment algorithm, and writes the corresponding *pageRepAlg* value in the *Kernel.java* file to reflect the user selection, and applies that page replacement algorithm when a page faults occur. On a reset action, again *pageRepAlg* is reset to FIFO.

We also added the Choice in the global variable definition area as follows

Choice choice = new Choice();

And initialized the choice in the init() as follows

choice.add("FIFO");

choice.add("LRU");

choice.add("OPT");

……..

add(choice);

We also changed the *pageRepAlg* variable in the *Kernel.java* file whenever the user chooses a different algorithm or reset by

kernel.setPageRepAlg(choice.getSelectedItem());

3.2 Page Replacement Algorithm Implementation

To handle different page replacement algorithms based on the user selection, we added code that implements the LRU and Optimal algorithms in the *PageFault.java* file.

The *replacePageLRU* method selects the victim page and swaps it out by applying the Least Recent Used page replacement algorithm. This method checks the *lastTouchTime* parameter for all the pages in memory, and finds out the page with the longest *lastTouchTime* to page out. The *replacePageLRU* code follows.

```
public static void replacePageLRU(Kernel kernel, Vector
mem, int virtPageNum, int replacePageNum, ControlPanel
controlPanel) {
        int count = 0;
        int lruPage = -1;
        int lruTime = 0;
        int firstPage = -1;
        boolean mapped = false;

        while (!(mapped) || count != virtPageNum) {
            Page page = (Page) mem.elementAt(count);
            if (page.physical != -1) {
                if (firstPage == -1) {
                    firstPage = count;
                }
                if (page.lastTouchTime > lruTime) {
                    lruTime = page.lastTouchTime;
                    lruPage = count;
                    mapped = true;
                }
            }
            count++;
            if (count == virtPageNum) {
                mapped = true;
            }
        }
        if (lruPage == -1) {
            lruPage = firstPage;
        }
        Page page = (Page) mem.elementAt(lruPage);
        Page nextpage = (Page) mem.elementAt(
                                        replacePageNum);
        kernel.printPageFaultInfo(nextpage, page);
        controlPanel.removePhysicalPage(lruPage);
        nextpage.physical = page.physical;
        controlPanel.addPhysicalPage(nextpage.physical,
                                replacePageNum);
        page.inMemTime = 0;
        page.lastTouchTime = 0;
        page.R = 0;
        page.M = 0;
        page.physical = -1;
    }
```

Similarly, the *replacePageOPT* method handles page faults under the Optimal page replacement scheme. In the Optimal replacement algorithm, in order to find out the page candidate, there is a need to find out which page will be used in the latest future, and select that one for replace-ment. The idea is that by replacing the page that will be used in the longest time possible from the present, the minimum page fault cost will be in-curred. This is achieved by checking the next ac-cess number or time for all the pages in the memory, find out the page which will be accessed after the longest time among all the pages in the mem-ory, then page it out. As this access time is not available to operating systems in reality, the optimal replacement algorithm is not implementable. Yet it is used as a yardstick for the best theoretical page fault cost. The following shows the Java code for the *replacePageOPT* method.

```
public static void replacePageOPT(Kernel kernel, Vector
mem, int virtPageNum, int replacePageNum, ControlPanel
controlPanel, Vector instructVector, int runs) {
        int count = 0;
        int longestNextRun = -1;
        int longestNextRunPage = -1;
        int firstPage = -1;
        boolean mapped = false;
        Instruction inst = null;
        boolean hasPageAgain=false;

        while (!(mapped) || count != virtPageNum) {
            Page page = (Page) mem.elementAt(count);
            if (page.physical != -1) {
                if (firstPage == -1) {
                    firstPage = count;
                }
                hasPageAgain=false;
                for (int i = runs + 1; i < instructVec
                        tor.size(); i++) {
                    inst = (Instruction) instructVec
                            tor.elementAt(i);
                    if (inst.pageNum == page.id) {
                        hasPageAgain = true;
                        if (i > longestNextRun) {
                            longestNextRun = i;
                            longestNextRunPage =
                                    page.id;
                        }
                        mapped = true;
                        break;
                    }
                }
                if(!hasPageAgain){
                    longestNextRunPage = page.id;
                    break;
                }
            }
            count++;
            if (count == virtPageNum) {
                mapped = true;
            }
        }
if (longestNextRunPage == -1) {
            longestNextRunPage = firstPage;
        }
        Page page = (Page) mem.elementAt(
                        longestNextRunPage);
        Page nextpage = (Page) mem.elementAt(
                        replacePageNum);
        kernel.printPageFaultInfo(nextpage, page);

    controlPanel.removePhysicalPage(longestNextRunPage);
        nextpage.physical = page.physical;
        controlPanel.addPhysicalPage(nextpage.physical,
                            replacePageNum);

        page.inMemTime = 0;
        page.lastTouchTime = 0;
        page.R = 0;
        page.M = 0;
        page.physical = -1;
    }
}
```

The *replacePageFIFO* method is used to handle the FIFO algorithm, and as it is already implemented in the MOSS simulator code under the method name *replacePage*, we renamed the method *replacePage* to *replacePageFIFO* in order to distinguish it from the other algorithms. The FIFO algorithm, or first come first served, maintains a queue of the pages accessed with the head of the queue containing the id information for the oldest page that has been accessed in the longest time while the tail of the queue containing the id information of the most recent page that was accessed. Upon a page fault, the queue is popped with the id information for the oldest page that is selected by the FIFO algorithm to be the victim page.

The file *Instruction.java* was also modified to add a new attribute, *pageNum*, which represents the logical page number which will be read or written.

In the *Kernel.java* file, we added a new method to accommodate the selection of different algorithms as follows.

```
private void pageFault(int replacePageNum) {

        if (this.pageRepAlg.equalsIgnoreCase("LRU")) {
            PageFault.replacePageLRU(this, memVector,
                virtPageNum, replacePageNum,
                controlPanel);
        } else if (this.pageRepAlg.equalsIgnoreCase("OPT")) {
            PageFault.replacePageOPT(this, memVector,
                virtPageNum, replacePageNum,
                controlPanel, instructVector, runs);
        } else {
            PageFault.replacePageFIFO(this,
                memVector, virtPageNum,
                    replacePageNum, controlPanel);
        }
        pageFaultTimes++;
        controlPanel.pageFaultValueLabel.setText(
                "YES");
        controlPanel.pagefaultTimesValueLabel.setText(
                pageFaultTimes+"");
    }
```

3.3 Page Fault Calculation

Whenever a page fault occurs, we calculate the page fault cost based on whether the victim page is modified. We keep track of whether the page has been modified or not by adding the *page modified* flag which we set when a page write is executed. All these changes are made in the *kernel.java* file. Thus we set the *page modified* flag, page.M = 1, every time that a write instruction is executed on an item in the page.

In the *printPageInfo* method, we calculate the page fault cost based on whether the victim page is modified or not, resulting in the *pageFaultCost* to be either 2 or 1, respectively. The Java code for this method follows.

```
public void printPageFaultInfo(Page in, Page out) {
        if (out.M == 1) {
            pageFaultCost = 2;
            controlPanel.victimPageIsDirtyValue-
                Label.setText("YES");
        } else {
            pageFaultCost = 1;
            controlPanel.victimPageIsDirtyValue-
                Label.setText("NO");
        }
         controlPanel.victimPageValue-
                Label.setText(out.id+"");
        controlPanel.pageFaultCostValue-
                Label.setText(pageFaultCost + "");
        ………..
    }
```

The *pageFaultCost, victimPage, victim-PageIsDirty* variables are added into the MOSS memory simulator by changing the *ControlPanel.java* file, and these variables are set whenever a page fault occurs.

```
controlPanel.victimPageIsDirtyValue-
        Label.setText("YES");
controlPanel.victimPageValue-
        Label.setText(out.id+"");
controlPanel.pageFaultCostValue-
        Label.setText(pageFaultCost + "");
```

3.4 Code and Data Alignment

The alignment of code and data is implemented by defining the text and data page virtual page ranges in the configuration file. For example this can be added into the configuration *memory.conf* file as "text 0 10   data 11 63."

In the above, virtual pages 0 to 10 are read-only text pages, and virtual page numbers 11-63 contain data that can be read or written In the *Kernel.java* file, we set up the text and data range by adding the following attributes and values based on the values in the configuration file.

```
public int textPageLow = 0;
        public int textPageHigh = 0;
        public int dataPageLow = 0;
        public int dataPageHigh = 0;
             while ((line = in.readLine()) != null) {
               StringTokenizer st;
                          if (line.startsWith("text")) {
                    st = new StringTokenizer(line);
                    while (st.hasMoreTokens()) {
                            tmp = st.nextToken();
                            textPageLow = Common.s2i(
                                                st.nextToken());
                            textPageHigh = Common.s2i(
                                                st.nextToken());
                            if (textPageLow < 0 || textPageLow >
                            63|| textPageHigh < textPageLow
                                 || textPageHigh > 63) {
                            System.out.println( "MemoryManage-
                                    ment: text page out of bounds.");
                            System.exit(-1);
                            }

                    }
               } else if (line.startsWith("data")) {
                                  st = new StringTokenizer(line);
                    while (st.hasMoreTokens()) {
                            tmp = st.nextToken();
                            dataPageLow = Common.s2i(
                                                       st.nextToken());
                            dataPageHigh = Common.s2i(
                                                       st.nextToken());
                            if (dataPageLow < 0 || dataPageLow >
                            63|| dataPageHigh < dataPageLow
                                 || dataPageHigh > 63) {
                                 Sys-
             tem.out.println("MemoryManage
                                       ment: data page out of
                                       bounds.");
                                 System.exit(-1);
                            }

                    }
               }
```

The *R/O RangeViolation* attribute is added into the *ControlPanel.java* file to indicate whether the instruction violates the Read Only range, and takes the value YES only when an instruction is written into a text virtual page range which is read-only, otherwise it takes the value NO.

```
Label roRangeVioLabel = new Label("R/O RangeViolation: " ,
Label.LEFT) ;
```

In the *Kernel.java* file, we check whether the page range falls into the text page range for a write instruction, then set the *R/O RangeViolation* variable to YES.

```
if (instruct.inst.startsWith("WRITE")) {
  Page page = (Page) memVector.elementAt(
         Virtual2Physical.pageNum(
                instruct.addr, virtPageNum, block));
  if (page.id >= textPageLow && page.id <=
         textPageHigh) {

         if (doFileLog) {
                printLogFile("WRITE"+Long.toString(
                              instruct.addr, addressradix)
                              + " ... R/O Page");
         }
         controlPanel.roRangeVioValue
                              Label.setText("YES");
         runs++;
         return;
    }
  …}
```

4. Uses of the Simulator

The simulator requires a command file containing a sequence of READ and WRITE commands immediately followed by a virtual memory address. The graphical simulator can be used for

i. stepping through memory accesses and stopping when a page fault occurs, and stepping again to watch which page is chosen by the replacement algorithm for eviction. The mapping between virtual to physical pages is always visible allowing the apprentice to keep track of which virtual pages are currently present in physical memory and their locations in physical memory.

ii. The apprentice can change the command file e.g. changing order of READs and WRITEs to the same page and make other simulation runs.

iii. Alternatively, the apprentice can switch to another page replacement algorithm and rerun simulations with the same command file and collect performance data, in order to compare the operation of the different page replacement algorithms.

iv. Evaluation and performance comparison of the different page replacement algorithms can be made by analyzing the simulation statistics from each run.

5. Example of Use: Performance Comparison of Different Page Replacement Algorithms

Students made simulation runs with the three replacement algorithms (FIFO, Optimal and LRU) and calculated their page fault costs to compare their goodness. The page size was assumed to be 4096 Bytes. The number of virtual pages was capped at a maximum of 64 virtual pages. Virtual pages number 0-10 were assumed to contain read-only text, while virtual pages numbers 11-63 were assumed read/write data. The *memory.conf* configuration file has a number of *memset* operations which map virtual pages to physical pages and sets initial attributes of virtual pages including *Read, Modified, inMemTime* in nanoseconds, and *lastTouchTime* in nanoseconds. The physical page's number is referred to below as the frame number.

5.1 FIFO Replacement

To demonstrate Belady's anomaly when the FIFO algorithm is selected (MOSS' default algorithm), the students executed the same set of memory instructions on the MOSS simulator over and over again increasing each time the number of frames. They observed that when the number of frames is 4, the page fault cost is greater than the cost with only 3 frames. This is an unexpected re-sult, as normally the number of page faults should decrease as the number of frames increases. They concluded that since the FIFO page replacement algorithm features Belady's Anomaly, there is no way of predicting the page fault cost. In this way, FIFO is not a good page replacement algorithm.

5.2 Comparison of Replacement Algorithms

The students varied the number of frames and reran simulations with the same configuration files (same read/write traces) with the various page replacement algorithms and collected the page fault costs generated with each algorithm. They observed a large difference in page fault costs between numbers of frames in the range 2–6. As expected, they verified that the Optimal algorithm always results in the least page fault costs compared to the FIFO and LRU.

When the number of frames falls between 2 and 6, the students observed a clear distinction in the goodness of the replacement algorithms. In that range, the Optimal algorithm is still the best with the least page fault cost, while FIFO has the worst page fault cost. The LRU replacement algorithm performs very closely to the Optimal page replacement algorithm as the number of frames in physical memory increases within that range, with the smallest difference detected at 6 frames.

The students repeated these experiments with various configuration files which were randomly generated (randomly select READ or WRITE, and virtual memory address). Again, the Optimal and FIFO algorithms resulted in the best and worst costs, respectively. However the LRU costs are much closer to the FIFO costs than they are to the Optimal costs. The same findings were obtained after the students localized the virtual memory addresses to mimic real workloads where the same set of pages were hit again.

6. Conclusion

We modified the Java source code of the MOSS Memory Management simulator and integrated code for various page replacement algorithms and correspondingly updated the simulator's GUI to reflect these added choices. As a result, we have obtained a useful educational tool for operating system students. The resulting simulator allows the students to visualize the operation of various page replacement algorithms, step and stop the simulator for statistics reading, visualizing how full the page table is, watching the mappings from virtual pages to physical frames, and easily identifying which page is chosen for replacement. In addition the students enhanced their Java programming skills by coding the new algorithms and their integration skills by understanding exist-ing source code, and modifying it and adding the code, in order to modify the simulator to handle the new requirements.

The simulator allowed the students make various runs with pre-determined and random memory commands and collect page fault costs under the various page replacement algorithms, allowing them to evaluate and compare these various algorithms. The simulator can be further modified as described in this paper to allow the simulation of a wider variety of page replacement algorithms.

7. Acknowledgments

This work is based on the MOSS Memory Management Simulator developed by Alex Reeder. We acknowledge J. Weiss for the idea behind this work.

8. REFERENCES

[1] A. Tanenbaum, Page Replacement Algorithms, PHR http://www.phptr.com/articles/ article.asp?p=25260&rl=1

[2] MOSS Memory Management Simulator, http://www.ontko.com/moss/ memory/memory.exe

[3] R. Ontko. MOSS Memory Management Simulator User Guide,http://www.ontko.com/moss/ memory/user_guide.html

[4] A. Tanenbaum. Modern Operating Systems, Prentice Hall, 2nd edition, 2001.

[5] A. Silberchatz, P. Galvin, and P. Gagne. Operating Systems Concepts with Java, 7th Edition, J. Wiley & Sons, Inc., 2005.

[6] A. V. Aho, P. J. Denning, J. D. Ullman. Principles of Optimal Page Replacement. J. ACM, Jan. 1971, 18(1):80-93.

[7] Y. Smaragdakis. General Adaptive Re-placement Policies. In Proc. of 4th Int. Sym-posium on Memory Management, Vancouver, Canada, 2004, pp. 108-119.

[8] G. Glass, and P. Cao. Adaptive Page Re-placement Based on Memory Reference Be-havior. In Proc. ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems. Seattle, U.S.A. 1997, pp. 115-126.

[9] M. Ujaldon, S. Sharma, and J. Saltz. Page Replacement Using Marginal Loss Functions. In Proc. of ACM/IEEE Conf. on Supercomputting. San Jose, U.S.A. 1997. pp. 1-12.

[10] C. Waldspurger. Memory Resource Man-agement in VMware ESX Server. ACM SI-GOPS Operating Systems Review. Winter 2002, 36 (SI):181-194.

[11] Y. Joo et al. Demand Paging for One-NAND™ Flash eXecute-in-place. In Proc. of 4th Int. Conf. on Hardware/software codesign and system synthesis. Seoul, Korea, 2006, pp. 229-234.

A Novel Sorting Animation: Permuting Picture Pixels

John Edgar
School of Computing Science
Simon Fraser University
Surrey, BC V3T0A3
johnwill@sfu.ca

Toby Donaldson
School of Computing Science
Simon Fraser University
Surrey, BC V3T0A3
tjd@cs.sfu.ca

Abstract
Algorithm animation has a long history in CS education, and in this paper we describe a novel way to animate basic sorting routines. The idea is to scramble the pixels of a picture, and then use a sorting routine to unscramble them. The resulting animation of moving pixels is both enjoyable to watch, and provides enough clues to figure out what algorithm is doing the unscrambling. We have used this as a class activity in numerous data structures and algorithms courses to test students understanding of different sorting algorithms. In addition to describing how our animation technique works and is implemented we also discuss some of the related work in the field of algorithm visualization, and the importance of student engagement in such visualizations.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms
Algorithms

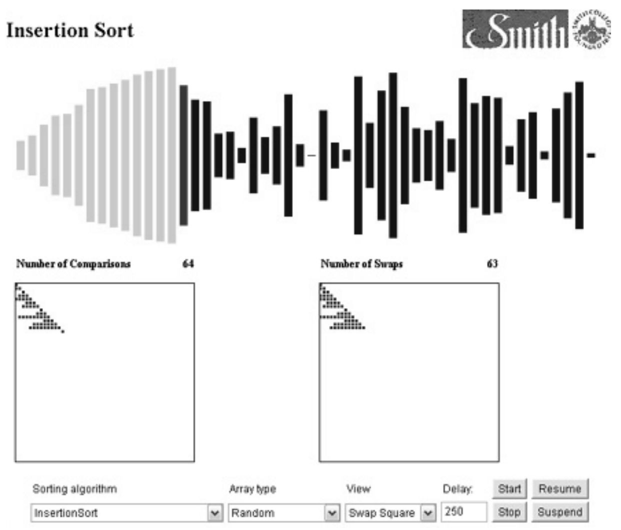
Keywords
Algorithm Visualization, Algorithm Animation, Computer Science Education.

1. Introduction
Computer science has a long history of animating algorithms for pedagogical purposes. The 1981 video[1] "Sorting Out Sorting" [1] was the first widely distributed visualization, and BALSA [2] was one of the first complete algorithm animation systems used in programming courses. Since then hundreds of algorithm visualizations have been implemented.

There are many different algorithm visualizations and animations. For instance, Algoviz Wiki [3] links to nearly 400 algorithm visualizations. One of its goals is to more widely distribute algorithm visualizations and to inform developers about what makes a good visualization [4]. Sorting algorithms, in particular, have been animated in many different ways: AlgoViz alone lists 115 different sorting visualizations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

The usual way of animating a sorting algorithm is to by swapping the bars of a bar graph. A typical example is shown in Figure 1. The bars are initially scrambled, and then re-ordered swap by swap. Each individual swap is animated, and different sorting algorithms have distinctive movement patterns.



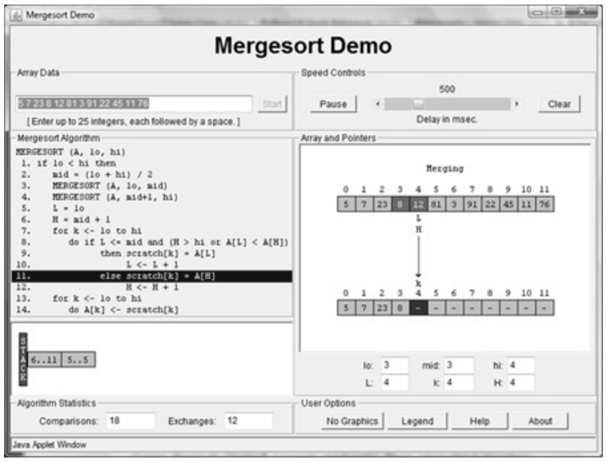
Number of Comparisons 64
Number of Swaps 63
Sorting algorithm: InsertionSort
Array type: Random
View: Swap Square
Delay: 250

Figure 1. A Bar Sorting Visualization[5].

Figure 2 shows an example of another common sorting animation. This animation shows the algorithm progressing from line to line in one part of its window and displays the corresponding changes to the array being sorted in a second part of its window.

Continuing in this tradition, we have developed a novel and fun classroom activity that tests students understanding of comparison-based sorting algorithms. We take ordinary pictures, scramble their pixels, and then restore them by sorting. An example of a scrambled image and an image that has been approximately 60% sorted using mergesort are shown in Figure 3.

[1] The entire movie is now available on Google Video: http://video.google.com/videosearch?q=sorting+out+sorting&emb=0&aq=f#



5 7 23 8 12 81 3 91 22 45 11 76

```
MERGESORT (A, lo, hi)
 1. if lo < hi then
 2.    mid = (lo + hi) / 2
 3.    MERGESORT (A, lo, mid)
 4.    MERGESORT (A, mid+1, hi)
 5.    L = lo
 6.    H = mid + 1
 7.    for k <- lo to hi
 8.       do if L <= mid and (H > hi or A[L] < A[H])
 9.             then scratch[k] = A[L]
10.                  L <- L + 1
11.             else scratch[k] = A[H]
12.                  H <- H + 1
13.    for k <- lo to hi
14.       do A[k] <- scratch[k]
```

0	1	2	3	4	5	6	7	8	9	10	11
5	7	23	8	12	81	3	91	22	45	11	76

0	1	2	3	4	5	6	7	8	9	10	11
5	7	23	8	-	-	-	-	-	-	-	-

lo: 3 mid: 3 hi: 4
L: 4 k: 4 H: 4

Comparisons: 18 Exchanges: 12

Figure 2. A Typical Sorting Animation[6].

Each sorting algorithm re-orders the pixels in a visually distinctive way that makes it possible to determine how, approximately, it works. The general pattern of how pixels move is typically quite clear, and so someone who knows how the algorithms are implemented can usually figure out which one is running. We have used this technique to illustrate comparison-based sorting algorithms including selection sort, insertion sort, bubble sort, mergesort, heapsort, and quicksort.

In what follows, we will discuss the underlying idea of how we scramble and restore images, discuss some of the implementation details, survey related work, and finally propose some ideas for future research.

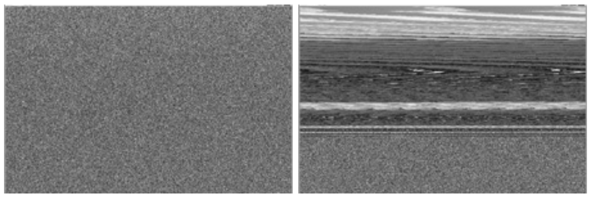


Figure 3. Scrambled and Partially Sorted Image.

2. Sorting Images

2.1 Description

Our approach is quite different from traditional sorting animations as we sort the pixels of an image instead of the bars of a graph. Any image can be used, and watching a picture gradually being revealed is part of what makes this approach interesting.

Our software starts by displaying the image with its pixels completely scrambled; the result is typically a gray fog (as in the first image of Figure 3). Then the sorting algorithm starts to move the pixels, re-displaying the image after each move. Just as with the bar graph animations, the patterns of pixel movements are enough, with some thought, to identify the particular sorting algorithm.

Each sorting algorithm has its own pattern of pixel movement. Figure 4 shows the various stages of each algorithm we have implemented.

Next we discuss the individual sorts in more detail.

2.2 Selection Sort

The selection sort animation reveals the image one pixel at a time. It goes row by row starting from the top left corner. As the sort nears completion the viewer gets the impression that the pixels are moving faster. Indeed this is the case because selection sort does less work as it nears the end. We raise this point in the class activity (Section 4) because it ties directly to the implementation of selection sort (which the students have already seen).

Another interesting detail is that this animation runs much faster than the other two quadratic sorts (discussed next) since it does fewer swaps. This matters because our software only animates swaps and does not spend any time animating comparisons.

2.3 Insertion Sort

Insertion sort’s animation is particularly interesting – and particularly slow. Like selection it works from top to bottom, but the partial images are very different. Related pixels flow together like a liquid to form diagonal lines. Slowly, the lines morph into multiple distorted versions of the final image; this can be seen clearly in the third image of insertion sort in Figure 4. The final few seconds of the animation are quite appealing; the entire image suddenly swings into place as if it were being un-stretched.

In contrast to selection sort, this animation slows down as more of the image is revealed. The reason for this is that inserting a single pixel may involve re-positioning all the pixels before it.

Each of these points makes for interesting discussions when this is presented as a class activity.

2.4 Bubble Sort

Bubble sort is just as slow as insertion sort. Like selection sort it speeds up as more of the image is revealed. The image is revealed slowly from the bottom up one pixel at a time. True to its name pixels can be seen bubbling across the image. Furthermore all of the pixels are constantly marching across the screen, with an almost organic feel.

We think the distinction our algorithm makes between swaps and comparisons is useful. In practice, the time it takes to swap items and compare items can be quite different in an object-oriented language. For example, in Java all objects are referred to via references, and so swapping is a fast, constant-time operation no matter the size or complexity of the underlying object. However, comparing objects can be slow because it involves the more expensive operation of calling (possibly virtual) comparison methods, which in some cases (e.g. strings) can depend upon the size or complexity of the objects being compared.

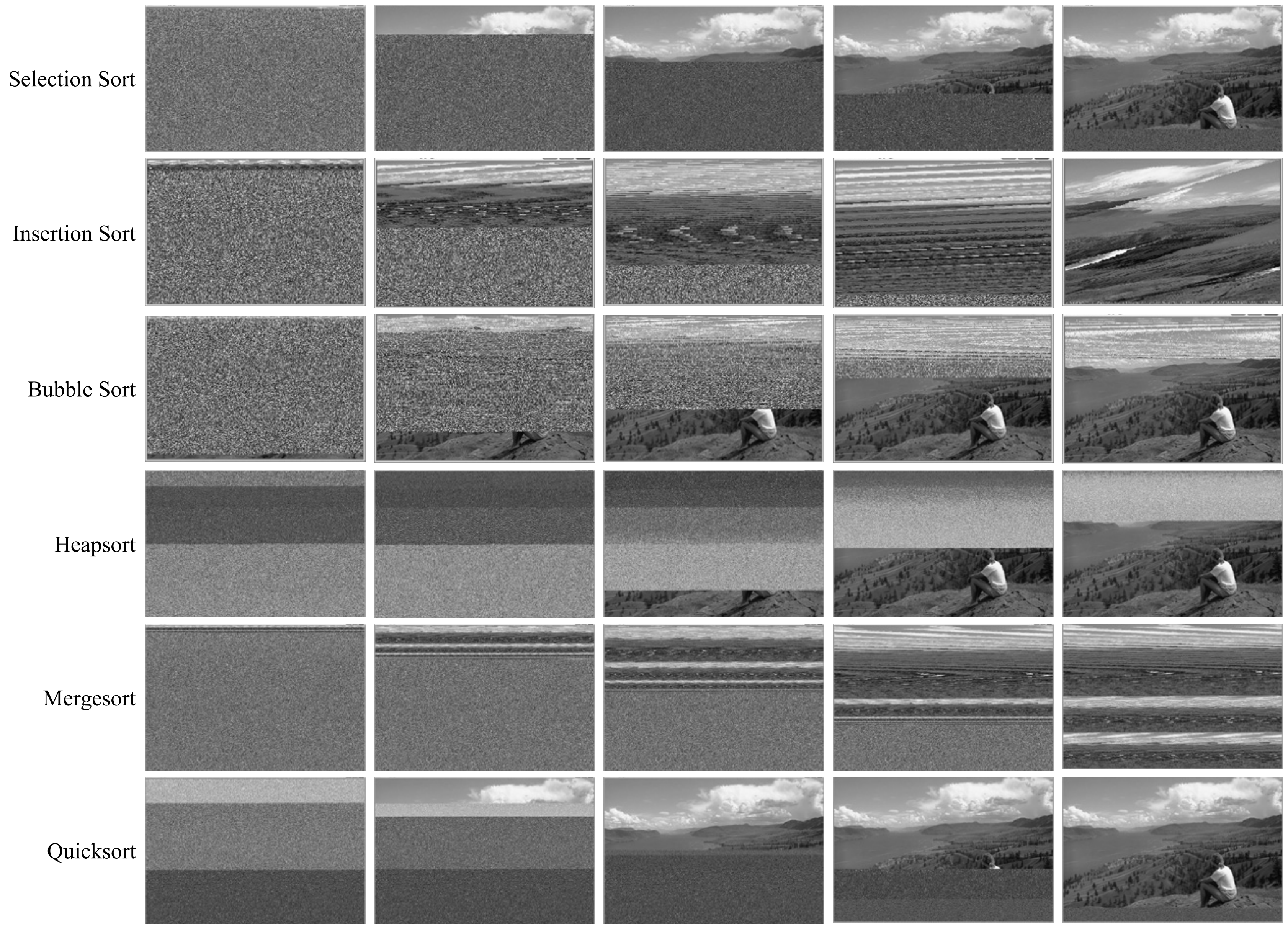


Figure 4. Image Visualizations for Different Sorting Algorithms. Progress is shown from left to right; the first column shows images shortly after starting sorting and the last column just before the image is fully sorted. The other three central columns show intermediate stages of the sort.

2.5 Heapsort

The heapsort animation has two distinct stages. In the first stage, which is over very quickly, the image is divided into bands, with the colours that predominate in the lower part of the final image appearing at the top of the image, and vice versa. This corresponds to the heapification of the underlying array. In the second stage the image is revealed from the bottom to the top (pixel by pixel). Like selection sort, the revealed part of the image is perfect but revealing it happens much faster. Given that heapsort is an O(n log n) algorithm this what we would expect.

In class we discuss that heapification is so fast because it is an O(n) process while removing all of the elements from the heap takes O(n log n) time.

2.6 Mergesort

Compared to the other algorithms the mergesort animation is very regimented. First the pixels in a thin strip at the top of the image are rearranged. Then another strip of pixels immediately below are rearranged in the same way. Then the two strips are combined into a wider strip. This pattern is repeated until the entire top half of the image has been rearranged (but not revealed). The bottom half of the image then undergoes the same treatment. Finally the two halves are merged together to reveal the entire image.

The animation is somewhat similar to the insertion sort animation. On reflection this makes sense because both algorithms rearrange pixels in a similar way.

The mergesort animation has another interesting aspect when compared to heapsort and quicksort. As was noted earlier, the image is initially scrambled by randomizing the positions of pixels. This means that when an animation is run multiple times on the same image the pixels begin in different positions. For heapsort and quicksort this difference is very noticeable: for instance the bands of colours that move through the animations are different each time. In contrast, the mergesort animation looks almost identical every time it is run. In class we often run these three animations multiple times to demonstrate this.

2.7 Quicksort

In the quicksort animation the image is repeatedly divided into bands of colour, with each band corresponding to the colours that predominate in that section of the original image. As bands are created, families of pixels of the same colour appear to fall down the image. Once these heavy pixels have fallen away the band is replaced by the original image from top to bottom. The process then repeats with the next band.

In contrast to the mergesort animation, the width of these bands varies each time the algorithm is run. For example after the first partition the image is divided into two bands, each of which might be roughly the same height, or one much larger than the other. Quicksort partitions pixels around a pivot pixel, which is determined randomly, so each time the algorithm is run the animation is different.

Quicksort has a dark side. Unlike Mergesort and Heapsort it is quadratic when the pixels are already ordered (forwards or backwards). We note in class that this has never happened for hundreds of runs of the animation. This is because our images are made up of hundreds of thousands of pixels and the chances of them being randomly shuffled into order are vanishingly small. This notion is further discussed in Section 5.

3. Implementation

Our implementation of the pixel-sorting animation is quite general, allowing us to visualize almost any comparison-based sort.

Essentially, the 2-dimensional image of pixels is mapped into a linear array of integers called the **picture array**. The pixel at position (x, y) of the image is mapped to index location x*WIDTH + y of the picture array. Given an index location in a picture array, we can recover the corresponding (x, y) position using the mod and div functions.

Initially the values of the picture array are set to be the same as their index locations, indicating that the pixels of the image are in their original order. To scramble the image, we randomly shuffle the picture array changing the corresponding image pixels as we go. We don't usually animate the scrambling.

To unscramble the image, we simply call a sorting function on the picture array. Every time a pair of items is swapped in the picture array the corresponding pixels in the image are immediately swapped.

It's relatively easy to modify almost any existing comparison-based sorting algorithm to sort pixels. Essentially, all that is needed is to pass in the picture array, and call the picture array swap function (which is tied to the animation code).

Our software is implemented in Java [2], and the graphical display is based on a wrapper class that enhances the standard `BufferedImage` class that comes in the Java library. It works with most image formats, such as .jpg, .gif, and .png.

The sorting algorithms sort ImageMap objects. ImageMap objects contain a TheMatrix object and an integer array which represents the picture array. We use the TheMatrix class to conveniently move pixel and load and save images in a BufferedImage.

The performance is usually quite good. The mergesort, quicksort and heapsort animation all complete in just a few seconds even on images that contain hundreds of thousands of pixels. Naturally the quadratic algorithms (selection sort, insertion sort and bubble sort) are much slower.

4. Class Activities

We've developed a 60-minute in-class activity based on pixel-sorting. We run each sorting algorithm on the same image, but

[2] The software is available at :
http://csed.googlecode.com/files/imageSortingWccce2009.tar.gz

keep the algorithms secret. After each demonstration we discuss what we have seen, and ask the students which algorithm was being used, and why. We also discuss the time it takes to recreate an image, as the differences in running times can be dramatic.

Most students appear engaged in the activity, and some of them have mentioned that it improves their understanding of the sorting algorithms. Students appear to enjoy the activity and to be making an effort to analyze each of the algorithms: they typically ask questions and make positive comments. We've noticed that even students who are usually reserved seem more at ease when discussing this activity. We speculate this is because they view this as a fun puzzle to be solved rather than an assessed test of their knowledge of the course.

The question of the pedagogical effectiveness of algorithm animations is still being debated. Many people have strong intuitions that algorithm animations ought to be better than traditional textbook descriptions, but a number of studies have shown otherwise [4]. How actively students are involved with a visualization has been identified as being an important factor in the effectiveness of the activity. Often this has been found to be more important than the features (or quality) of the visualization technique [7, 8].

5. Conclusions

We've been using these animations for a number of years and feel that our class activity has been beneficial to students.

The software could be improved and extended in a number of obvious ways. Currently, only pixel swaps are visualized. It might be useful to also animate comparisons, e.g. by highlighting the compared pixels in red. This is not as easy as it might sound at first: so many comparisons happen so quickly that it could be difficult to distinguish the red pixels from all the others. Some other visualization technique is probably needed for showing comparisons.

Many bells and whistle could be added to the GUI. Currently, we run the software from the command-line or an IDE, and set the algorithm to be visualized directly in the code. Being able to run and configure it from a stand-alone GUI would certainly be more convenient, as we would be able to more easily select different images, switch algorithms and so on. We would also like to allow the animation to be paused, slowed-down, reversed, have snapshots taken, statistics recorded, etc.

It would also be interesting to empirically evaluate the effect of the class activity in improving student understanding or engagement. However, our suspicion is that the results would not differ significantly from those reported in [4].

In addition the class activity could be extended in a number of ways. For instance we could show the effect (if any) on the sorting algorithms of starting with partially sorted images instead of totally scrambled ones. We, or, even better, students, could implement other comparison based sorting algorithms such as shaker sort, shellsort or treesort. We could also experiment using the same sort algorithms on different images, such as ones with less colours or with interesting shapes. Finally, it might be interesting to show the animations to students *before* they see the source code for the algorithms. This way they might have a better understanding of the general sorting strategies.

6. Acknowledgements

Thank you to all the students who participated in this activity and provided helpful feedback.

And also thanks go to Judy who unknowingly posed for the photograph used in this paper and has kindly allowed us to scramble and sort her again and again (and again).

7. References

[1] R. Baecker, D. Sherman, and U. of Toronto Computer Systems Research Group. Dynamic Graphics Project. Sorting Out Sorting. 1981.

[2] Brown, M.H., Sedgewick R. (1984). "A System for Algorithm Animation" Computer Graphics, July 1984, pp.177-186.

[3] Structure and Algorithm Visualization, Algoviz Wiki, http://algoviz.org/AlgovizWiki/, 2009

[4] Shaffer C.A., Cooper M., Edwards S.H., "Algorithm visualization: a report on the state of the field", *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, New York, NY, USA, ACM, pp. 150–154, 2007

[5] Biliana Kaneva, Dominique Thiébaut, http://maven.smith.edu/~thiebaut/java/sort/demo.html, 2009

[6] Brian S. Borowski, http://www.brian-borowski.com/Sorting/

[7] S. Grissom, M. McNally & T. Naps, "Algorithm Visualization in CS Education: Comparing Levels of Student Engagement," *ACM Symposium on Software Visualization*, New York: ACM, pp. 87-94. Distinguished Paper Award. 2003.

[8] Thomas L. Naps, Guido Rößling, Vicki L. Almstrum, Wanda Dann, Rudolf Fleischer, Christopher D. Hundhausen, Ari Korhonen, Lauri Malmi, Myles F. McNally, Susan H. Rodger, J. Ángel Velázquez-Iturbide: "Exploring the role of visualization and engagement in computer science education". SIGCSE Bulletin 35(2): 131-152 (2003)

Simplifying Algorithm Learning Using Serious Games

Sahar Shabanah
George Mason University
4400 University Dr.
Fairfax, VA 22033, USA
sshaban1@gmu.edu

Dr. Jim X. Chen
George Mason University
4400 University Dr.
Fairfax, VA 22033,USA
jchen@gmu.edu

ABSTRACT

Algorithm Visualization using Serious Games (AVuSG) is an algorithm learning and visualization approach that uses serious computer games to teach algorithms. It visualizes an algorithm to be learned in four forms: a text, a flowchart, a game demo, and a game. Moreover, (AVuSG) method integrates learning theories and models in addition to motivation theory to introduce three learning models that simplify algorithm learning.

Categories and Subject Descriptors

L.5.1 [**SIMULATION/GAMES**]: Game Based Learning; I.6.8 [**Gaming**]: Simulation and Modeling

Keywords

Serious games, algorithm visualization, algorithm learning

General Terms

Learning Theories, Game Engine, Algorithm Visualization

1. INTRODUCTION

Data structures and algorithms are important foundation topics in computer science education. Students deal with algorithms in many computer science courses. For instance, in computer graphics, students learn objects rendering algorithms, in networking, they study algorithms that solve networks traffic congestion, and in database, they learn algorithms that search or sort data. Accordingly, teaching algorithms is a common activity that takes place in many computer science classes. However, algorithms are often hard to understand because they usually model complicated concepts, refer to abstract mathematical notions, describe complex dynamic changes in data structures, or solve relatively difficult problems. Consequently, teaching algorithms is a challenging task that faces instructors and requires a lot of explaining and illustrating. Therefore, teaching aids other than chalkboard and viewgraph are needed to help students understand algorithms better [1]. The human ability to realize graphic representations faster than textual representations led to the idea of using graphical artifacts to describe the behavior of algorithms to learners that have been identified as algorithm visualization [9].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1-2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

Motivation. Educational Computer Games are computer games that involve learning of certain knowledge [31]. Despite the frustration with educational computer games in the past, they reemerged recently as Serious Games. Serious games are computer games that have been developed for serious purposes other than entertainment such as training, advertising, simulation, and education. Since 2001, there have been several researches related to serious games such as the *Games To Teach* [19] and *The Making Games* [25] projects. The following features of computer games qualify them to be used for education in general and for algorithm learning in particular:

- **Computer games are popular.** Computer games have been broadly played all over the world by adolescents and young adults. In particular, the number of hours the standard college students spend on reading is half the time that they spend on playing computer games [26]. Therefore, the best method for teaching today students is to use computer games.

- **Computer games are interactive.** Computer games fully interact with players and encourage them to think and act. Thus, the use of computer games will improve algorithm learning since there is an increasing correlation between algorithm learning and the level of students' engagement [13].

- **Computer games are competitive.** A computer game player spends many hours in learning and playing the game. The motivation of the player to spend all that time with the computer game aroused from the game itself (Intrinsic Motivation) and not from an external incentive. Intrinsic Motivation improves learning as research shows [21].

- **Computer games simplify assessment.** To win a computer game, the player must understand its rules very well. Therefore, students understanding of the algorithms can be assessed using the winning/losing criteria of computer games that simulate the behavior of those algorithms.

- **Computer games utilize entertainment.** The use of computer games converts the unpleasant and tedious operation of algorithm learning into an enjoyable and interesting experience.

This paper introduces a new algorithm visualization method, namely Algorithm Visualization using Serious Games (AVuSG) that uses learning theories and serious computer games to simplify algorithm learning. The main part of this paper is divided into three sections. Section 2 describes related work. Section 3 explains the conceptual framework of (AVuSG) method, section 4 describes the (AVuSG) systems design, section 5 illustrates the algorithm game design, and section 6 describes three algorithm games prototypes.

2. RELATED WORK

SOS (1981) video was the first recognized algorithm visualization system. It uses animation, color, and sound to explain three sorting algorithms: insertion, exchange, and selection sorts [2]. However, BALSA (1984) was the first real-time, interactive algorithm visualization system, it allows the user to start, stop, or even run an algorithm backward [5]. Algorithm Explorer (2007) is an algorithm visualization system that uses three-dimensional objects to represent common data structures and animations to visualize algorithms. Its user interface allows the user to control how the visualization played and displayed on the screen [7]. SOS allows learners to view the visualization passively while BALSA and Algorithm Explorer provide the user with some kind of control over the visualization viewing process. Viewing is the most form of engagement that existing algorithm visualization systems provide for their users, other less supported engagement forms are responding, changing, constructing, and presenting [23]. MatrixPro (2004) is a visualization system that supports responding by asking viewers to answer questions related to the presented visualization. It uses a server called Trakla2 (2003) to automatically generate algorithm exercises and assess students returned answers [20]. JHAVE (2005) adds a responding support for a variety of algorithm visualization systems by providing stop-and-think questions [22]. Some algorithm visualization systems allow their viewers to change the visualization data or some other features. For example, GeoWin (2002) is a visualization system that allows the user to manipulate a set of actual geometric objects through the interactive interface [3]. Other system is CATAI (2002), which enables the user to invoke some methods on the running algorithm. In terms of method invocations, it is possible directly to access the content of the data structures or to execute a piece of code encapsulated in an ordinary method call [11]. ANIMAL (2002) supports constructing by enabling users to visualize algorithms through the composition of low-level drawings and the animation operations, but not on the abstract data types represented in the animation [27]. ALVIS (2004) is an interactive environment that enables students to quickly construct rough, unpolished (low fidelity) visualizations, and interactively present those visualizations to an audience. Its successor is ALVIS-Live! (2005) that reevaluates an algorithm line on every edit to update its visualization dynamically [18]. A project called Algorithm Studio (2004) supports presenting by allowing students to present their visualizations to their instructors in one-to-one sessions then to the entire studio. Finally, at the end of the semester, the students must present their solutions for an extra set of design problems to a jury of instructors [16].

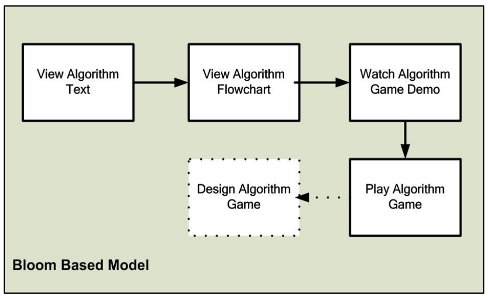


Figure 1: Bloom Based Model

Some researchers found that there is no significant difference in educational outcomes between students who use visualizations and those who do not [6] [17]. Moreover, in a recent effort to build a wiki for existing algorithm visualization systems, Shaffer *et al.* searched and analyzed hundreds of visualization systems and found that "most existing algorithm visualization systems are of low quality, and the content coverage is skewed heavily toward easier topics [29]."However, researchers remain positive about visualizations in general as Shaffer *et al.* states "while many good algorithm visualization systems are available, the need for more and higher quality visualization systems continues. There are many topics for which no satisfactory visualization systems are available. Yet, there seems to be less activity in terms of creating new visualization systems now than at any time within the past ten years [29]."The focus on graphics and sound instead of teaching aspects in the design of many algorithm visualization systems is responsible for their failing to be effective in teaching algorithms [30]. Other reason is the lack of features that encourage students' engagement with the displayed visualization [28]. Nevertheless, researchers identified the level of students' engagement as the most effective factor in the success of any algorithm visualization system since there is an increasing correlation between learning and the level of students' involvement with the visualization [13].

Algorithm Visualization using Serious Games (AVuSG) addresses the shortness in current algorithm visualization techniques by integrating learning theories and models in algorithm learning. Moreover, by visualizing algorithms as computer games, (AVuSG) introduces "playing"as a new engagement form that maximally engages students in the learning process.

3. (AVUSG) CONCEPTUAL FRAMEWORK

(AVuSG) framework consists of three parts: the Algorithm Representation Forms, the Learning Processes, and the Learning Models.

3.1 Algorithm Representation Forms

(AVuSG) produces four forms of representations or visualizations for the algorithm to be learned:

1. **The Algorithm Text:** a description of the algorithm steps that shows its basic idea and how it works.
2. **The Algorithm Flowchart:** a graph depicting the static visualization of the algorithm functionality.

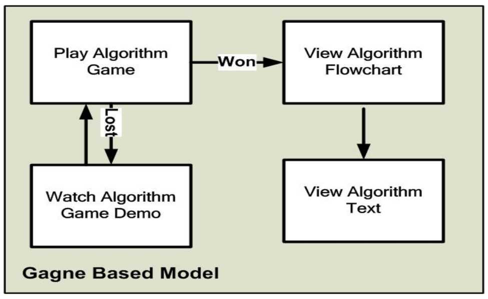


Figure 2: Gagne Based Model

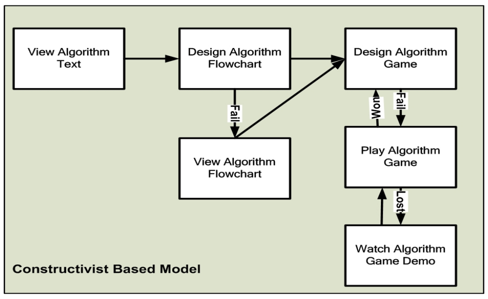


Figure 3: Constructivist Model

3. **The Algorithm Game Demonstration:** a dynamic visualization of the algorithm operations using the self-running demo of the algorithm game.
4. **The Algorithm Game:** an educational computer game with a game-play that simulates the behavior of the algorithm and graphics depict the features of its data structure.

3.2 Learning Processes

(AVuSG) defines three learning processes for learners to engage with any produced algorithm representation form:

1. **The Viewing Process:** in this process, the learner views the algorithm text, flowchart, and game demo.
2. **The Playing Process:** in this process, the learner plays the algorithm game.
3. **The Designing Process:** in this process, the learner develops the algorithm text, flowchart, game, and its demo.

3.3 Learning Models

(AVuSG) introduces three learning models that can be adopted either by students to learn the algorithm or by the instructors to teach the algorithm depending on the learning objectives that they want to achieve.

3.3.1 The Bloom Based Model

By applying this model (Figure 1), the learning objectives of Bloom Taxonomy (A.2.1) can be achieved as follows:

1. The learning starts in the viewing process, where learners watch the algorithm text to build their knowledge
2. Then, learners view the algorithm flowchart and game demo to comprehend how the algorithm works.
3. Next, in the playing process, learners apply their knowledge of the algorithm to play the algorithm game.
4. The game playing is divided into several game moves; each move simulates one algorithm step. During game playing, learners first analyze the algorithm into its steps to win each game move then synthesize all moves to win the whole game.
5. Learners can easily evaluate the speed, complexity, and efficiency of an algorithm by playing its related game. For example, if the game is hard to win, this means the algorithm is complex. Also, if the game data is not large this means the algorithm is suitable for small size data and so on.
6. As an optional and for more learning outcome. In the design process, learners analyze the algorithm and its played game to synthesize a new algorithm game design. Then, learners evaluate their designs for the new algorithm game.

3.3.2 The Gagne Based Model

The steps of this model (Figure 2) simulates the events of Gagne model of instruction (A.2.2) as follows:

1. The learning starts by playing the algorithm game that was created by the instructor, to gain the learners attention and activate their motivation. If lost the game, learners are provided with guidance through algorithm game demo. The playing process builds and stimulates the required prerequisite information related to the algorithm.
2. After winning the game, the learners are presented with the algorithm text and flowchart in the viewing process to learn the algorithm.
3. Eliciting and assessing the performance of learners can be achieved using the winning/losing criteria of the algorithm game.
4. Promoting retention and transfer can be acquired in the playing process.

3.3.3 The Constructivist Model

A description of this model (Figure 3) is as follows:

1. Learners start learning by viewing the algorithm text to have an idea about how the algorithm works.
2. Then, learners go to the designing process to design the flowchart of the algorithm.
3. If succeeded in building the flowchart, learners can start to design and develop their own algorithm games.
4. However, if learners fail any of these processes, they always can watch the algorithm flowchart, game demo, and play the algorithm game that has been built by their instructors.

The model deploys the constructivist theory (A.1.3) to affect learning in three ways:

1. Curriculum: for the same algorithm, each student builds his own algorithm game and flowchart using his prior knowledge.

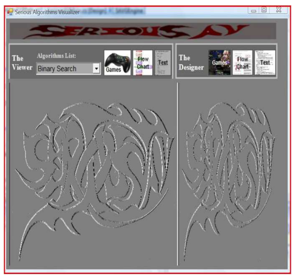
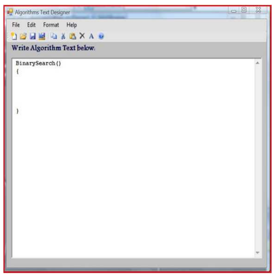


Figure 4: Serious-AV Main



Write Algorithm Text below:

```
BinarySearch()
{


}
```

Figure 5: Algorithm Text Designer

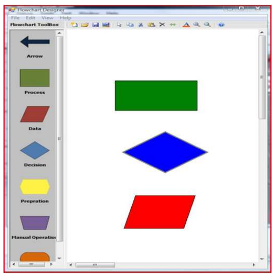


Figure 6: Algorithm Flowchart Designer

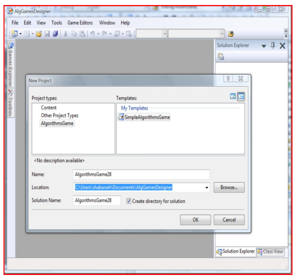


Figure 7: Algorithm Game Designer

2. Instruction: students analyze, interpret, and predict information about the algorithm to design its flowchart and game.

3. Assessment: students learn the algorithm by playing its game and judge their own progress through the winning/losing criteria of the game. Therefore, the assessment becomes part of the learning process.

4. (AVUSG) SOFTWARE SYSTEMS

Serious Algorithm Game Visualizer (Serious-AV) (Figure 4) is a visualization system that has been designed and developed to demonstrate (AVuSG) framework by providing several viewers and designers.

4.1 (Serious-AV) Viewers

(Serious-AV) supports the viewing and the playing learning processes by providing three viewers to enable the learner to view the algorithm text, flowchart, game demo, and game.

4.1.1 The Algorithm Text Viewer

A Windows Form that shows the algorithm text to the learner. It provides the learner with many options to manipulate the algorithm text such as copy, cut, paste, and print features.

4.1.2 The Algorithm Flowchart Viewer

A Windows Form that presents the algorithm flowchart to the learner. It provides the learner with many options to manipulate the algorithm flowchart such as copy, cut, paste, and print features.

4.1.3 The Algorithm Game Viewer

An XNA Window that displays the algorithm game to the learner to play the previously built game or watch its self-running demo.

4.2 (Serious-AV) Designers

(Serious-AV) supports the designing process by providing three types of designers for designing each form of the algorithm forms.

4.2.1 The Algorithm Text Designer

A Windows Form Application that simplifies the creation of the algorithm text by providing: a Text Editor to write the algorithm text, a File menu to save, open, and export the algorithm text to the Algorithm Text Viewer, and a Format menu to format the font; an Editing menu to cut, copy, paste, and delete the text (Figure 5).

4.2.2 The Algorithm Flowchart Designer

A Windows Form Application that simplifies the creation of the algorithm flowchart by providing: a Graphics Editor with a flowchart toolbox that has a set of drag-and-drop flowchart shapes, a File Menu to save, open, and export the flowchart to the Algorithm Flowchart Viewer, and an Editing Menu to copy, paste, move, resize, and delete a flowchart shape (Figure 6).

4.2.3 The Algorithm Game Designer

A Visual Studio Shell (Isolated Mode) that simplifies the development of an algorithm game and its demo by providing the following components (Figure 7):

- Code Editor: the main editor to develop the algorithm game using XNA framework and C# language.

- Script Editor: an editor to write game-specific script code that features IntelliSense-like programmer assistance.

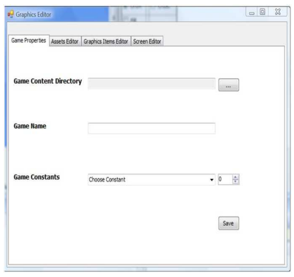
- Developer Graphics Editors: four editors that support the creation of different components of an algorithm game using a flexible and user-friendly graphical user interface. The Properties (Figure 8) and Assets Editors (Figure 9) are both Windows Form Applications to enter the game properties such as Number of Lives and to load the game assets such as fonts, textures, and models. The Graphics Items and Game Screens Editors are both XNA Applications to support the creation of game graphics items and the game screens.

- The Algorithm Game Template: a blueprint for the creation of a new algorithm game.

- Serious Algorithm Visualization Game Engine (SAVGEngine): an XNA library that provides the new algorithm game by many useful components such as Play and Base game classes to handle the timing, loading, and rendering operations; basic game operations modules such as graphics, sound, input, physics, script, and game screens managers; and a Repository that has ready-to-use algorithm game components to be plugged-in into the new game.

5. ALGORITHM GAME DESIGN

The algorithm game features 2D and 3D representations of common data structures, a game-play that simulates the visualized algorithm operations, and sound effects that reinforce the game events. Any algorithm that has specific



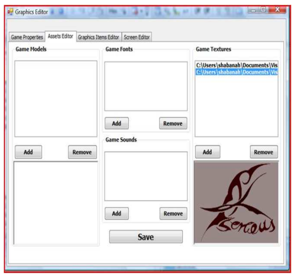


Figure 8: Properties Edit. Figure 9: Assets Editor

steps and a data structure can be visualized as an algorithm game. To create an algorithm game, the designer can either design a totally new game or modify the game-play of an existing game to simulate the algorithm steps. For example, the Binary Search Game (6.1) is created by modifying a known game called Pong to simulate the binary search algorithm. To simplify the development of algorithm games, for both the instructors and the students, they have been developed as XNA games. Since the Microsoft XNA Framework is a set of managed libraries based on the Microsoft .NET Framework that are designed for simplifying game creation for students and hobbyists. Moreover, the Algorithm Game Designer 4.2 has been designed to help in the design and the development of the algorithm games with as less code as possible.

5.1 Algorithm Game Motivation

Motivation theory (A.3) defines several guidelines that can be used to design algorithm games with internal motivation. First, the algorithm game should challenge the player by setting clear goals with appropriate difficulty levels and giving encouraging feedback. The information in the game should be complex and unknown to increase the player curiosity and imagination. The game must give the most control to the player by providing many customizing options. Moreover, it should encourage competition, collaboration and the recognition of peers.

5.2 Algorithm Game Design Elements

The following design elements [24] have been used to describe each algorithm game prototype:

- The Game Idea describes the game main goal and topic.
- The Game Start describes the game start up screen components.
- The Game Levels describes how the difficulty increases, how a level ends. Each completed level must achieve a learning sub-goal.
- The Game Milestone Events are points of the game at which the player rewarded or penalized.
- End of the Game explains what happens when the player loses, wins, or gets a high score.
- Game User Interface:
  - The Game Input is the player's contact with the game such as keyboard, mouse, and Xbox gamepad.
  - The Game Graphics are everything that contributes to the visual appearance of the game. The algorithm game graphics must depict the characteristics of its data structure, for example a block can be used to visualize one element of a data structure, while a set of blocks used to visualize an array.
  - The Game Sounds are either musical sounds that play at game goal events or sound effects that play at other game events.
  - The Game Screen is a collection of visual and audio components that describe the state of the game at any one time during the game life cycle. The basic game screens of every algorithm game are Title, Main Menu, Play, Won, and Lost screens.
- The Game Play explains how the game will be played and simulates the functionality of the algorithm.

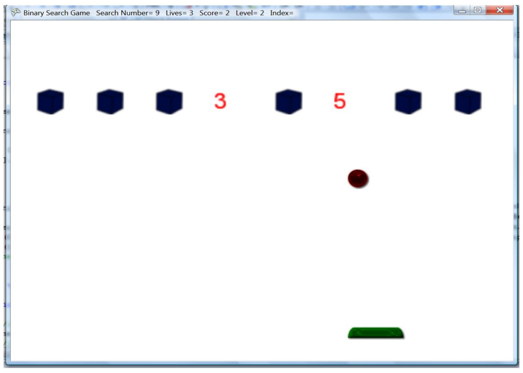

Binary Search Game Search Number= 9 Lives= 3 Score= 2 Level= 2 Index=
3
5

Figure 10: Binary Search- Play Screen

6. ALGORITHM GAME PROTOTYPES

To explain how an algorithm game can be constructed for a given algorithm, three algorithm games prototypes are described.

6.1 Binary Search Game Prototype

The binary search algorithm finds the index of a specific value in a sequential list of sorted elements (array) as follows: it selects the middle element (median) of a sorted list and compares it with the (target value),

- if (median) $>$ (target value), the middle element (index - 1) becomes the new upper bound of the list,
- else if (median) $<$ (target value), the middle element (index + 1) becomes the new lower bound,
- else if (median) $=$ (target value) then return the index of the middle element.

Then, it pursues this strategy iteratively for the new list bounded by the middle element; it reduces the search span by a factor of two each time, and soon finds the target value or else determines that it is not in the list.

6.1.1 Game Design Elements

• The main idea of the game is hitting an array of blocks with a ball using a paddle. The game is a mod of the Pong game.

• The game starts by displaying one game level that includes a group of blocks with their values hidden, a ball, a paddle, the Level Number, the Player Lives, and the Player Score.

• The game has several levels, at each new level the number of blocks is increased to make the game more challenging.

• The game ends when the player either loses all his lives or completes all the game levels successfully.

• The game milestone events are start of new level and a lost live.

• The game user interface includes game graphics items (array of cards, each card has a value to represent one element of the array, a ball, and a paddle), game sounds (HitBall, LostLive, Won, and Lost sounds), and game screens (Title, Main Menu, Play (Figure 10), Won, and Lost).

• The game play simulates the algorithm steps as follows:

1. The Player hits one block of the array with the ball using a paddle.
2. If the block is in the middle, the player scores one point.
   (a) If (search number> middle block value), the player plays on the right section of the array.
   (b) If (search number< middle block value), the player plays on the left section of the array.
   (c) If (search number== middle block value), the level ends.
3. If it is the last level and (Player Lives > zero), the player wins the game.
4. Else the level number is increased and the player starts new level.
5. If the block is not in the middle, the player loses one live.
6. If (Player Lives==0), the player loses the game.
7. Else the player repeats the same level.

6.2 Selection Sort Game Prototype

The Selection Sort algorithm sorts an array of numbers as follows: it finds the minimum value in the list, swaps it with the value in the first position, and repeats for remainder of the list (excluding the swapped elements at the beginning).

6.2.1 Game Design Elements

• The main idea of the game is sorting a group of dominoes in a fixed time according to the algorithm rules.

• The game starts by displaying one game level that includes a group of dominoes with their values shown, the Repeating Limit, the Level Number, the Player Score, and the Level Time.

• The game has several levels, at each new level the number of dominoes is increased, and the time is decreased to make the game more challenging.

• The game ends when either all the game levels are completed successfully or the Repeating Limit equals zero.

• The game milestone event is the start of new level.

• The game user interface includes graphics items (group of dominoes, each one has a value), game sounds (TimeEnds, Won, and Lost sounds), and game screens (Title, Main Menu, Play, Won, and Lost).

• The game play simulates the algorithm steps as follows:

1. The player chooses the smallest dominoes value and inserts it in its correct sorting place on the left.
2. If the player inserts the selected domino in incorrect place, the Player Score is decreased by one.
3. If the Level Time ends without sorting all the dominoes, the player loses the level, then:
   (a) If (Repeating Limit > zero), the player repeats the same level and the Repeating Limit is decreased by one.
   (b) Else the player loses the game.
4. If completes the level in the specified time, the player goes to the next level and the Player Score and the Level Number are increased.
5. If completes all levels on time, the player wins the game and the Player Score is displayed.

6.3 Insertion Sort Game Prototype

The Insertion Sort algorithm sorts an array of numbers as follows: it removes an element from the input data, inserts it into the correct position in the already-sorted list, until no input elements remain, and repeats for remainder of the list (excluding the elements in already-sorted list).

6.3.1 Game Design Elements

• The main idea of the game is sorting a pile of cards in a fixed time.

• The game starts by displaying one game level that includes a pile of unsorted, covered cards, the Player Lives, the Game Timer, the Repeating Limit, the Level Number, the Player Score, and the Level Time.

• The game has several levels, at each new level the number of cards is increased to make the game more challenging.

• The game ends when either all the game levels are completed successfully or the Player Lives equals zero.

• The Game milestone events are start of new level and a lost live.

• The game user interface includes game graphics items (array of cards, each card has a value), game sounds (LostLive, Won, and Lost sounds), and game screens (Title, Main Menu, Play, Won, and Lost).

• The game play simulates the algorithm steps as follows:

1. The player chooses one card at a time to be the key.
2. The player uncovers the chosen card to see its value.
3. The Player must compare the card with all the cards on the lift to insert it in its sorted place.
4. If inserts the card in incorrect place, the player loses one point.
5. If the Level Time ends without sorting all the cards, the player loses the level, then:

(a) If (Player Lives $>$ zero), the player loses one live and repeats the same level.

(b) Else the player loses the game.

6. If completes the level in the specified time, the player goes to the next level and the Player Score and the Level Number are increased.

7. If completes all levels on time, the player wins the game and the Player Score is displayed.

7. CONCLUSIONS AND FUTURE WORK

(AVuSG) is an algorithm learning and visualization method that uses serious computer games to visualize algorithms. It benefits from the players desire to win, love to compete, and entertaining resulted from playing games to motivate students learning algorithms. Moreover, it facilitates the students' assessment using the winning-losing criteria of computer games without the need for external questions. In addition, the approach integrates learning theories with game design to introduce three educational models, the instructors can deploy in their classes to teach students algorithms.

(Serious-AV) system implemented the (AVuSG) framework with a goal to be used by both instructors and students. The instructor uses (Serious-AV) designers to design a text, a flowchart, and a game for the algorithm under study. The students use (Serious-AV) viewers to view their instructor designs. Depending on the deployed learning model, students may also create their algorithm text, flowchart, and game designs. However, game design and development are not easy tasks, so more attention is given to the development of the Algorithm Game Designer. It has several components and editors to automate and simplify the game development. The implementation of most of these components have been completed except for the Script, Screens, and Graphics Items editors. The Script editor will allow designers to design their games using a script language such as Iron-Python or Lua, where the Screens and Graphics Items editors will allow for the graphical design of several game components. Currently, the (SAVGEngine) game engine supports the creation of 2D algorithm games, but it will be extended to support 3D games. In addition, the algorithm game repository will be updated with more game components that support the design of different algorithm games. Serious-AV has one implemented algorithm game, the Binary Search game, but more games will be added later.

When the proposed system is fully implemented, a user study for the (AVuSG) approach can be conducted. The students can be divided into four groups, one of the three learning models will be applied on each group, and the fourth group will use the traditional instruction methods.

8. REFERENCES

[1] R. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. In *Software Visualization: Programming as a Multimedia Experience*, pages 369–381. The MIT Press, 1998.

[2] R. Baecker and D. Sherman. Sorting out sorting. 30 minute colour sound film, Dynamic Graphics Project, University of Toronto, 1981. Excerpted and reprinted in SIGGRAPH Video Review 7,1983.

[3] M. Bäsken and S. Näher. Geowin - a generic tool for interactive visualization of geometric algorithms. In *Revised Lectures on Software Visualization, International Seminar*, pages 88–100. Springer-Verlag, 2002.

[4] S. B. Bloom. *Taxonomy of educational objectives.* Pearson Education, 1984.

[5] M. H. Brown and R. Sedgewick. A system for algorithm animation. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 177–186. ACM Press, 1984.

[6] M. D. Byrne, R. Catrambone, and J. T. Stasko. Do algorithm animations aid learning? Technical Report GIT-GVU-96-18, 1996.

[7] E. Carson, I. Parberry, and B. Jensen. Algorithm explorer: visualizing algorithms in a 3d multimedia environment. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education,Covington*, pages 155–159. ACM Press, 2007.

[8] K. Crawford. Vygotskian approaches to human development in the information era. *Educational Studies in Mathematics*, 31:43–62, 1996.

[9] P. D. Eades and K. Zhang, editors. *Software Visualization*, volume 7. World Scientific, 1996.

[10] R. S. Feldman. *Understanding Psychology.* McGraw-Hill, 1996.

[11] G. I. G. Cattaneo and U. Ferraro-Petrillo. Catai: Concurrent algorithms and data types animation over the internet. *Visual Languages and Computing*, 13(4):391–419, August 2002.

[12] R. Gagne, L. Briggsm, and W. Wager. *Principles of Instructional Design.* New York: Holt, Rinehart & Winston, 3rd edition, 1988.

[13] S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in cs education: comparing levels of student engagement. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization, San Diego*, pages 87–94. ACM Press, 2003.

[14] A. Hejdenberg. The psychology behind games. Gamasutra– Website, April 26, 2005. Accessed 10/10/2008, http://www.gamasutra.com/features/20050426/hejdenberg_01.shtml.

[15] W. Huitt and J. Humme. *Educational Psychology.* College of Education, 1999.

[16] C. Hundhausen. The "algorithms studio" project: using sketch-based visualization technology to construct and discuss visual representations of algorithms. In *HCC '02: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments, Arlington*, pages 99–100. IEEE Computer Society, 2002.

[17] C. Hundhausen, S. Douglas, and J. Stasko. A meta-study of algorithm visualization effectiveness. *Visual Languages and Computing*, 13(3):259–290, 2002.

[18] C. D. Hundhausen and J. L. Brown. What you see is what you code: A radically dynamic algorithm visualization development model for novice learners. In *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric*

Computing, pages 163–170. IEEE Computer Society, 2005.

[19] H. Jenkins. The games to teach project. Comparative Media Studies-MIT –Website, 2001. Accessed 03/10/2008, http://www.educationarcade.org/gtt/proto.html.

[20] A. Korhonen. *Visual Algorithm Simulation.* Doctoral dissertation (tech rep. no. tko-a40/03), Helsinki University of Technology, 2003.

[21] T. W. Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *SIGSMALL '80: Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems, Palo Alto*, pages 162–169. ACM Press, 1980.

[22] T. L. Naps. Jhave: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25(5):49–55, 2005.

[23] T. L. Naps, G. Rossling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. A. Velazquez-Iturbide. Exploring the role of visualization and engagement in computer science education. In *ITiCSE-WGR '02: Working group reports from ITiCSE on Innovation and technology in computer science education, Aarhus*, pages 131–152. ACM Press, 2002.

[24] M. Packard. A crash course in game design and production. Lord Generic Productions– Website, 1996-2001. Accessed 10/26/2008.

[25] C. Pelletier. The making of games project. London Knowledge Lab– Website. Accessed 03/17/2008, http://www.lkl.ac.uk/research/pelletier.html.

[26] J. M. Randel, B. A. Morris, C. D. Wetzel, and B. V. Whitehill. The effectiveness of games for educational purposes: a review of recent research. *Simul. Gaming*, 23(3):261–276, 1992.

[27] G. Rossling and B. Freisleben. Animal: A system for supporting multiple roles in algorithm animation. *Visual Languages and Computing*, 13(3):341–354, 2002.

[28] G. Rossling and T. L. Naps. A test-bed for pedagogical requirements in algorithm visualizations. In *ITiCSE '02: Proceedings of the 7th annual conference on Innovation and technology in computer science education, Aarhus*, pages 96–100. ACM Press, 2002.

[29] C. A. Shaffer, M. Cooper, and S. H. Edwards. Algorithm visualization: a report on the state of the field. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education, Covington*, pages 150–154. ACM Press, 2007.

[30] L. Stern, H. Sondergaard, and L. Naish. A strategy for managing content complexity in algorithm animation. In *ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, Cracow*, pages 127–130. ACM Press, 1999.

[31] M. J. Wolf. *The Medium of the Video Game.* University of Texas Press, 1st edition, 2002.

APPENDIX

A. THEORIES AND MODELS

A.1 Learning Theories

Learning theories provide a conceptual framework for interpreting the examples of learning.

A.1.1 Behaviorism

The learner responses to environment stimulation in ways that increase or decrease the likelihood of the same response in the future [15].

A.1.2 Cognitivism

Learning is a complex process that utilizes problem-solving, insightful thinking, and repetition of a stimulus-response chain [10].

A.1.3 Constructivism

Constructivism states that human beings actively construct knowledge for themselves through active interaction with the environment and previous experiences. Vygotsky's social development theory focuses on the connections between people and the sociocultural context [8]. Constructivism influences learning in three ways:

- Curriculum: using curricula customized to the students prior knowledge.
- Instruction: encouraging students to analyze, interpret, and predict information.
- Assessment: assessment becomes part of the learning process so that students judging their own progress.

A.2 Models of learning

Models of learning attempt to examine and organize all the elements that contribute to learning in a systematic way that can easily be applied to learning situations.

A.2.1 Bloom's Model

Bloom defined a hierarchy of six objectives for any learning process: 1) Knowledge: remembering previously learned material; 2) Comprehension: grasping the meaning of the material; 3) Application: using learned material in new situations; 4) Analysis: breaking down material into its component; 5) Synthesis: combining parts to form a new whole; 6) Evaluation: judging the value of material [4].

A.2.2 Gagne's Model

Gagne, Briggs, and Wager have derived nine events of instruction that can be applied during a learning process: 1) gaining attention, 2) activating motivation, 3) stimulating recall of prerequisite learning, 4) presenting stimulus material, 5) providing learning guidance, 6) eliciting the performance, 7) providing feedback, 8) assessing the learner's performance, 9) promoting retention and transfer [12].

A.3 Motivation Theory

Motivation theory is concerned with the factors that stimulate or inhibit the desire to engage in a behavior. Malone and Lepper [21] distinguish between two types of motivation: Extrinsic that is supported by factors external to the activity and Intrinsic that arises directly from doing the activity. Some of the factors that enhance the motivation of the learner are individual since they operate even when a learner is working alone: challenge, curiosity, control, and fantasy. Other interpersonal factors play a role only when someone else interacts with the learner: competition, cooperation, and recognition [14].

Why having in-person lectures when e-learning and podcasts are available?

Philippe J. Giabbanelli
School of Computing Science, Simon Fraser University
Burnaby, B.C., V5A 1S6, Canada
giabba@sfu.ca

ABSTRACT

Due to technological progress, students can now watch courses on their television using a professional multimedia courseware, or on their computer with podcasts broadcasting the lectures in real-time. As they do not have to spend time commuting and can pause a lecture to follow it at their own pace, one may think that they would stop attending classes. Furthermore, research has shown students only using online mediums performed in average as well as the ones opting for an in-person lecture. In this paper, we wish to foster a discussion on what is becoming tangible reality. We examine how to *combine* videos with more active class settings, suggesting several ideas taking into account not only technical but also human aspects. We conclude our exploration by advising a use of videos to distill skills while putting more emphasis in processes in the classroom: having students presenting their approaches on related problems would ensure mastering of the content from the videos as well as critical thinking that many courses often lack.

Categories and Subject Descriptors

K.3.1 [**Computers and education**]: Computer Uses in Education—*Computer-assisted instruction*

Keywords

Podcast, hybrid lecture, recordings, active learning

1. INTRODUCTION

Dragging oneself out of bed at 6:30 a.m. to attend an 8 a.m. lecture, after enduring an hour of commute, is rarely mentioned among their favorite activities by students. Similarly, students are sometimes quite mixed about the educational value of a lecture nearly three hours long. This is a long-lasting organizational problem: instructors and students have very disparate preferences regarding times, while the administration only has so many rooms. Then, a small miracle happened. Each student can *see* and *hear* the lecture at his convenience, and the same goes for the instructor who is by no means bound to the students. Furthermore, the instructor can stop for a little while, go have a cup of coffee, and the students can make him repeat as many times they want. This is the enchanting world of podcasts and multimedia courseware, in which the instructor records his lecture or performs in a studio, and the material is distributed to students. But all is not well in this world.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1-2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

When it comes to education, the main thing that might not go well is the quality of the teaching. This is classicaly assessed by means of tests, which show that students are at least as successful with recorded lectures than with in-person lectures [6]. One element for the explanation, beside not falling asleep because of inappropriate times, is that the attention span of an individual barely exceeds 20 minutes; in other words, a significant quantity of information is lost in a traditional lecture. But here comes the keyword: *traditionnal.* While its significance has a large range, with examples such as reading notes on a lectern in the arts or using PowerPoint in the sciences, it is clear that recorded lectures are a novelty of the late 20th century if not simply of the 21st.

In this paper, we analyze and suggest several approaches combining a parcimonious use of recordings with active teaching techniques peculiar to in-person lectures. This is carried on while keeping in mind two intrinsically linked aspects. Firstly, a *technical* aspect: while it appears that lectures based on readings are the most targeted ones for replacement by media-based approaches, not all settings can sustain such changes. Indeed, we know that teaching is not just a monologue but rather a conversation, in which the instructor can build up on students' reactions as in the Socratic Method or facilitate debates. Secondly, a *human* aspect. It can be rather challenging to communicate with the same warth to a camera than with five graduate students in seminars taking place in the cozy lounge of a department: the atmosphere sometimes matters. Indeed, people have feelings, and examples also show that some instructors could not put up with the disappearence of students in their classrooms and forced them to come back by various means [8], thus ruining what we believe to be one of the positive points of recordings.

2. VIDEOS: HOW AND WHAT IMPACT ON LEARNING?

The authors of [6] cancelled all lectures of a 3^{rd} year database course, replacing them by videos available online. They found that the average grade increased from 49.4 to 51.1, with a standard deviation of 13.6 instead of 13.5: in other words, there seems to be no difference. Almost ten years later, a closer look at the situation was offered in [7] by comparing three methods: 'traditional' lectures in which the lecturer uses PowerPoint, virtual lectures made of small clips each on a particular topic (as for the previous authors) with text and self-evaluation questions, and recorded lectures. The results, displayed in figure 1 from [7] exhibit interesting disparities. Among the interpretations of those results, the authors suggested that an increased support through seminars and tutorials might be able to make up for the deficiency in analysis. Thus, one of the good points exhibited by one approach could try to balance the drawbacks of the other, and we start considering tradeoffs. Similarly, the use of videos in [6] is balanced by meeting the students once every two weeks, in order to answer questions and illustrate the most delicate points through examples.

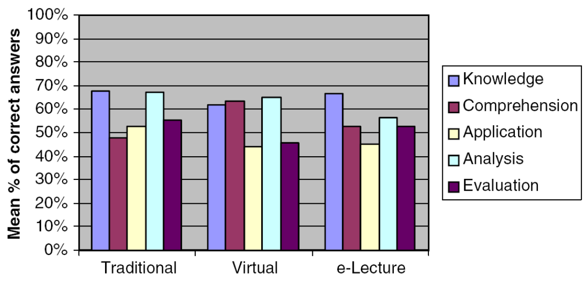


Figure 1: Mean grade of correct answers by teaching method and question type.

This makes up for our first suggestion: videos could be distributed, and the instructor ensures understanding by hands-on examples during tutorials or office hours (those being often used by only a handful of students). While some may fear that this is not doable because of time constraints, it is worth noticing that the authors turned 40 lectures of 1 hour into a set of chunks accounting for 18 hours, claiming that no material was lost: the ratio might seem a bit extreme, but it is clear that this approach rationalizes time costs. Furthermore, it may not be necessary to re-do the whole material for the next session, but rather updating parts of it, which leads to a huge saving of time but also to a major concern: why would an instructor be hired as many hours if he only has to do a few updates and tutorials? That leads to an interesting paradox: this approach could be *too* time saving and instructors would not be encouraged to record all of their lectures because by doing so they might lose parts of their job.

However, an interesting option comes into play if we consider that a same course might be offered online (by accessing the videos prepared for example a year before)... but also in-class. That way, alternative teaching approaches could be tried in class, and students are offered a choice of which style they prefer. Furthermore, this alternative could also be provided by a different instructor. In our experience, a class was taught simultaneously by two instructors, each having their classrooms next one another and separated only by a glass panel: students would try one instructor, then the other, and opt for the one that suits them best. This also provides immediate feedback: after a few weeks, one instructor could see his class becoming almost empty while his colleague's class was overcrowded, and this calls for changes.

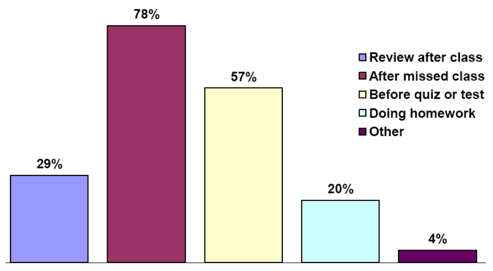


Figure 2: Cases in which students view recorded lectures.

The principles of an alternative were studied in [2], with the lectures being recorded and available online for distance learning students as well as the ones enrolled in the regular section. Studies showed that the former also watched some of the videos, and raised an interesting question: why do students watch videos of something they have seen in person? Or why, after all, do they keep coming to see it? Other research highlighted that only a third of the students may consider attending as optional when proposed with recorded lectures, and that recorded lectures are also massively used as a complement before exams or during homework (see figure 2 from [3]). Nevertheless, the concern that students would simply vanish into thin air because of recordings was expressed by a few instructors in [8]. Larry C. Booth, director of Veterinary Education & Technology at Iowa State University, summarized this situation in personal communication :

> "most faculty are receptive, but some resist with the excuse that if they offer [video recordings], then students won't come to class. My answer to them has been, if all you do in class is lecture, then why do they have to attend? Maybe they learn better and are more attentive at 11:00 PM at night, rather than in your 8:00 AM class. I encourage them to do things in class that will make students want to come such as in class group work, problems to solve, etc."

A corollary might be to wonder what is the point of having somebody physically present if there is no conversation. This leads to another suggestion: video recordings could be prepared, for example in studio, for the parts in which there is little or no interaction with students such as readings from notes or PowerPoint; in other words, the more *passive* parts of a lecture. Then, the lecture could introduce more *activ*ities, as instructors often have the feeling that they would like an increased interaction with students but that they would not be able to cover as much material.

3. VIDEOS FOR PASSIVE PARTS AND IN-CLASS SESSION FOR ACTIVITIES

A problem faced in undergraduate courses, and particularly within the sciences, is that we are teaching skills while what really matters are processes. In other words, students have memorized a set of tools, or facts, and may be able to apply them on a few cases, but it is challenging for them to design their own tools or evaluate one because critical thinking was simply not part of the package. This was constated by several authors, for example:

> "Science instruction in which students acquire a memorized collection of scientific facts and formulas does little to prepare students for their future courses or changing career demands [...] Instead, instruction should enable students to apply basic scientific principles flexibly, to explain or predict diverse phenomena, and to become good problem solvers and independent learners". [5]

However, the ability to explain diverse phenomena is linked to how many tools and approaches one is aware of: such tools, or facts, could be exposed online with videos putting emphasis on skills, whereas lectures or tutorials would be more process-oriented. Furthermore, as noted in [4], "first-year calculus courses are almost completely a matter of skill transfer"; thus, not only could we expose in such courses what mathematics are really all about by introducing problems and experiments, but we could also hope that it would show to be a more motivating approach against the struggles that freshman sometimes experience. After all, how attractive are sciences if all we show in introductory courses are techniques that have to be applied mechanically? Similarly, one can wonder about the usefulness of a course in which students keep writing down the instructor's monologue: it surely strengthens the wrist but perhaps not the mind; in particular, research has shown that faculty teach around 120 words/min whereas students record 20 up to 30 words/min [3], thus the later end up writing frantically and can be worried about taking notes instead of paying attention to the content. In a nutshell, the numbers speak for themselves:

> "traditional teaching, which many consider as the 'chalk and talk' method is ineffective [...] students generally remember 10% of what they read, 26% of what they listen to, 30% of what they see, 50% of what they see and listen to, 70% of what they say and 90% of what they say as they do something." [1]

Thus, one may hope that by providing videos (that students may pause when they have enough self-awareness to realize that they need a break), students would retain at least a third of the material, and this can be reinforced by activities in which students speak and do. Among the ideas, students could present developments on a topic or propose approaches to solve a problem: they would have to ensure a better mastery of the content to be able to present it, *i.e.* they would have to actually watch the videos and do complementary research, may it be for a few examples or to add significant material if time permits. In order to be efficient with regard to learning, we believe that PowerPoints or notes should be avoided in such presentations, and that most meaningful illustrations can be drawn on a flipchart. This has two reasons: firstly, we are aiming at a lively presentation in which the material is presented in real-time rather than read; secondly, while students could (and sometimes do) copy and paste a few lines on a PowerPoint, having to present without support is a better guarantee that learning has taken place. Meanwhile, other students could be asked to write down anonymous critics of the presentations: they would have to pay attention, apply critical thinking, and that would ultimately benefit the students about which they are writing. Note also that the role of an instructor in such sessions is of a *facilitator*, and it does not just come down to sitting and listening, which is of little use for the students. Instead, his knowledge on topics approached by the presentations should get him involved: he can define the flow by directing students through questions, and either suggest complementary research when students are facing open questions or disclose his answer.

4. CONCLUSION

The use of technology and in particular videos is a broad subject that shakes the foundations of teaching. Indeed, it asks fundamental questions: what do we *physically* need an instructor for? How to balance active and passive teaching styles? How can I provide enough content if I try to get students more involved by hands-on activities? While we tried to foster discussion on such topics, some aspects have inevitably not been mentioned. Among those, universities face budgetting aspects: cameras have a price as does technology in general; on the other hand, it does not only benefit students but constitutes a potential argument for advertising. Indeed, broadcasting some of the finest lectures contributes to the university's reputation, as many influential ones have already starting doing: Oxford, Cambridge, Stanford, and the MIT propose lectures through iTunes U (an educational initiative supported by Apple).

5. REFERENCES

[1] S. Ali. Effective teaching pedagogies for undergraduate computer science. *Mathematics and Computer Education*, 39(3):243–257, 2005.

[2] K. M. Cramer, K. R. Collins, D. Snider, and G. Fawcett. Virtual lecture hall for in-class and online sections: A comparison of utilization, perceptions, and benefits. *Journal of Research on Technology in Education*, 38(4), 2006.

[3] M. B. Harrity and A. Ricci. Putting classroom lectures online: the atc's course capturing pilot. *ATC, Worcester Polytechnic Institute*, 2008.

[4] D. Holton. Teaching versus lecturing. *Teaching mathematics and its applications*, 17(2):49–54, 1998.

[5] F. Reif and L. A. Scott. Teaching scientific thinking skills: students and computers coaching each other. *American Journal of Physics*, 67(9):819–831, 1999.

[6] A. Smeaton and G. Keogh. An analysis of the use of virtual delivery of undergraduate lectures. *Computers and Education*, 32(1):83–94, 1999.

[7] J. E. Stephenson, C. Brown, and D. K. Griffin. Electronic delivery of lectures in the university environment: an empirical comparison of three delivery styles. *Computers and Education*, 50:640–651, 2008.

[8] J. R. Young. The lectures are recorded, so why go to class? *The Chronicle of Higher Education*, 54(36), 2008.

Diversity in an Environment for Accessible Learning

Shohreh Hadian
Camosun College
Department of Computer Science
Victoria, B.C.
Tel: (250) 270 3971
shadian@camosun.bc.ca

Bill Wadge
University of Victoria
Department of Computer Science
Victoria, B.C.
Tel: (250) 472 5723
wwadge@cs.uvic.ca

ABSTRACT

The objective of the current study is to research, design, implement and test a tool that provides a repository of learning objects and records a profile of the users, in order to deliver a personalized and customized version of the learning object to the user taking into account their optimal learning profile and inclusive design in mind. The intention is to package and deliver content taking into account the diversity in learning subjects and materials encountered by teachers and learners in a classroom setting. To this end, intensional logic and programming is introduced to accommodate user diversity in a web-based teaching and learning environment. The Markup Macro Processor Language (MMP) [1] is used as an interpreter language that makes adaptations on the server as it transforms web pages "on the fly" without requiring web content to be re-written for each specific user.

Categories and Subject Descriptors

K.3.1 [**Computer Uses in Education**]: computer-assisted instruction, computer-managed instruction, distance learning.

General Terms

Algorithms, Design, Languages.

1. INTRODUCTION

The Web is being used as a major tool to teach and to conduct research. Web-based education presents several benefits since the material can be presented in various formats, and it is possible to review supporting material without interrupting the flow of learning. However, the technology implementation has proved far more difficult than anticipated and developers, educators and students need to become aware of the accessibility barriers and diversity issues in web-based environments to ensure that all users are able to participate in this rapidly evolving form of education.

The current research takes a small step towards understanding how web-based educational technology can help and enhance teaching and learning taking into account the diversity in learning outcomes. In order to optimize the learning experience, documents can, for example, be produced in different variants or versions, corresponding to different languages, different level of expertise, different dates or different target audience.

To this end, a new framework has been designed and implemented that facilitates multi-versioned web content authoring, content delivery, and web operation and navigation in order to aid teachers with content and pedagogical development and help learners access information in an optimal way with respect to the particular ability or disability.

A few studies in the nineties revealed that the available Web-based teaching tools fail to properly consider accessibility [2], and that web accessibility is about three times better for sighted users than for users with visual impairments [3]. In 2000, Storey et al [4] conducted a study on the usability of web-based learning tools. The study revealed that tools seem to enhance learning when they are perceived as being invisible. Also instructors, as the designer of course web sites, play an important role in using web-based learning tools. Furthermore, the study also recommends that the development of web-based learning tools with accessibility in mind is crucial to facilitate the learning process. A more recent study by Hadian [5] proposed a definition for accessibility; a rationale based on the ESSI-SCOPE software design guideline, was adopted [6]. The subsets of these characteristics that impact accessibility are depicted in Figure 1.

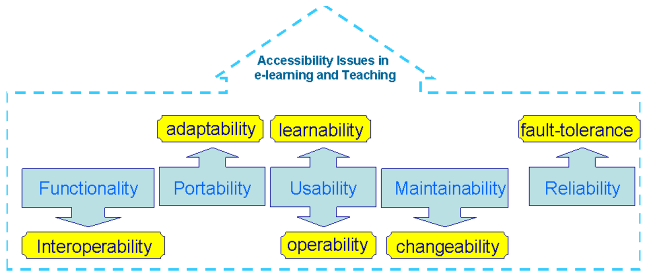


Figure 1. Definition of Accessibility

In this study, Hadian raised fundamental new issues in web based education: "In the quest for universal access and design for all-inclusive virtual classrooms, are we inadvertently creating barriers to other user groups that may also need to have access to the same virtual information? How do we target different users groups while maintaining optimal experience to each and every group? "The study revealed that changeability is one design characteristic that is important in the definition of accessibility. In a virtual classroom, the author has to deal with different types of learners that require varying types and levels of accessibility and they have different style of learning. The fundamental question that the current research project attempts to address is:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1-2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

"How do we foster the development of learning and teaching tools that accommodate "diversity" in a virtual classroom while ensuring "optimal" experience for all users?"

2. PROJECT OVERVIEW

The methodology and implementation of the project is described in Figure 2. This project consists of three inter-connected sub-projects that need to be dealt with concurrently and subsequently integrated. These three issues are: versioning issues, learning and teaching styles, and content issues.

The core effort presented in this paper is the multi-versioning issue, in order to be able to provide multiple versions of the same course content. This means that we need to move from the conventional linear environment to a multidimensional intensional programming environment. MMP [1] is the software that was designed and implemented as a macro processor and an interpreter to enable the author to insert intensions into the HTML code. Presently, the development environment is only suitable for advanced programmers. The second issue that was addressed is the need to create a learners profile in order to describe the users learning abilities. The third component deals with issues concerning storage and retrieval of the multi-versioned components.

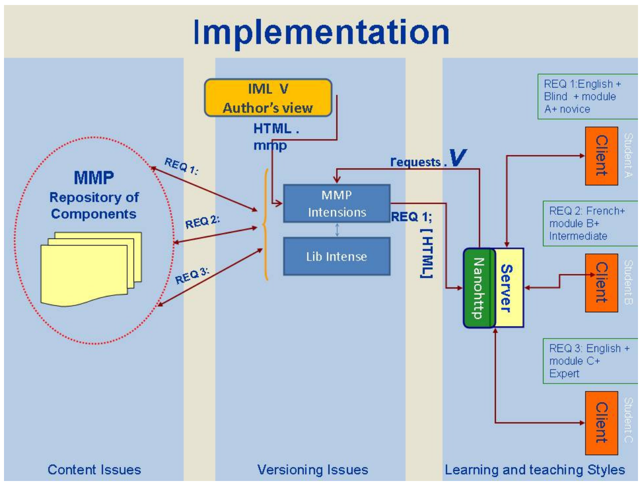


Figure 2 – The proposed methodology for addressing diversity in web-based education

2.1 Versioning Issue

Programming in a language based on intensional logic is called intensional programming [7]. This refers to programs that are sensitive to a global multidimensional runtime, which the program itself can modify. In the general case, an ***intensional*** program contains a number of versioned entities, each of which has a single valid instance under a particular global context. The set of (versioned entity, current context) pairs are termed an ***intension***, whereas a specific, resolved version of the entity is termed an ***extension***. As the program executes, Intensional entities are resolved to single versions against the global context.

With intensional languages, a versioned entity consists of a set of pairs of context-tagged extensional values (Ex, Cx), termed versions. The process of determining which version of an entity is used is termed a best-fit, and uses a partial order over the set of all contexts called refinement to compare the tag contexts of an entity with the global reference context Cr. The set of all tags in a versioned entity is termed a context domain [8].

In 1979, the idea of versioning started with Fieldman's work outlining MAKE for maintaining software [9] Wadge and Plaice created SLOTH as a new approach for controlling the multiple versions of a software. However, this original version of SLOTH was written in a monolithic manner and did not handle versions [8]. Subsequently, Brown coded LEMUR that allowed files to exist in multiple versions [10]. LEMUR was the result of intensionalizing SLOTH. From here, a subversion of LEMUR was created that allowed LEMUR to use not only the basic versions but sub-versions as well. Next, MARMOSET was created as an extension to LEMUR and this allowed different languages to be used using very simple configuration files and simpler than standard makefiles [11].

Hence, the idea of IHTML (Intensional Hyper Text Markup Language) arrived in 1996. This envisioned parallel documents similar to Nelson's Parallel Universe [12]. This was not a programming language and there was no provision for evaluating expressions. For instance, it could neither do constructs nor iterative constructs. Subsequently, Wadge and his co-workers proposed the Intensional HTML where the assembling work was done as a plug-in for the apache server. In 1999, this work was expanded and the Intensional Sequential Evaluator (ISE), a full-featured Perl-like scripting language that incorporated a (run-time) parametric versioning system, was created and written by Swoboda [13]. Here, all the entities in the language such as variables, arrays and function could be versioned. Due to difficulty in using ISE and placing intensions in HTML code, Scraefel and Wadge created a front end called Intensional Markup Language (IML) which consisted of TROFF macros to create encapsulation of the entities in the language.

Due to difficulty in using ISE and placing intensions in HTML code, in 2000, Scraefel and Wadge created a front end called Intensional Markup Language (IML) which consisted of TROFF macros to create encapsulation of the entities in the language [14]. The macros were for stretching text and other constructs in an attempt to repair the deficiencies in HTML by making it practical to author web pages over a multidimensional version space. The IML implementation used ISE, and it was a PERL like CGI language with run time parameterization. Figure 3 shows the current stage of development for the macro processor software for the *Intensional Markup Language* (IML) developed by Wadge and his co-workers [1,8].

2.2 Methodology

With focus on simplicity for authoring and usability, functionality, and extensibility in mind, and the Markup Macro Processor (MMP) was created to replace the various steps required to create requests with intended versions.

The modification in the architecture makes the steps simpler and increases the functionality of the tool as a programming language. For extensibility; the libintense (library of predefined macros) was created to enable an object modeling. The request and versions arrives from the user and the MMP interpreter performs the best fit algorithm to match the optimal component with the macro definitions in the libintense.

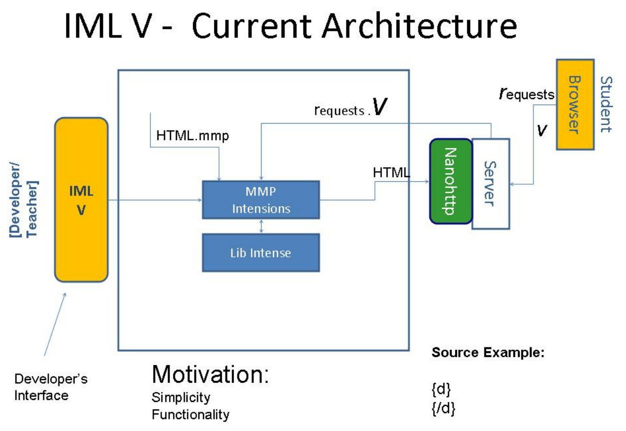


Figure 3. Intensional Markup Language (IML) Architecture with Markup Macro Processer as an Interpreter

In all the previous attempts at the IML, the load on the server was an issue. Apache as web server opens a session for every request it receives, so the bottleneck of data transmission becomes an issue; therefore, it was replaced by nanohttpd, a light weight web server written in Java, once we start an instance of the server, it keeps running and listen for any incoming traffic and sends requests through the MMP; code(which can be HTML, JavaScript, PHP) with the intensions imbedded in them gets processed at runtime and gets rendered to the browser (user) as an HTML document. The effect of nanohttp in high traffic area is still unknown. But in moderate traffic it has performed very well.

In theory, digital web pages can be updated frequently and provided in customized versions on demand. In practice, however, this is rarely the case. The cost of delivering a new or customized version is negligible, but the maintenance is a real problem. HTML itself provides very little support for the production and maintenance of multi-versioned sites. The problem is that different versions of the same site cannot share pages except through links, but then they have to share all the pages accessible from the shared page.

Consider a situation where a multilingual web site needs to be developed. For each language, the web site will have roughly the same structure and graphics. However, the text contained within each version of the web site will have to be different to reflect the chosen language. The conventional way to structure such a site is to create the English version, and subsequently translate the web content to a complete parallel version, one for each other language. In this scenario, each version would be a complete parallel site, with minimal reuse of the common structure of the web pages. Authors of multi-version sites are required to create copies by cloning, editing and maintaining the consistency of many separate but parallel pages. [15]

2.3 Implementation

MMP is a general purpose Macro Processor in the tradition of TRAC, M6, the troff/groff macro facility and others. It accepts an input text document and produces a corresponding output text document which results from copying ordinary text unchanged while expanding macro calls. MMP differs from other macro processors in the form of macro calls, as they look like XML elements, using a slightly changed and vastly simpler syntax. Expanding MMP macros can therefore be thought of as implementing user-defined tags.

MMP differs from XML in that tags are marked using curly parentheses ({ and }) rather than angle brackets. Also, attributes are not named; instead, the order in which they appear determines their significance (for which reason they are called arguments). Here is a typical MMP macro call:

```
{memo Tom Paul}I received your proposal.{/memo}
```

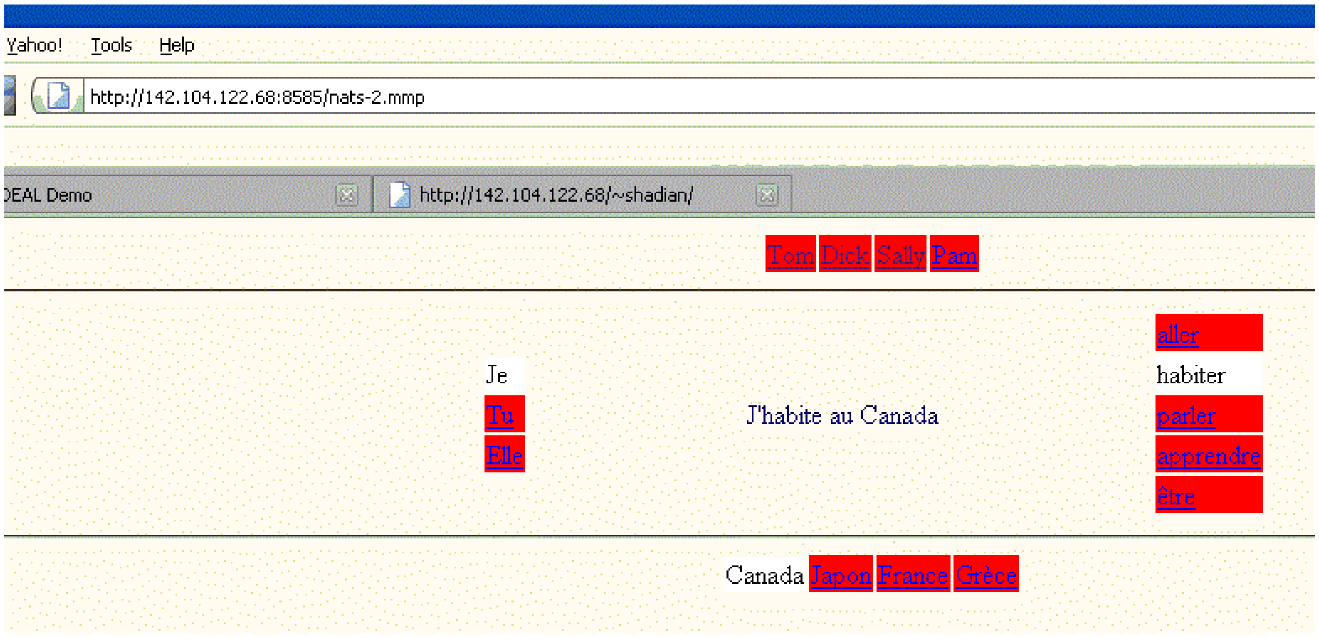
This is a call to a macro called *memo*. The first argument (actual parameter) is *Tom* and the second argument is *Paul*. The text between the opening and closing tags *I received your proposal.*) is called the *body* of the call. Let us assume that we have three clients connected to the server. Consider a web site that has more than one dimension. For example: a multi-lingual reference manual for an automobile, with low-bandwidth (for slow internet connections) and high-bandwidth (for fast internet connections) versions. There would be many dimensions here: language, automobile model number, model-year, specific options, and low or high bandwidth. Creating this web site using the conventional approach would result in an exponential blowup of the number of different versions that would need to be created and maintained. Creating the site using MMP would be possible since the problem itself can be simplified by partitioning the versions into different dimensions, and then tackling versions within each dimension.

The author creates the HTML code with version intensions in an ***.mmp*** file. Once the MMP processor has recognized and parsed a macro call, it replaces the entire call its value or result. The HTML pages that are sent to the client's browser are in the form of standard HTML. It is not necessary to provide files for each version that the user might request. The server uses a best fit algorithm to choose the version that most closely approximates the requested one. If the requested version is available, it will be used. Otherwise, if there is a unique closest match, it is used.

Let us look at the illustrative example in Figure 4. The upper part includes a group of people (*male and female*). On the left hand side it includes the nouns (*singular and plural*). On the right hand side we have the verbs (*aller, habiter, parler, apprendre, etre*). At the bottom, we have the countries. Based on the user's request, the central frame represents a customized output. This frame displays the user's preference based on the input selected. For example, if the user selects *je*, *habiter* and *Canada*, the central frame automatically displays *j'habite au Canada*. If the user selects *Nous, Sally and Pam, aller and Grece*, code. The display automatically shows *Nous allons en Grece*. The server is delivering a best fit customized output based on the request selected by the user. If the tool provides ways to provide multiple versions of the content, then it eliminates the need to create multiple version of the same site. The source code for this simple example is included in Appendix 1. Authors create multiple labeled versions of the MMP source for a given page. In turn, requests from clients specify both a page and a version. Next, the server software selects the appropriate source page and uses the source page to generate the requested actual html page. If the server-side software cannot find a source page with the exact version, it uses the page whose label most closely approximates the requested version. In turn, it treats the refinement ordering as a (reverse) inheritance ordering, which means that different versions can share source, authors can write generic, multi-version

3. ACKNOWLEDGMENTS

This project has been supported with a grant from the Canadian Council on Learning (CCL) and the CanAssist Research Group at the University of Victoria.



http://142.104.122.68:8585/nats-2.mmp

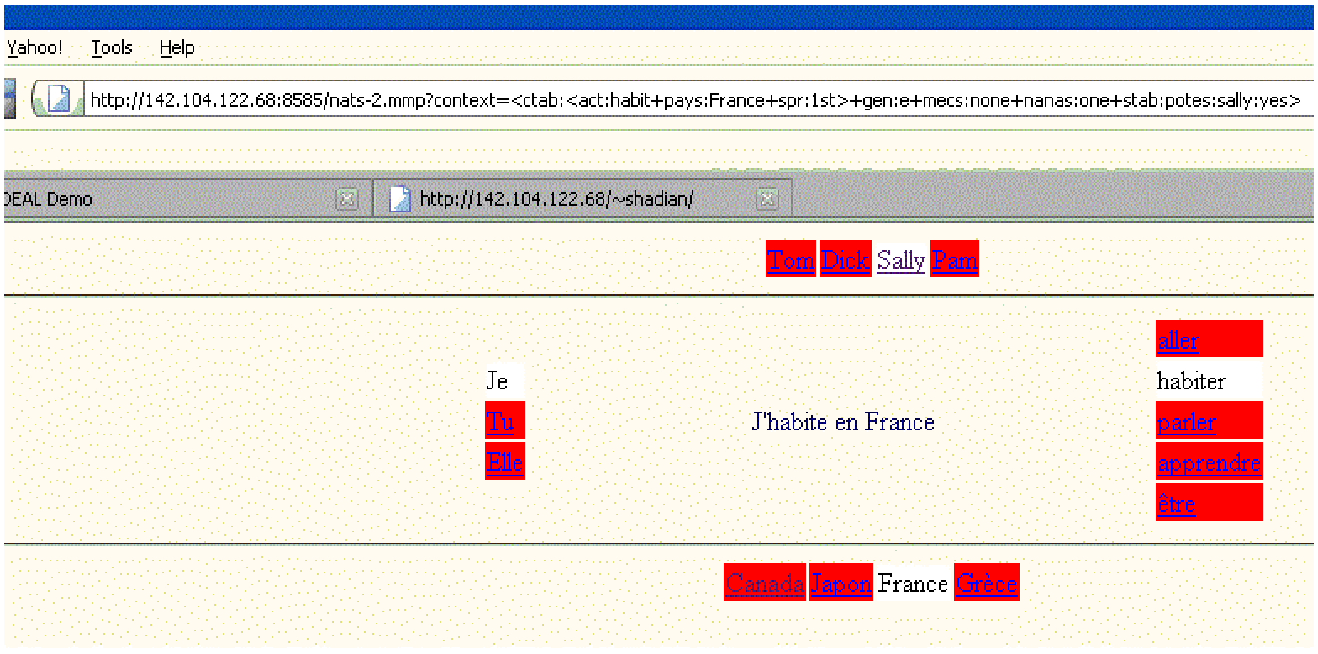
Tom Dick Sally Pam

Je
Tu
Elle

J'habite au Canada

aller
habiter
parler
apprendre
être

Canada Japon France Grèce



http://142.104.122.68:8585/nats-2.mmp?context=<ctab:<act:habit+pays:France+spr:1st>+gen:e+mecs:none+nanas:one+stab:potes:sally:yes>

Tom Dick Sally Pam

Je
Tu
Elle

J'habite en France

aller
habiter
parler
apprendre
être

Canada Japon France Grèce

Figure 4 – Implementation of an example for multi-versioning content

4. CONCLUDING REMARKS

A CMS (Content Management System) called *Diversity in an Environment for Accessible Learning* (DEAL) was designed and implemented for the dissemination of information and also to extend the CMS as a prototype tool to be intensionalized based on the MMP interpreter.

The challenge of dealing with diversity issue of users and the ability of a teaching and learning tool to deal with the issue of changeability is a daunting task. Software engineers and designers have developed tools like *ajax* and the *javascript* language to make the web site more dynamic for the user, but to fundamentally change the web interface philosophy to make it suitable to a diverse universe of users according to the learning profiles has not been accomplished to date. The multi-versioning capabilities in web software design is still in its infancy. This project has advanced the state of the art by proposing a new framework where it is possible to create web content based on the learner's ability profile. The teaching and learning tool, when completed, will prove to be beneficial to the community and teaching and learning as a whole. The development site and all of the reports and information are posted online. The information can be accessed using the following login parameters:

The DEAL website can be found at the following URL: http://142.104.122.68/framework/ and the original Online Learning project information is posted on the following website: http://142.104.122.68/OnlineLearning/index.php .

5. REFERENCES

[1] W. Wadge, G. Brown, M. Schraefel and T. Yildirim, 1996 "Intensional HTML", Proceedings of the 4th International Workshop on Principles of Digital Document Processing,.

[2] L. Harrison, 2000 ,"Inclusion in an Electronic Classroom - 2000: the Role of the Courseware Authoring Tool Developer", Adaptive Technology Resource Centre, University of Toronto, http://snow.utoronto.ca/initiatives/access_study/ATrec.html

[3] N. Nielsen, 2001,"Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities", Nielsen Norman Group, http://www.nngroup.com/reports/accessibility

[4] M-A. D. Storey, J. Bavelas, M. Wang and B. Phillips, 2002, "The Importance of Usability Testing for Web-based Learning Tools", Educational Technology Conference Series, May 4-5, University of Victoria, Victoria, B.C., Canada.

[5] S. Hadian, 2004,"Accessibility Issues in Web-Based Education: a Case Study for the Visually Impaired", M.Sc. Thesis, University of Victoria, Department of Computer Science, Victoria, B.C., Canada.

[6] ESSI-SCOPE software design guideline , 2009, http://www.cse.dcu.ie/essiscope/ http://www.cse.dcu.ie/essiscope/sm2/9126ref.html

[7] Faustini A.A. and Wadge W.W. ,1987,Intensional programming ,The Role of Languages in ProblemSolving ,J.C. Boudreaux, B.W. Hamil and R. Jenigan, eds.2, Elsevier North-Holland.

[8] Plaice J. and Wadge W., 1993,"A New Approach to Version Control", IEEE Transactions on Software Engineering, Vol. 19, No. 3, pp. 368–276.

[9] Feldman S.I., 1979, MAKE – A program for maintaining software , software-practice and Experience,Vol.9, no.3, pp.255-265,

[10] Brown G., 1998, Intensional HTML2, MSc Thesis, University of Victoria , Canada

[11] Tchuente M., 1992, Reducing the complexity of software configuration, Proc. 1st African Colloquium on Research in Computer Science,Yaoundé, Cameroon, Oct. 1992, pp. 85-96.

[12] Yildrim T., 1997, Intensional HTML, MSc Thesis, University of Victoria , Canada,

[13] Swoboda P., 1999,Practical languages for intensional programming Intensional programming ,MSc. Thesis, University of Victoria, Canada

[14] Wadge W. and Schraefel M.C., 2000,Putting the Hyper back in Hypertext,Intensional Programming II, Based on the papers at ISLIP, World Scientific Publishing Co ,

[15] Wadge William W., Brown G., schraefel m.c., and Yildirim T.,1998, Intensional html, Principles of Digital Document Production,In E.V. Munson, C. Nicholas and D. Wood, eds.,. LNCS 1481:128–139.

APPENDIX 1: SOURCE MMP FILE

```
{include html.mmp/} // link to another macro
{" : web}<html>     // embedded macro
<title>{1/}</title>
<body>
{.../}
</body>
</html>{/"}

{" : emph}<i>{bold}{.../}{/bold}</i>{/"}
{" : bold}<b>{.../}</b>{/"}

{web "Les nationalit&eacute;s"}  //macro call

{@: tdir}{/@:}
{@: tab}{/@:}

{" : initdim}{v}{" , ""}{[] {1/}:{2/}/}{/"}{" , {1/}:~}{/"}{/v}{/"}
{: so}{/:}
{: def}{/:}
{" : ctc}{cell{@ tdir/}}{button {1/} {2/}/}{/cell{@ tdir/}}{/"}
{" : stc}{cell{@ tdir/}}{sbutton {1/} {2/}/}{/cell{@ tdir/}}{/"}
{" : sbutton}<td bgcolor=white>{2/}</td>{/"}

{" : sbutton}
 {v}
  {" , ""}<td bgcolor=red>{alink}{2/}{/alink stab:{@ tab/}:{1/}:yes} </td>{/"}
  {" , stab:{@ tab/}:{1/}:yes}<td bgcolor=white>{alink}{2/}{/alink stab:{@ tab/}:--+stab:{@
tab/}:{1/}:yes}</td>{/"}
 {/v}
{/"}

{" : button}
 {v}
  {" , ""}<td bgcolor=red>{alink}{2/}{/alink ctab:{@ tab/}:{1/}} </td>{/"}
  {" , ctab:{@ tab/}:{1/}}<td bgcolor=white>{2/}</td>{/"}
 {/v}
{/"}

{" : cell}<tr>{_/}</tr>{/"}
{" : cellh}{_/}{/"}
{" : ctab}{@= tab}{1/}{/@=}{@= tdir}{2/}{/@=}{",/}<table>{_/}</table>{/"}
{" : stab}{@= tab}{1/}{/@=}{@= tdir}{2/}{/@=}{",/}<table>{_/}</table>{/"}

{" : def}{@: {*/}}{_/}{/@:}{/"}
{: dimexp}{/:}
{: incp}{/:}
{" : val}{@ {*/}/}{/"}
{: sval}{/:}

 {initdim ctab:pays Canada/}
 {initdim ctab:act habit/}
 {initdim ctab:spr 1st/}

 {[] mecs:-+nanas:-+num:-+gen:-/}

 {cvm stab:potes:<dick:yes+tom:yes> mecs two/}
 {cvm stab:potes:<dick:yes> mecs one/}
 {cvm stab:potes:<tom:yes> mecs one/}
 {initdim mecs none/}

 {cvm stab:potes:<pam:yes+sally:yes> nanas two/}
 {cvm stab:potes:<sally:yes> nanas one/}
 {cvm stab:potes:<pam:yes> nanas one/}
 {initdim nanas none/}

 {cvm mecs:two num s/}
 {cvm nanas:two num s/}
 {cvm mecs:one+nanas:one num s/}

 {cvm mecs:none gen e/}

{def fpp}Je{/def}
```

```
{def fpp num:s}Nous{/def}

{def spp}Tu{/def}
{def spp num:s}Vous{/def}

{" def tpp}Il{# num/}{/"}
{" def tpp gen:e}Elle{# num/}{/"}

{incp pays:Canada act:parl/}
{incp pays:Canada act:appren/}

{" def suj ctab:spr:1st}{val fpp/} {/"}
{" def suj ctab:spr:2nd}{val spp/} {/"}
{" def suj ctab:spr:3rd}{val tpp/} {/"}
{" def suj ctab:<spr:1st+act:appren>}J'{/"}
{" def suj ctab:<spr:1st+act:habit>}J'{/"}
{" def suj ctab:<spr:1st+act:appren>+num:s}Nous {/"}
{" def suj ctab:<spr:1st+act:habit>+num:s}Nous {/"}

{" def vph}{# ctab:act/}{val term/}{/"}
{" def vph ctab:<act:etre+spr:1st>}suis{/"}
{" def vph ctab:<act:etre+spr:2nd>}es{/"}
{" def vph ctab:<act:etre+spr:3rd>}est{/"}
{" def vph ctab:<act:etre+spr:1st>+num:s}sommes{/"}
{" def vph ctab:<act:etre+spr:2nd>+num:s}&ecirc;tes{/"}
{" def vph ctab:<act:etre+spr:3rd>+num:s}sont{/"}

{" def vph ctab:<act:all+spr:1st>}vais{/"}
{" def vph ctab:<act:all+spr:2nd>}vas{/"}
{" def vph ctab:<act:all+spr:3rd>}va{/"}
{" def vph ctab:<act:all+spr:1st>+num:s}allons{/"}
{" def vph ctab:<act:all+spr:2nd>+num:s}allez{/"}
{" def vph ctab:<act:all+spr:3rd>+num:s}vont{/"}

{" def vph ctab:<act:appren+spr:1st>}apprend{/"}
{" def vph ctab:<act:appren+spr:2nd>}apprends{/"}
{" def vph ctab:<act:appren+spr:3rd>}apprend{/"}
{" def vph ctab:<act:appren+spr:1st>+num:s}apprenons{/"}
{" def vph ctab:<act:appren+spr:2nd>+num:s}apprenez{/"}
{" def vph ctab:<act:appren+spr:3rd>+num:s}apprennent{/"}

{" def cmp ctab:act:etre}{val adje/}{/"}
{" def cmp ctab:act:parl}{val lang/}{/"}
{" def cmp ctab:act:parl+ctab:pays:Canada}fran&ccedil;ais et anglais{/"}
{" def cmp ctab:act:appren+ctab:pays:Canada}le fran&ccedil;ais et l'anglais{/"}
{" def cmp ctab:act:appren}le {val lang/}{/"}
{" def cmp ctab:act:all}{val dest/}{/"}
{" def cmp ctab:act:habit}{val dest/}{/"}

{" def dest}en {# ctab:pays/}{/"}
{" def dest ctab:pays:Grece}en Gr&egrave;ce{/"}
{" def dest ctab:pays:Canada}au {# ctab:pays/}{/"}
{" def dest ctab:pays:Japon}au {# ctab:pays/}{/"}

{" def adje ctab:pays:Canada}canad{val ien/}{/"}
{" def adje ctab:pays:Japon}japon{val ais/}{/"}
{" def adje ctab:pays:Grece}grec{# num/}{/"}
{" def adje ctab:pays:Grece+gen:e}grecque{# num/}{/"}
{" def adje ctab:pays:France}fran&ccedil;{val ais/}{/"}

{def ais}ais{/def}
{" def ais gen:e}aise{# num/}{/"}
{" def ien}ien{# num/}{/"}
{" def ien gen:e}ienne{# num/}{/"}
{" def lang}{! num:-+gen:-}{val adje/}{/!}{/"}

{def term}e{/def}
{def term num:s}ent{/def}
{def term ctab:spr:2nd}es{/def}
{def term num:s+ctab:spr:1st}ons{/def}
{def term num:s+ctab:spr:2nd}ez{/def}

<body bgcolor="EAEAEA">
<center>
{stab potes h}
 {stc tom Tom/}
 {stc dick Dick/}
 {stc sally Sally/}
 {stc pam Pam/}
{/stab}
<hr>
<table border=0>
<tr>
<td>

<table border=0>
<tr>
<td>

 {ctab spr}
  {ctc 1st {val fpp/}/}
  {ctc 2nd {val spp/}/}
  {ctc 3rd {val tpp/}/}
{/ctab}

 </tr></table>
   </td>
   <td align=center width=360>
    <font color="000099">
     {val suj/}{val vph/} {val cmp/}
   </font></td>
   <td>
{ctab act}
  {ctc all aller/}
  {ctc habit habiter/}
  {ctc parl parler/}
  {ctc appren apprendre/}
  {ctc etre &ecirc;tre/}
{/ctab}
  </td>
  </tr>
 </table>
 <hr>
{ctab pays h}
 {ctc Canada Canada/}
 {ctc Japon Japon/}
 {ctc France France/}
 {ctc Grece Gr&egrave;ce/}
{/ctab}

</center>
</body>
{/web}
```

html.mmp Linked Macro Source Code

```
{" : link}<a href={1/}?context={en}{! {-
1/}}{#/}{/!}{/en}>{.../}</a>{/"}
{" : alink}{link "{% URI/}"}{_/}{/link {-1/}}{/"}
{" : en}{r "<" "%3C" ">" "%3E"}{.../}{/r}{/"}
{" : h}<a href={1/}.mmp>{<</}{*/}</a>{/"}
{" : ph}<a href={1/}.mmp?{<</}{ap{a#/}
{*/}/}>{.../}</a>{/"}
{" : ap0}{/"}
{" : ap1}1={1/}{/"}
{" : ap2}1={1/}&2={2/}{/"}
{" : ap3}1={1/}&2={2/}&3={ap3}{/"}
{" : r$}{r {"}\$1 "{1/}" \$2 "{2/}" \$3 "{3/}" \$_
"{_/}"{/"}}{_/}{/r}{/"}
{" : de}{: {1/}}{r$}{_/}{/r$}{/:}{/"}
{" de cvm}
 {v}
  {" , $1}{[] $2:$3/}{/"}
  {" , $1+$2:~}{/"}
  {" , ""}{/"}
 {/v}
{/"}
{" : html#}{r \< \&lt; \> \&gt;}{#/}{/r}{/"}
```

An Oral Communications Course for EECS Majors at MIT

Tony Eng
MIT Department of Electrical
Engineering and Computer Science
32 Vassar St
Cambridge, MA 02139 USA
+1 617 253 5868
tleng@mit.edu

ABSTRACT

This paper is about an oral communications course called "6.UAT" which is required of all students in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. We describe details about the course, discuss some high-level design decisions, and offer some results from student surveys about their communication ability before and after taking the course. Some of the curriculum ideas are not specific to EECS and thus are adoptable by the general technical community, and can serve as a possible starting point for practitioners at other institutions going through similar communications course design efforts.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education -- Curriculum

General Terms

Human Factors

Keywords

Communication skills, Soft skills. Professional skills. Oral communication. Oral presentations.

1. INTRODUCTION

Undergraduate students at MIT are required to take two communication intensive courses offered in their major (CI-M), one during their third-year and one during their fourth.

In the Fall of 2003, we were asked to design a course suitable as a fourth year CI-M for the Department of Electrical Engineering and Computer Science(EECS). At the time, one school of thought envisioned a course that would assist students with the writing of their undergraduate thesis. But we found the prospect of editing thesis drafts every semester unappealing, and sought something that had longer-term and further-reaching impact on the career of an MIT graduate.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1-2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

After some discussion with EECS alumni, company recruiters, and supervisors of EECS graduates, we confirmed the existing need for better oral communication skills in the workplace (something already well-reported in the literature[2,4,7]), and after researching the communications landscape at MIT, we found that there was indeed a place for such a course at MIT.

We designed and proposed a course focusing on professional communication skills, and not only was the course approved, it was made into a formal requirement and designated as the only fourth year CI-M available to students in EECS. In Spring 2004, 17 students from the Class of 2004 volunteered to take the experimental pilot, and now, five years and almost 1000 students later, we describe the course, its goals and assignments, some of the initial challenges and decisions that were made, as well as some feedback from students who have taken the course.

2. BACKGROUND

The current Communication Requirement at MIT, established in Spring of 2000, requires that each undergraduate (starting with the Class of 2005) take four communication-intensive(CI) courses, roughly corresponding to one per year. The first two courses should come from the Humanities (CI-H), and the remaining two, from the student's major (CI-M). These CI-M subjects had to "teach the specific forms of written, oral, and/or visual communication appropriate to the field's professional and academic culture" (http://web.mit.edu/commreq), and it was left to the departments to decide what these professional communication skills were, how to incorporate them into the curriculum, and how to administer the requirement within the parameters set forth by a Subcommittee on the Communication Requirement.

The Department of Electrical Engineering and Computer Science (EECS) is the largest department at MIT, with approximately 200-250 students per undergraduate class, or about one-fourth or one-fifth of the entire undergraduate population. Currently, EECS students can satisfy the third year CI-M by choosing one course from a set of communication-intensive labs which usually involve writing lab reports/papers and/or giving presentations. All EECS students satisfy the fourth year CI-M by (1) taking 6.UAT and (2) doing a senior capstone thesis project.

3. CONTEXT

Efforts to include communication skills development in the undergraduate engineering curriculum abound in the literature ([1,3,5,8,9] to name a few) so 6.UAT is definitely not new in this regard. However, 6.UAT was a novel offering for MIT(to the best of our knowledge), and was originally conceived and designed independently, without knowledge of what other campuses were doing in this area. Since then, we've met with a few individuals who run similar programs at other universities (e.g. Professor

Stock and Dr Osburn with CONNECT at Cooper Union[8] in 2006) to understand their approach and to learn from their experiences.

4. 6.UAT TODAY

6.UAT is an oral communications and professional skills course, designed to help EECS students be more effective in the workplace. It is a 6-unit course that is a prerequisite for the senior capstone thesis project. (MIT is on a 14-week semester system, with most courses typically carrying 12 units.) The course consists of large group lectures and recitation sections of 8-9 students; the latter are each led by a faculty member/graduate student pair. There are three contact hours per week (depending on the week, either two lectures and one recitation section, or one lecture and two recitation sections). On average, we expect students to spend three hours on homework per week, although in practice, this varies from student to student. Finally, 6.UAT is offered every semester, with about 150-200 students (mostly seniors) in the Fall, and about 50-70 (mostly juniors) in the Spring, and although it is part of the fourth year CI-M in EECS, anyone can take the class.

There are three main goals: each student should

1) improve in some aspect of their oral presentation ability,
2) make headway towards finding a thesis project, and
3) begin to develop professionally.

4.1 Improving Presentation Skills

Part of being an effective engineer is to be able to present and explain what it is that you know. As a result, the lectures in the first part of the course (see the first six lectures in Table 1) discuss different aspects of effective oral presentations: from the importance of the beginning of a talk, to the creation of glanceable visuals, from the need to understand the audience, to the specific needs of the non-technical audience, and from the value of good delivery to methods of recovery when things go wrong. Recitations reinforce lecture material with various exercises, and provide a venue for students to give the presentations that they've prepared for their assignments (discussed later).

By the time students take 6.UAT as upperclassmen, they have had a range of communication experiences and abilities from their coursework, internships and activities. However, there is still something for everyone:

- those with little experience gain valuable stage time and learn to deal with nervousness,
- those with some more experience, master certain aspects of delivery so that they become second nature, so that e.g. the student can then concentrate on the content and not have to be consciously worrying about whether or not they are making eye contact, and
- those with much experience can experiment with different presentation styles.

We stress to students that this is a learning environment, and that it is better to make mistakes and learn from them now, in the relative safety of the recitation section, than later in a professional workplace setting, where the consequences of a mistake can be far greater. Finally, we also remind them that the more they put into the course, the more they'll likely get out of it.

4.2 Making Progress Towards a Thesis

In the past, students received little formal assistance in finding a thesis at MIT - it was largely their own responsibility to contact professors and to secure a suitable project in order to graduate. 6.UAT nudges them along towards the thesis by providing information on possible ways to find a thesis, and how to interact with the thesis advisor while working on the thesis (foreshadowing the supervisor relationship they'll have in the workplace).

The final project in 6.UAT (the Proposal Talk) is an oral presentation in which a student proposes a possible technical project that may serve as their thesis undertaking. In preparation for this assignment, there is a lecture on effective proposals and a short assignment on library research skills. The latter is intended to equip them with information gathering and search tools (a problem highlighted in [6]) which they can immediately use to find previous work in their thesis area.

4.3 Developing Professionally

While a student is in school, he/she is following a path that someone else has decided -- someone determined what students needed to learn, created a syllabus and designed the problems and labs. In the real-world, there is no clear delineated path for continued professional development; a good manager may identify areas for further develop and recommend training, but the graduate is largely on his or her own.

While the lectures in the first half of 6.UAT address presentation skills, those in the later half address other skills that will help a graduate succeed in the workplace. The particular set of topics sometimes differ slightly from term to term, but the Spring 2009 topics are listed in Table 1. Many of these topics can themselves be the subject of a semester-long course. The idea here is to make students aware of the need to be proactive in their professional developments after MIT, that this development can in fact start now, and that, should they be interested in further studies, many of these topics are covered in more detail in classes they can take while they are still at MIT.

Table 1. Sample Lecture and Recitation Topics.

Lectures	Recitations
Beginnings : First Impressions and Credibility	Introductions
Audience	Feedback
Visual Communication	Storyboarding
Explaining the Technical to the Non-Technical	Signposting
Delivery	Impromptus
Recovery	Coaching
Looking Beyond/Working with Your Manager	
Being Effective in the Workplace	
Negotiation	
Estimation	
Workplace Ethics	
CrossCultural Communication	
Networking	
Proposals / Literature Searching	

4.4 A Range of Oral Assignments

Over the course of a semester, students give five oral presentations, representing at least 85% of their grade in the course. The content of all but one of the talks is technical in nature and chosen by each student, and except for the last one, all of the presentations occur during normal recitation time when they get immediate peer feedback and staff comments.

The presentations vary in length, theme, audience and purpose.

Table 2. Oral Presentation Assignments.

Assignment	Time	Audience	Purpose
PreviousProjTalk	4-5min	General MIT	Continuity Storyboarding
ConceptTalk	7-9min	NonTechnical	Explaining Recovering
EthicsDebate	Team	General	SpeechReading Impromptus
ChalkTalk	8-10min	EECS Peers	TechnicalDetails Using board
ProposalTalk	15min	Supervisor	Persuasion Interruptions

The first talk, the Previous Project Talk, is given by week 3 of the course. In this talk, a student describes a project that they worked on. It is a very structured talk of four slides (Title, Context, Background and Contribution), and it gives us a sense of each student's baseline presentation ability. We have them perform a self-critique by watching a video of the presentation, so it also gives them a sense of where they are at. Lastly, we also use this talk to teach students about storyboarding (the ordering of slides) and signposting (the inclusion of language that guides the audience through a presentation).

For the second talk, the Concept Talk, students choose a technical topic from EECS and struggle with how to present the intuition behind this topic to a non-technical audience. This assignment is combined with a high school service-learning outreach activity aand culminates in a one-day conference for local high school students who attend to learn about some topics in EECS and to serve as a live target audience. There is a mandatory revision of this talk: students first give a dry-run in recitation, and they are expected to incorporate comments into an improved version which is delivered at this conference. From this assignment, they learn how to explain technical concepts and gain some experience reacting to and adjusting to a live audience that is not their usual "safe" recitation audience.

The third talk is a debate in which students are given a hypothetical scenario that they may encounter in the workplace. In this scenario, an individual is confronted with a decision of some sort, and students within a recitation form teams to debate what the individual should do. Students are expected to come up with supporting arguments, and frequently this leads to due diligence in the form of consulting codes of ethics, finding previous case history, interviewing professionals, etc. Due to the debate format, students have a chance to try their hand at speech-reading and impromptu-speaking.

The last two assignments are thesis-related. For the fourth talk, students pick a technical paper from their intended thesis area, and present some of the results in the paper using a whiteboard only. This gives students a chance to start delving into some of the background/previous work associated with their thesis topic, and to gain experience using a medium other than slides.

In the last talk, the Proposal Talk described earlier, a student needs to convince a potential thesis supervisor, that a proposed project is feasible and worth doing. It is a 15 minute talk given over a 30 minute slot because unlike the previous assignments, the audience is expected to interrupt at any time.

Students can and are encouraged to do a consequence-free dryrun of this presentation before any 6.UAT staff member, who will give them point-blank feedback on what worked and what didn't work.

All talks are graded and evaluated according to rubrics for content and delivery that are listed at the end of each assignment.

5. DISCUSSION

There are a number of questions that we are often asked, and these are addressed briefly below.

How did you settle on oral communication?

There were more written assignments in the first few offerings of 6.UAT, but we decided to remove them. Rather than trying to improve both written and oral communication and doing a mediocre job at both, we decided to focus on one in order to do a more thorough job. Since there is plenty of writing elsewhere in the curriculum, we chose oral communication.

Additionally, in the original version of the course, we envisioned an industry track and an academic track. But we discovered that enough skills were common enough to both career trajectories, that such a separation was unnecessary.

Why in the senior year?

One could easily argue the benefits of teaching basic presentation skills earlier in the undergraduate curriculum, and we've had students enroll as sophomores (and even a freshman). But aside from the logistical difficulty of moving this earlier (into either a General Institute Requirement or an earlier EECS requirement), there is something to be said about keeping 6.UAT as an upperclassman class -- juniors and seniors have more proficiency in their field to understand the examples and appreciate the need to communicate, and they have more technical experiences, gained through coursework, internships and research opportunities, they can draw from as content for their presentations. In some ways they might be more motivated to learn and improve their communication skills because of their impending entry into the next professional stage of their careers.

How do you overcome student sentiment/resistance?

In the early years, there was student resistance to this course. This was probably due to three factors: (1) the imposition of a new mandatory requirement, (2) the "soft airy-fairy" nature of the material in the course, and (3) student belief that they presented well enough and didn't need a course to "teach them how to talk".

There were several things we tried to do. We worked hard to present material in a relevant, logical and when possible, creative manner, and we try to do so in a way that an engineer would find palatable. The EECS Department showed their commitment to the course and to the importance of good communication by assigning faculty members to teach recitation sections. We wanted to show that alumni and industry supported the effort by involving them in the course as well (this aspect was largely unsuccessful). But perhaps the largest factor contributing to increased acceptance of 6.UAT was the influence of alumni of the course – their advice and comments (e.g. "6.UAT is not that bad") to their fellow students affect the pre-conceptions and opinions of students who have not taken the course.

As for the third factor, we try to find ways to challenge even the students who have had a fair amount of presentation experience, and we encourage students of all levels to be proactive in finding/making value for themselves in the course.

Isn't grading subjective?

There are some guidelines for grading, but the nature of grading in this course is still very subjective - what strikes you one way, may strike me differently. At the start of the term, the staff watch and grade two sample talks to get a feel for what an "A" talk might be like, what a "B" talk might be like, etc. Each talk is reviewed by two staff members: one faculty member and one graduate teaching assistant (TA). A TA works with multiple faculty members, so assuming individuals grade consistently, this overlap helps to normalize the grades a bit. Lastly, a student can always request a dryrun to get comments from their instructor/TA before the actual presentation. Even with these measures, grading remains subjective, and in some ways, this mirrors the real world situation in which some bosses are stricter and more demanding than others.

6. RESULTS

Every semester, students in 6.UAT are surveyed and asked to self-rate (1) their ability to perform seven presentation-related actions (e.g. deliver a presentation, evaluate a presentation, receive feedback, etc) and (2) their public-speaking experience. They are surveyed at the start of the term ("pre") and at the end of the term ("post") to see if 6.UAT had any effect. Tables 3 and 4 contain five semesters-worth of averaged self-ratings. Note that (1)the values are rounded to the nearest tenth, (2)the rating scales are different (0-5 and 0-4 respectively), and (3) the difference in number of respondents is due to the fact that some students dropped the course and in some semesters, completion of the end-of-term survey was optional.

The data from Table 3 is visually represented for easier viewing in Figure 1. Each line segment represents a pair of related pre and post values. These lines are organized into groups of five, (because there are five semesters of data). Each group of five is labeled with one word representing its corresponding question .

7. DISCUSSION

The positive slope for every line in Figure 1 shows that students felt that they had improved in all seven abilities – the higher the slope/longer the line segment, the greater the improvement. The data seems to indicate that they improved more in certain areas (e.g. identifying the kind of presentation to give, architecting a presentation and evaluating a presentation) than in others. The plot suggests that students feel most comfortable with their ability to receive feedback, and least with adjusting to an audience in real-time.

Questions #2 and #4 of Table 4 ask students to self-rate their overall ability as public speakers and technical presenters, at the start and end of the term. Every semester, students believe that they are better speakers and technical presenters by the end of the course. Their answers to questions #1 and #3 aren't really comparable; these were included only to get some sense of the amount of experience they thought they had prior to taking the course, and the extent to which they thought the course had contributed to their improvement.

The questions in these surveys were not part of an official rigorous informed assessment effort; it was merely information that we thought was important to know. While a more formal assessment of 6.UAT by MIT's Teaching and Learning Laboratory is currently in progress, our simple survey seems to suggest that students perceived that 6.UAT had some positive effect on the development of their oral presentation abilities.

8. CONCLUSIONS

We've presented 6.UAT along with some justification of certain high-level design decisions and some ideas for activities that may hopefully be of inspiration for and/or of use to others in the community. The course has come a long way, and is constantly being updated and improved. It is the first of its kind at MIT, and despite the initial challenges of teaching a non-technical soft skills course to a technical audience, it appears to be making a difference, at least from the students' perspective. We leave you with some student remarks about the course from last semester.

9. ANECDOTAL STUDENT COMMENTS

The following are answers made by students from the Fall 2008 semester to a question asking them for the best piece of advice they received in 6.UAT. All answers are quoted verbatim.

A TA told me to avoid walking around while speaking and this helped me a great deal; I had never realized this before.

I especially appreciated the feedback from the other students in the class during presentations.

B/C of this class, I am a lot more aware of my delivery and how I present myself. Also, I am very careful about storyboarding now as well.

To pay attention to breathing while presenting. I didn't notice until then that I tend to lock up and stop breathing as deeply as I should, which adds to my tension.

Watch yourself on tape. You really see stuff you didn't know was there.

You must be able to measure your progress and be honest with your shortcomings. I was able to measure myself throughout the semester and I learned that public speaking can be fun and improvements can be made with practice.

It is very hard to teach these skills in a short amount of time, and UAT did a much better job of it then any other similar course that I have taken. I strongly recommend the lecture videos to my friends in other fields.

I wouldn't consider myself to be a great public speaker, but I believe that the time we spent giving presentations/doing speaking exercises improved my public speaking skills, which I didn't think was possible.

Being aware that many activities provide opportunities to improve our speaking abilities was great advice! It definitely lets us think about our speaking style and skills while doing seemingly unrelated activites.

Keeping this in mind gives us a great number of opportunities to practice and improve.

10. ACKNOWLEDGMENTS

Our thanks go to the various alumni, recruiters, students, faculty, TAs and industry representatives whose feedbacks and suggestions helped evolve 6.UAT to what is it today. We are also indebted to the anonymous reviewers for their thoughtful comments and suggestions.

11. REFERENCES

[1] Agoki, G., Ng, B., and Johnson, R. 2007. Development of communication skills and teamwork amongst undergraduate engineering students. In Proceedings of the 37th ASEE/IEEE Frontiers in Education (October 10-13, 2007). FIE. IEEE Computer Society, Milwaukee, WI.

[2] Darling, A., and Dannels, D. 2003. Practicing engineers talk about the importance of talk: a report on the role of oral communication in the workplace. Communication Education, 52,1(2003), 1-16.

[3] Gruba, P., and Al-Mahmood, R. 2004. Strategies for communication skills development. In Proceedings of the 6th Australasian Computing Education Conference, 101-107.

[4] McDonald, G., and McDonald, M. 1992. Developing oral communication skills of computer science undergraduates. In Proceedings of the SIGCSE Technical Symposium on Computer Science Education, (1992), 279-282.

[5] Norback J., and Hardin, J. 2005. Integrating workforce communication into senior design. IEEE Transactions on Professional Communication, 48,4 (2005), 413-426.

[6] Rodriguez, R.. 2001. Industry expectations of the new engineer. Science & Technology Libraries, 19, 3 (2001), 179-188.

[7] Sageev, P., and Romanowski, C. 2001. A message from recent engineering graduates in the workplace: results of a survey on technical communication skills. Journal of Engineering Education, (2001), 685-693.

[8] Stock, R., and Osburn, J. 2004. Communicating in the engineering curriculum: practice makes perfect sense. In Proceedings of the Conference on Integrating Practice into Engineering Education, (University of Michigan-Dearborn, M.I., October 3-5, 2004).

[9] Watson, J., and Alexander, C. 2005. Communication aspects of ProSkills: a non-technical skill development and enhancement program for engineers. In Proceedings of the 2005 IEEE International Professional Communication Conference, 631-637.

Table 3. Students were asked to self-rate their ability (on a scale of 0 to 5) to do certain presentation-related actions, both at the start and end of the semester ("Pre" and "Post" respectively). Average ratings, rounded to the nearest tenth, are reported for five consecutive semesters, from the Fall 2006 (F06) to Fall 2008 (F08). The number of students responding to the Pre and Post surveys is noted in the second row.

	F06		S07		F07		S08		F08	
(Scale: 0 to 5)	Pre 195	Post 91	Pre 71	Post 21	Pre 189	Post 34	Pre 80	Post 70	Pre 156	Post 142
Identify kind of presentation to give	3.3	4.0	3.1	3.9	3,3	4.0	3.2	3.9	3.3	3.9
Architect a presentation	3.5	4.1	3.2	3.9	3.4	3.9	3.2	4.0	3.4	3.9
Deliver a presentation	3.3	3.9	3.1	3.4	3.2	4.0	3.1	3.7	3.2	3.6
Dynamically adjust to an audience	2.8	3.3	2.6	3.2	2.9	3.6	2.9	3.5	2.8	3.3
Evaluate a presentation	3.2	3.9	3.3	4.1	3.3	3.9	3.2	3.7	3.4	3.8
Receive constructive feedback	3.8	4.3	3.9	4.0	3.8	4.2	3.7	4.1	3.8	4.1
Given constructive feedback	3.3	3.9	3.3	4.0	3.3	3.8	3.2	3.6	3.4	3.7

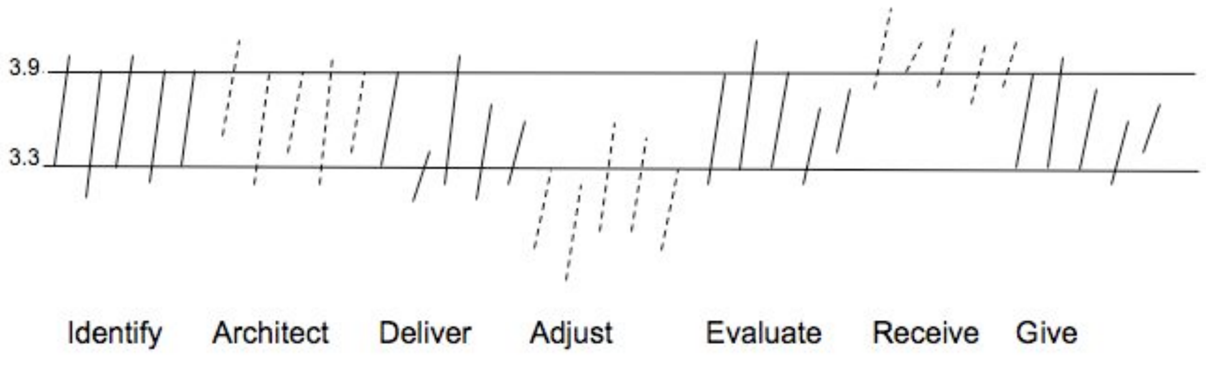


Figure 1. A visual representation of the data in Table 3 is presented. The Pre and Post average values for a question in Table 3 determine the left and right endpoints of a line. Each cluster of five lines represents five semesters worth of Pre and Post averages for a single question; this question is identified below the cluster. The lines for all questions are plotted on the same scale with guide lines shown for values of 3.3 and 3.9.

Table 4. The same students from the survey in Table 3 were asked to self-rate their presentation experience (scale of 0 to 4) at the start and end of the semester ("Pre" and "Post" respectively). Averages are shown below, rounded to nearest tenth, for the past 5 semesters.

(Scale: 0 to 4)	F06	S07	F07	S08	F08
PRE					
How much experience do you have in general public speaking?	2.8	2.6	2.8	2.6	2.8
How would you describe yourself now as a public speaker?	2.4	2.2	2.4	2.2	2.3
How much experience do you have giving technical presentations?	2.5	2.4	2.4	2.3	2.5
How would you describe yourself now as a technical presentation giver?	2.2	2.0	2.2	2.1	2.2
POST					
How much experience **have you gained this term** speaking in general?	3.6	3.6	3.5	3.5	3.5
How would you describe yourself now as a public speaker?	3.0	2.7	3.1	2.8	2.8
How much experience **have you gained this term** giving technical presentations?	3.2	3.3	3.3	3.4	3.4
How would you describe yourself now as a technical presentation giver?	3.0	2.6	3.1	2.9	2.8

SW Development Projects in Academia

Youry Khmelevsky, Ph.D.
Computer Science Department
Okanagan College, 1000 K.L.O. Road
Kelowna, BC, V1Y 4X8
+1 (250) 762-5445 loc. 4741

Youry <Khmelevsky@acm.org>

ABSTRACT

A significant amount of current software engineering research is conducted within the context of computer science and computing departments or colleges. Similarly, software engineering degree programs are being developed by such academic units as well as within engineering colleges [1]. However, every computer or computing department has its own experiences, successes or pitfalls in software engineering (SE) and software development (SD) teaching, which would be useful to share and discuss with the education community. In this paper we discuss the experiences and results from four years of teaching "Projects in Computer Science" in Computer Information Systems (CIS) Diploma and "Software Engineering" in Bachelor of Computer Information Systems (BCIS) Degree programs [2] at Okanagan College. Also we provide analysis and evaluation for several already finished and current projects. The class teaching in both programs was organized into two parts. The learning of SD in the CIS as well as SE in the BCIS programs were synchronized with the practical SD and SE projects with real sponsors from industry and academia in small and medium size groups of students (3-6 members in SD and 5-11 members in SE projects). Students developed their final projects incorporating four main Rational Unified Process phases [3] and 4 - 7 short iterations typical to the Agile SD process. Additionally, in the SE projects students learned and used Extreme Programming (XP) iterative process [4]. In several projects students successfully combined Software prototyping [5] with Agile SD and Rapid Application Development (RAD) tools [6]. Instructors supervised and supported students in the role of sponsors or mediators. Many student groups were able to develop impressive, high quality final project applications and systems. The sponsors provided very positive feedback and references for most of the projects.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management - *Programming Teams, Life Cycle*

D.2.10 [**Software Engineering**]: Design

K 3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education*

General Terms

Management, Documentation, Design, Human Factors

Keywords

Projects; Software Engineering education; Software Development; Computer Information Systems; College

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1-2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

1. INTRODUCTION

SW Engineering education (curriculum, outcomes and delivery) has received considerable attention from research and professional societies [7]. "Software engineering is the means by which we attempt to produce all of this software in a way that is both cost effective and reliable enough to deserve our trust . . . . [It is] the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them" [8]. The modern definition of SE states, that "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software" [9]. Additionally, as it is clearly stated in [1] SE is about creating high-quality software in a systematic, controlled, and efficient manner. There are important emphases on analysis and evaluation, specification, design, and evolution of software. In addition, management and quality, novelty and creativity, standards, individual skills, and teamwork and professional practice also play a vital role in software engineering [1].

In our previous papers [10 - 14] we started discussion about quality and security issues in the SD and SE projects. In this paper we discuss the "Projects in Computer Science" course [1] in the Computer Information Systems Diploma program and the "Software Engineering" course in the Bachelor of Computer Information Systems program [2] at Okanagan College more deeply. We present in some detail the teaching methods and SW development tools we used.

In the next sections the goals and objectives for both SD and SE project courses, technological environments and management solutions will be discussed, and then examples of some successful projects in the last four years and some current student work examples will be given. In the Summary and Suggestions to

Improvement Section some possible improvements for the SD and SE courses will be suggested.

The successful projects were selected by feedback from the external or internal sponsors, by the final project grades and from the verbal instructors' feedback, who took part in the final project presentation. The examples are related to the goals of the paper only.

2. GOALS AND OBJECTIVES

The main goal of "Projects in Computer Science" (CoSc 224) is stated as follows [1]: "the student will analyze, design and implement a chosen system design under the tutelage and guidance of the professors. A selection of suitable projects will be provided to realize the system design theory studied in the program. Students may select local industry and business problems with the co-operation of the companies selected".

On the other hand in "Software Engineering" (CoSc 310/319) the main goal is different [2]. The students design and implement large, multi-module program systems by implementing the software life cycle; design tools and use of module-oriented programming languages. One of the goals of this course is to introduce Extreme Programming (XP) by emphasizing team work; make stresses on sponsor satisfaction, deliver the SW to the sponsor when it is needed, confidently respond to changing sponsor requirements, and promote SW quality and security. This is a unique style in SE, which is based on pair programming, and four main rules and practices: planning, designing, coding and testing [15].

In the last few years the following SW programming languages, development tools and IDEs were introduced to the students in projects and other related courses: Java, C++, .NET, LAMP, UNIX shell scripting, Oracle Express Edition (EX) and Oracle Application Server Express Edition (APEX), Oracle Designer, Oracle Server Standard and Enterprise Editions, Oracle Real Application Cluster (RAC), Oracle Java Developer, Oracle SQL Developer, Netbeans, Eclipse, ESX VMWare virtual hosts, CentOS Linux and Windows Servers, IBM Rational SW Architect, IBM Requisites Pro, IBM Rational Rose Enterprise, CVSNT and SVN version control systems, free DotProject and proprietary MS Project management tools. We have seen fruitful experiences with commercial Web hosting for student projects at local and international companies (mostly in Canada and the US). Moreover, local companies actively supported student projects in many cases for free.

SE places a great emphasis on abstraction, modeling, information organization and representation, and the management of change. SE also includes implementation and quality control activities. A central challenge of SE is still the kind of decision-making known as engineering design. An important aspect of this challenge is that the supporting process must be applied at multiple levels of abstraction. An increasing emphasis on reuse and component-based development hold hope for new, improved practices in this area [1].

In the last few years we increased attention to the security solutions within student projects [12, 13, 15]. Security and data protection projects are already in the list of possible projects. Two years ago, a security SD project was recognized as one of the best student projects in CoSc224/319 courses in the Computer Science department (CoSc). After the internal project presentation there was self valuation. All of the group members were evaluated anonymously by each other and themselves. After each final presentation we collected informal feedback from the OC's staff and visitors.

3. STUDENT PROJECTS

In this section selected successful student SD and SE projects in the last four years and some initial projects, which are still under development are described. The problems we encountered and what we learned from them will be discussed as well. Our recommendations will be given in the Summary and Suggestions for Improvement section.

3.1 Okanagan Business Students Association Marketplace

This project was developed by CoSc 310/319 students in 2007/2008.

In 2007 we had a request to develop a Web-based Market Place for the Okanagan Business Students Association (OBSA). Students from the Bachelor of Information Systems program selected this project from a list of about 10 other projects and selected the Oracle XE/APEX platform for the development, IBM Requisites Pro as a requirements management tool and IBM Rational SW Architect (IRSA) as a design tool.

The students had no prior experience with Oracle APEX, but already studied COSC 126 (Systems Analysis and Design), COSC 236 (Object-Oriented System Analysis and Design), COSC 304 (Introduction to Database Management Systems), COSC 305 (Project Management), COSC 404 (Database Systems II), COSC 416 (Topics in Database), COSC 434 (Database Administration) and some other related courses. We found, that students managed to deal with the technological problems relatively easily in this project, but had many management and organizational problems. They established very good contacts with the sponsors of the project (OBSA) and still provide good support for the project after the final delivery to the sponsor. The first prototype of the system was successfully used in spring 2008, before the final project presentation and is used this year as well.

On the other hand, some students had many problems with the design and documentation development, especially with the Developer's Guide and Configuration Management documents.

From this project we've learned, that our students need more help in project management and design than in SW development and sponsor communication. Conflict resolution is especially important in the SW development team during the first few weeks of the project. The project milestones should be set by instructor in the beginning of the projects and should be strictly controlled.

Another positive result, which was learned from this project, is the importance of the sponsors for the student's success. In our opinion, the local student organizations or other local projects are probably the best sponsors for the student's projects (see here: http://142.23.93.232/apex/f?p=110:1:920286383447658::NO::::)

3.2 Building Small Business Human Resources Resource Guide

This Web-based system is very intriguing from the educational point of view, because the system was initially developed alone by a single COSC 310 student for a large Government project in partnership with the permanent Small Business Roundtable, the BC Chamber of Commerce, Retail BC, the Canadian Federation of Independent Business, Okanagan College and the Ministry of Small Business and Revenue. The Resource Guide Web-based system was developed to assist small businesses in British Columbia with recruiting and retaining skilled workers (see here: http://www.bsbhr.org/apex/f?p=104:1:2718307338180512)

The project was started at the same time that CoSc 310 students selected a SE project from the course project list, and it was developed in parallel with the main course project by only one student in his free time. The learned SE principles and methods in the CoSc 310 as well as development environment are used successfully in the external business project. In a few months, several other students joined the project as well.

Why is this intriguing? Why should we discuss this project?

There are several positive impacts of this project that could be summarized in the following list:

- Students started to successfully use their knowledge, experience and skills, gained in the SD and SE courses in the middle of the course projects. Also, the external project experience helped students finish the course project successfully.
- Students developed additional stimulus in SE study, which increased their interest in the Computer Science education.
- At the end of their education at the College, students accomplished several external or internal projects and have obtained professional references from their sponsors.
- Students recognized the importance of the SD and SE project courses with external and internal sponsors at the CoSc.

Currently this project is supported by other students from the BCIS program.

3.3 Current Student Projects

Taking into account our past experience, this year the list of possible projects with external and internal sponsors was presented to the students in the SD course. Three project groups were organized by students themselves (3-4 members in the group). Two LAMP projects for small business companies in the Okanagan area and one Web-based system for a sponsor in other area of Canada were chosen by students from the list of 10 possible projects.

Within the first few weeks, students recognized some organizational problems with the sponsors in two projects and requested permission to change their projects. In one case, the communication problem was a reason, but in the other case the company already found another development team. The permission was granted by the instructor and students chose other projects – one of them with a strong research component. In this subsection we will discuss two projects only, because the third project was a very small LAMP project, which does not have any specific topic for the discussion.

One of the three current projects is the “Foreign Language Learning System, Oratio” (for more information see here: https://sourceforge.net/projects/oratiopr/). The sponsor of this Web-based system is a European world renowned linguist, who would like to implement the latest achievements in foreign language learning into free open source SW development projects. The sponsor was impressed by the student's projects, developed in the past at the Okanagan College. In addition, students seemed to enjoy developing a system that would be useful for other people and would be further developed by themselves in commercial projects or by the open source communities.

The goal of this project is to allow learning of different foreign languages in a different ways by using a Web-based system, free of charge and with customization options. It aims to accomplish three things that are difficult to achieve by any techniques of language learning: catching attention, intriguing the learner, and making him/her stay in the learning environment.

As it was said before, originally the project was started in the Oracle APEX environment (by the sponsor’s request), but in the middle part of the development, the group made a decision to change the development platform to LAMP. The reasons were a lack of experience with the Oracle APEX, not enough time to study new topics, and a previous development experience with LAMP. Such situations had happened in the past several times. In many cases students managed to finish development in time, especially if they already had the project documentation and system’s design established. If this project is successfully completed as it is currently planned by students, the impact of this system in foreign languages learning could be recognized by the community and society, especially in a country such as Canada, where two languages are used officially and many other groups of people use their native language and need help studying other languages.

Another example of the current students’ projects is the “SW Development Project Management System”, which was chosen by the CoSc 224 students and sponsored by the CoSc staff.

The project goal is to design and develop a flexible, scalable, customizable, free SW Development Project Management Web-based system, based on Oracle EX/Oracle APEX platform. We already had a request from industry to develop a SW Development Project Management Web-based system as well. Local companies expressed dissatisfaction with available proprietary or open-source project management systems. In the list of possible projects for students we had very similar projects, but with different sponsors (one external and one internal) and with different functional and non-functional requirements. Students chose the SD project with the internal sponsor.

This system is given as an example, because it shows, that students like to see computer science as a social, relevant, important, and caring endeavour. The projects should “require teams to create real systems for real clients”, as is said in [16].

This project is still under development and the student group wants to continue the project later, in the SE course at the College.

4. Summary and Suggestions for Improvement

In this section we summarize and analyze the suggestions for improving the teaching and delivery of our SD and SE classes.

Some student projects were already successfully integrated into other research, businesses or development projects in Canada (BC and Alberta), Europe (Poland) and the US (NY, California).

The recommendations for teaching SD and SE classes would be summarized as following:

- Dealing with real external or internal sponsors is a very important stimulus for students, which increases student activity and innovation in the projects. Students enjoy projects where they gain real practical experience working with sponsors. Especially we had impressive results when the sponsors were students from a different program. When students initiated the project by themselves with one of the students as a part of the three sponsors, the group made a decision to switch their project with another one. We recommend that students should not be a sponsor or co-sponsor during the development of a project.

- The model-driven iterative development, including configuration management, user and development testing, and documentation development are very important components of the SD and SE projects, especially with real sponsors. Students tried to avoid models and documentation development or postpone them until the end of the project. This is a challenge for the instructor to manage and control model-driven iterative development and documentation development starting from the project's first iteration.

- The internal project presentation is an effective way to prepare students for the final public presentation of their work and for their self-evaluation. During the internal project presentation students should test other students work for bugs and functional problems.

- Some of problems in the student projects were related to team management (especially in the first weeks of the projects), documentation development, and sometimes with sponsor communications. The service courses from other departments (in our case from the Business and Communication departments) should be selected and synchronized with the projects courses very carefully. Several projects were changed for other projects from the list of possible ones, because students had communication problems with the sponsors.

- In projects with real sponsors students very often are looking for immediate implementation of their new designs, and development and management skills. We should support such trends in student behavior. Some students have achieved high recognition from the local IT companies and some of them were offered mid-level professional positions with a relatively higher salary compared with other graduates. In the last few years we had a queue of requests for our students from local business companies, which are ready to give financial support to students to employ them during their study and after graduation.

- The list of possible projects with external sponsors should be given as early as possible. We are now discussing the idea of giving a list of possible projects with external sponsors in the Object-Oriented Systems Analysis & Design course (CoSc 236), which is taught before the "Projects in Computer Science" (CoSc 224). Typically, the CoSc 236 is a challenging and unpopular course with students, because it requires a lot of reading, abstract thinking, designing and modeling in UML. It also studies new industrial modeling and SD applications and platforms (IBM RequisitesPro, IBM Rational Rose Enterprise, etc). In the new approach, students will start to design a system with the sponsors in the fall and develop the system during the winter in the CoSc 224.

- As mentioned in other research papers [7], a number of student projects suffered from a lack of internal leadership (someone responsible for calling meetings, communication with instructors, checking schedules etc.). We think that it is a good idea for the instructor to select group leaders during class and not by the students themselves.

- External sponsors should be selected very carefully. Instructors should develop requirements for the external sponsors (something we already implemented this year). The main requirement is the time availability to work with students and help them with project design, documents development, system testing and evaluation. Sponsor has to be ready for weekly meetings with students personally or on-line (or work through the instructor) at least 1 hour a week. In the past we had several situations where sponsors were unavailable for several weeks or more.

5. CONCLUSION

In this paper we presented the experiences and some results from four years of teaching "Projects in Computer Science" in the Computer Information Systems (CIS) Diploma program and "Software Engineering" in the Bachelor of Computer Information Systems (BCIS) program at Okanagan College. We discussed goals and objectives of our SD and SE courses; evaluated two successful past projects as well as two projects under development. Additionally, we summarized and analyzed the suggestions for improvement of teaching our SD and SE classes.

Dealing with real sponsors requires a slightly different approach, and extra workload and responsibility for the instructor as well as for other staff. Additional attention should be give to appropriate support from institutional IT Services, from staff and other support departments.

In conclusion, the SD and SE project courses with external and internal sponsors drastically increased learning activity, creativity, productivity and success of students work. Students like to see computer science as social, relevant, important, and caring endeavor [16].

6. ACKNOWLEDGMENTS

The author appreciates the support given to him by the CoSc staff in the program development, student project sponsorship, projects evaluation, and for many recommendations in the last four years. Also the author would like to thank his colleagues: Rick Gee for many hours of discussion about problems related to the student projects and Alan Kennedy for help with writing of this paper.

The author likes to thank the reviewers for their valuable comments, which have helped to improve this paper.

7. REFERENCES

[1] Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, The Joint task Force on Computing Curricula, IEEE Computer Society, Association of Computing Machinery, August 23, 2004. http://sites.computer.org/ccse/SE2004Volume.pdf

[2] Okanagan College 2006/07 Calendar, Course Descriptions, University Studies, Computer Science. http://www.okanagan.bc.ca/calendar/course-descriptions/university_studies/computer_science.html

[3] http://en.wikipedia.org/wiki/IBM_Rational_Unified_Process

[4] http://en.wikipedia.org/wiki/Software_development_process

[5] http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle

[6] http://en.wikipedia.org/wiki/Rapid_Application_Development

[7] Petkovic, D., Thompson, G. and Todtenhoefer, R. 2006. Teaching practical software engineering and global software engineering: evaluation and comparison. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, 294-298. DOI= http://doi.acm.org/10.1145/1140124.1140202

[8] Boehm, B. W. 1979. Software engineering. In *Classics in Software Engineering*, E. N. Yourdon, Ed. ACM Classic Books Series. Yourdon Press, Upper Saddle River, NJ, 323-361.

[9] http://en.wikipedia.org/wiki/Software_engineering

[10] Khmelevsky, Y. and Dhanjal, S. 2007. Information Security and Data Protection in Computer Science Education. 12th Western Canadian Conference Education on Computing Education (WCCCE-2007), Thompson Rivers University, Kamloops, Canada, May 3-5.

[11] Khmelevsky, Y.M. and Dhanjal, S. 2004. Computing Science and Information Technology with E-Business Orientation", 8th World Multi-conference on Systemics, Cybernetics and Informatics (SCI2004), Orlando, Florida, U.S.A., July 18-21, 393-398.

[12] Khmelevsky, Y.M. and Dhanjal, S. 2004. "E-Business Technology education: a preliminary model", First International Conference on The Internet Society: Advances in Learning, Commerce and Security (formerly E-Business 2004), Skiathos, Greece, May 12-14 (published in hard back by WIT Press UK 2004, ISBN: 1-85312-712-4) in the Book Title: "The Internet Society".

[13] Khmelevsky, Y. 2008. Information and data protection within a RDBMS. Condensed Matter Physics, Vol. 11, No 4(56), 761-765.

[14] Khmelevsky, Y.M. and Ustimenko, V. 2003. Practical aspects of the Information System reengineering. The South Pacific Journal of Natural Science, USP, Suva, Fiji Islands, Vol. 21, 75 - 81., ISSN 1013-9877 (printed) ISSN 1726-0787 (online) S.Pac.J.Nat.Sci.

[15] http://www.extremeprogramming.org/

[16] Buckley, M. 2009. Computing as Social Science, Changing the way computer science is taught in college by encouraging students to develop solutions to socially relevant problems, Communications of the ACM, Vol. 52 No. 4, 29-30. doi:10.1145/1498765.1498779

Okanagan College and Vancouver Island University Educational Joint Projects Results

Youry Khmelevsky, Ph.D.
Computer Science Department
Okanagan College, 1000 K.L.O. Rd.
Kelowna, BC, V1Y 4X8, Canada
+1 (250) 762-5445 ext. 4741
ykhmelevsky@okanagan.bc.ca

Michael Govorov, Ph.D.
Advanced Diploma Program in GIS
Vancouver Island University
900 Fifth Str., Nanaimo, BC, V9R 5S5
+1 (250) 753-3245 Ext 2729
michael.govorov@viu.ca

Leif Burge, Ph.D.
Department of Geography & Earth
Okanagan College, 1000 K.L.O. Rd.
Kelowna, BC, V1Y 4X8, Canada
+1 (250) 762-5445 ext. 4280
lburge@okanagan.bc.ca

ABSTRACT

This article summarizes collaborative educational activities and joint educational program development between the Computer Science department and other departments at the same and other institutions. We hope that this article will aid Universities, Colleges and academia in achieving success in developing joint projects. The authors discuss the inter-institutional challenges of developing and deploying shared courses, and the benefits and challenges of new course development. Additionally, the paper describes the goals and results of three joint projects between Okanagan College (OC) and Vancouver Island University (VIU, formerly Malaspina University-College, BC, Canada), and presents information about the Nation-wide GIS Project in Lithuania, which was conducted by the VIU to develop national Spatial Data Infrastructure. Two of the four projects, mentioned above were developed for Okanagan College, while the third is the BC Campus Project: "Post Graduate Technical Diploma in GIS at the Malaspina University-College (lead), University of British Columbia (UBC), and Okanagan College" for the Vancouver Island University (VIU).

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - *Relational databases*

H.2.8 [**Database Management**]: Database Applications - *Spatial databases and GIS*

K 3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education*

General Terms

Management, Performance, Design, Experimentation

Keywords

Geographical information systems, Computer science education, Projects, ESRI, ArcSDE, ArcMap, Oracle Academic Initiative (OAI)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

1. INTRODUCTION

A GIS, or Geographical Information System, captures, stores, analyzes, manages, and presents data that is linked to location. In the strictest sense, the term describes any information system that integrates, stores, edits, analyzes, shares, and displays geographic information. In a more generic sense, GIS applications are tools that allow users to create interactive queries (user created searches), analyze spatial information, edit data, maps, and present the results of all these operations [1].

Geographic information science is the science underlying the geographic concepts, applications and systems and is often taught in degree and GIS Certificate programs at universities [1]. However, many universities and colleges introduce GIS [2] or Geographic Information Technology [3] in courses taught through Computer Science (CS) departments.

The topics of the discussion are the upper level CS elective CoSc 416 "Topics in Database: Foundations of GIS and Geodatabases" (GIS) course and Post Graduate Technical Diploma in GIS GEOG 511 "Introduction to Geodatabases" course at VIU. To support the new inter-institutional courses development, we applied for support to the Professional Development (PD) Fund at OC in 2005 and 2006.

In the next sections, we will outline the project goals, project plans and projects results for the two projects, funded by OC's PD Fund. Then we will present an overview of the BC Campus Project. Finally, the Lithuania Spatial Data Infrastructure Project will be discussed.

In the conclusion we will summarize our achievements, discuss possible implementations of our results at other institutions in BC and country-wide, the inter-institutional challenges of developing and deploying the shared courses, and the benefits and challenges of the new course development, opportunities in the area of interdepartmental and inter-institutional projects in Canada, North America and Europe.

2. Project Goals and Accomplishments

Computer Science (CS) is not only an area of study in its own right but it is also an important supporting area for many other disciplines [2]. Studies have shown that failure to motivate interest in science by establishing its relevance to the students' lives and personal interests is a significant factor for students who had an initial intention and the ability to major in a science field but instead switched to nonscientific fields [2]. In our opinion, we can and we should design curricula where CS students will learn how to apply their computer science content knowledge to an application area.

On the other hand, we recognize (from other studies and [3, 4] our practical experience [5-7]) that the success in Computer Science education can be achieved when our students understand the application of the Information Systems into the business environment and utilize their skills for business problems. It has been recognized that this occurs "*when business administration concepts are integrated with computer science technologies and software engineering principles*" [4]. This is achieved by using the cross-disciplinary study model. In the paper mentioned above, the authors raise a question about crises in Information Systems education. We agree that graduates need a comprehensive understanding of behavioral aspects as well as Software Engineering, Programming and Information Technology to be most successful.

Since 2005 we've conducted three joint projects in Canada between OC and VIU, two of them were funded by the OC's PD Fund and one of them by BC Campus:

- In 2005, "Geographic Information Systems (joint project between Malaspina University-College and Okanagan College" (First Project)
- In 2006, the "Geographic Information Systems New Course Development joint project between Okanagan College and Malaspina University-College" (Second Project)
- In 2006, the "Post Graduate Advance Diploma in GIS Applications at the Malaspina University-College (lead), University of British Columbia, and Okanagan College" (BC Campus Project) was funded by BC Campus.

2.1 The Main Projects Goals (First and Second Projects)

In the First Project, we planned to collaborate between two educational organizations and in two different educational areas of CS and Geography. Specifically, the areas are database management systems (DBMS) and the application area of the DBMS in GIS within the Bachelor of Computer Information Systems (BCIS) program in the CS department at OC (see more here: http://www.okanagan.bc.ca/departments/science/computer-science/Programs_-_detailed_descriptions/BCIS_degree.html) and within the Advanced Diploma in GIS Applications program at VIU (see here: http://www.mala.ca/adgisa/).

The Second Project was intended to establish collaboration between the Computer Science and Geography departments at OC to provide professional training in Computer Science, Information Technology, and Geography to the college diploma and undergraduate students with the help from the Advanced Diploma in GIS Applications at VIU.

2.1.1 The First Project Goals

- To organize a joint GIS education for OC students by developing advanced database courses in CS and for other departments. Published materials and instructional courseware packages will be provided by VIU for the GIS content.
- To incorporate a select number of course modules, available through Environmental Systems Research Institute, Inc. (ESRI) [1] Virtual Campus into the delivery of specific courses to enable students and instructors to keep pace with changes in the world's most popular GIS package – ArcGIS.
- To evaluate and simplify the transfer of the BCIS students to VIU.

2.1.2 The Second Project Goals

- To implement the results of two projects (First Project and BC Campus Project) by developing curriculum in the Computer Science and Geography Departments at OC.
- To develop and implement three new or revised undergraduate GIS courses, including two in Geography and one in CS.
- To explore how two courses, "Foundations of Geographic Information Systems" and "Geo-databases", could be developed based on materials from the BC Campus Project and cross-credited to courses in VIU's GIS Applications program in the future.
- To explore how OC students may continue study at Malaspina after completion of the CIS diploma or BCIS degree.

2.2 The Main Projects Deliverables

2.2.1 For the First Project

- New GIS-related modules for an Advanced Database course were developed within the project and delivered to the students at OC by instructors from both OC and VIU. New Spatial Database Engine (SDE) modules for a Malaspina University-College course (GEOG 511 Introduction to Geodatabases) was developed and delivered at VIU by an OC instructor.
- The course modules were delivered to the students of OC and VIU in the Fall 2006 by the instructors on both sites. OC's instructor taught ArcSDE Database creation and administration in both campuses and VIU's instructors delivered GIS and spatial database modules to OC. Each party delivered 12-hour modules.
- Exam components were marked by all participants.

2.2.2 For the Second Project:

- Three new or revised GIS courses were developed within the project and delivered to the students at OC by OC's instructors (one from CS and one from the Geography department). VIU's instructors were provided support in course development.
- New Spatial Database Engine (SDE) modules for VIU's GEOG 511 Introduction to Geodatabases course was developed and delivered at VIU by an instructor from OC.
- The "Introduction to Cartography, GIS and Remote Sensing" course was delivered to students at OC by the Geography department in the Fall 2007.

Later, VIU developed the "Nation-wide GIS project in Lithuania" for Europe (Lithuanian Project). The main goal of that project was to develop a Nationwide Spatial Data Infrastructure for Lithuania. The Lithuania's National Lands Service contracted Vancouver Island University to develop nine GIS courses and to provide civil servants with a set of geomatic tools and techniques that would strengthen Lithuania's geographic information infrastructure at local and national levels (more information about this project can be found here: http://www.viu.ca/sdi/index.asp).

3. The Experience of Delivering the Courses

When we started the First Project, we had no experience in developing and deploying inter-institutional, shared courses in Canada. We had experience from Europe and Fiji [5-6]. We had a lot of challenges and obstacles, especially in the first year of the project. Fortunately, in the next year we received support from the BC Campus for a large project at VIU, which was successfully integrated into small projects at OC.

3.1 The Experience within the First Project at OC and VIU

For this project, we developed a new optional upper level CS course, CoSc 416 "Topics in Database: Foundations of GIS and Geodatabases". Within CS, we implemented a new technological environment and deployed new SW applications from the ESRI Company: ArcGIS, ArcView and ArcSDE for the BCIS program students. Published GIS materials and instructional courseware packages were provided as support for GIS content by VIU. A select number of course modules, available through ESRI's Virtual Campus, were incorporated into the delivery of the courses. Use of these modules enabled students and instructors to keep pace with changes in the world's most popular GIS package – ArcGIS.

More specifically:

- New GIS-related modules for the CS's CoSc 416 "Topics in Database" course were developed by instructors from OC's CS department and from VIU. New Spatial Database Engine (SDE) modules were developed by OC and VIU instructors for the GEOG 511 "Introduction to Geodatabases".
- Introductory lectures were developed and delivered to OC students by VIU instructors to CoSc 224 and CoSc 236 students in the Fall 2006 and Winter 2007.
- The "ArcSDE Administration" lecture module was delivered to VIU students in the Fall of 2006 by an OC instructor.

The project budget covered the following items: ESRI software licensing sharing cost, conference participation and more importantly, the instructors exchange visits for the delivery of new courses. During the First Project we organized two visits to VIU, where an instructor from the OC CS department delivered a new technological module to the students in the Post Graduate Advanced Diploma in GIS. The instructors from VIU visited OC two times to deliver lectures for the Computer Information Systems (CIS) and BCIS programs. During our exchange visits we found that the students from the Post Graduate Technical Diploma in GIS Applications were very interested in practical topics and administration aspects of the ArcSDE (currently a component of the ArcGIS Server [7]) and the CIS and BCIS students were very interested in the special database systems applications, such as GIS and spatial DBMS (or Geodatabase) [8]. To collect students' opinions, we used informal interviews within CoSc 224 and CoSc 236. Two years later, students from the mentioned above courses enrolled into the developed CoSc 416, during the Second Project phase.

The last goal of the First Project: to simplify the transfer of the BCIS students to VIU after their completion of the CIS diploma or BCIS degree is still not implemented. We succeeded in introducing a new optional learning topic in CS and introduced a new area of study for BCIS students after graduation at VIU in traditional or on-line mode. We believe these goals were achieved successfully in the First Project.

3.2 The Experience within the Second Project at OC and VIU

In this project we concentrated on the curriculum development in the CS and Geography departments at OC as well as on technical modernization in CS for the advanced courses.

We successfully developed and implemented three new undergraduate GIS courses, including two in Geography and one in CS:

- GEOG 272 "Introduction to Cartography, GIS and Remote Sensing" as an introduction to working with maps and spatial data was delivered once.
- GEOG 374 "Foundations of Geographic Information Systems" was developed to introduce geography and CS students to GIS and spatial analysis and was delivered once.
- CoSc 416 "Topics in Database: Foundations of GIS and Geodatabases", which is the advanced course for the CoSc and Geography students. The CoSc 416 was delivered in CS two times.

In addition, we accomplished the following content development for the Second Project:

- New Spatial Database Engine (SDE) modules for VIU's "GEOG 511 Introduction to Geodatabases" course were developed and delivered to VIU by an OC instructor.
- The published materials and instructional courseware packages were provided through the BC Campus Project for GIS content. Instructors from VIU provided support in course development and took part in the exchange visits with OC.
- A course module, available through ESRI's Virtual Campus, were incorporated into the delivery of CoSc 416.

The ESRI's ArcGIS and ArcSDE applications as the main software packages in the delivery of these courses were purchased and deployed. We solved the problem of the sharing license cost between several OC departments, which use ESRI's application in the undergraduate courses.

After the delivery of the first course in the Winter of 2008, some other students from the same program who didn't select the new course in the Winter of 2008 requested the new optional course again in the Fall of 2008.

4. THE BC CAMPUS AND THE NATIONWIDE GIS PROJECTS IN LITHUANIA

In 2005, it was identified that access to instructor-led online education in GIS Applications is limited provincially and globally. According to the feedback received, coupled with labor market information, we confirmed that there is a high level of interest in credit-based GIS education. There exists a gap in online credit programming for GIS application specialists and for this reason VIU, UBC and OC have established a collaborative working model to develop online programming in an advancing field of technological application.

Since 2005, through the application of BC campus funds, the institutions have received the resources to support the development and enhancement of an existing GIS Applications Program for online delivery.

VIU instructors and the partners designed and developed 10 high quality, learner-centered interactive online GI-science courses. The instructional strategy provides opportunities for learners to check their understanding as they progress and receive immediate feedback, apply the knowledge and skills they have learned, and connect new concepts with prior learning as they share experiences and build knowledge together with their colleagues in online learning communities.

The online post graduate diploma in GIS program is a three-semester program in Geographic Information Systems (GIS). The program is comprised of 10 courses for a total of 400 instructional hours and 30 course credits. All ten courses are delivered 100% on line. The goal of this program is to provide an online technical education in GIS.

The program begins with a course in GIS foundations, which serves as both a refresher in GIS technologies and an overview of the themes that will be discussed in the program as a whole. The program then continues with a course in Geodatabases, which familiarizes students with the most advanced technologies used for spatial data storage today. Next, two courses in Spatial Analysis introduces students to spatial analysis concepts that are used in virtually all GIS applications (Spatial Analysis I), and then to specialized spatial analysis concepts that are used in application areas of study (Advanced Spatial Analysis II). The Remote Sensing course introduces students to the analysis and interpretation of satellite imagery, and how this technology serves as an input for GIS. The Spatial Positioning Systems course introduces students to technologies such as the Global Positioning System (GPS), which is used to accurately locate mapped objects and record spatial data in the field. Students are next introduced to the programming of Geographic Information Systems, with the Programming Foundations course that introduces students to Visual Basic for Applications, and a GIS programming course will help to develop skills and techniques in GIS programming and development with ESRI ArcObjects within VBA and .Net environments. Students then get an introduction to the management aspects of GIS with a course that emphasizes proposal design and writing, bid preparation, and project planning. The last course is an introduction to the presentation of Geographic Information on the Internet, as well as the design of Internet Map Websites. This course allows the students to summarize and present the findings of their previous courses.

The project partners from Okanagan College and UBC are involved in the Geodatabases course development, the program and individual course revision.

The first intake of this program will graduate in May, 2009 and the second intake of students started in February, 2009. Despite the high intensity and duration of this online learning strategy, we experienced only a 15% of loss rate during the first intake. We apply various Interactive Learning Objects throughout each course that address the following pedagogical issues: multiple learning styles; variety of learning domains; gradually testing higher levels of knowledge and skills; providing ways for students to interact with the content in relevant ways that simulates real world experiences; learner centric – empowering learners to check their own understanding, receive immediate feedback, identify areas of weakness which require additional review and/or practice; learner control – allowing learners to self assess – choose how often they complete a self assessment and/or an interactive learning exercise, in privacy and at their own time.

The methodology used to deliver the courses includes a combination of the following:

- Student learning through lecture and practical materials via the web-based learning portal (on Moodle and GIS Server), course book and assigned supplementary readings.
- Supervision of individual work is conducted by remote instructors via email correspondence and through web-based two-way forum support (on Moodle).
- Introduction seminar, initial surveying, and final exams are conducted online (Moodle). Introduction and orientation materials are also accessible from the web-based learning portal.
- Students use GIS and other software through the remote terminal on the GIS Server.
- Students complete and submit the results of assignments through the web-based learning portal (on Moodle).
- Media consultancies can also be done through the broadcasting software (e.g. Skype) or in face-to-face mode in a computer lab.

Based on the online instruction experience, VIU GIS instructors, with assistance from other GIS professionals, recently completed preparation of 9 GIS course modules for delivery to over 180 Lithuanian civil servants under the auspices of the Lithuanian Geographic Information Infrastructure (GII) training program [9].

The program, a combination of face-to-face and online curricula, was funded by European Union and the Republic of Lithuania. The 9-course program intended to prepare GIS professionals for the development of a national geographic information infrastructure. A broad range of subjects included technical courses in Geographic Databases and Web Programming, and issues-based courses such as GII Standards and Technical Infrastructure. This program is customizable for any government organization.

Delivery of the program has been carried out by GIS professionals in Lithuania over the 12 months in 2007-2008. Graduates from each of the 9 courses receive a “Certificate of Completion” from Malaspina University-College.

The methodology used to deliver the GII courses includes a combination of the following:

Table 1. Structure and sequence of the GII course delivery

Classrooms	**Online Home Work**	**Classrooms**
1	**2**	**3**
Surveying and Initial Evaluation Orientation and Introduction	Reading Laboratory Assignments and Tutorials on GIS Server Portal Forums Chats and Announcements ESRI Campus Virtual Courses Mid-term Exam	Final Exam

Classrooms ⇒	Seminar(s) Online and/or face-to-face consultancies	

The concept of using on-line modules has several advantages compared to the traditional university courses:

- Possible extension of the modules over a longer time span for full time employees;
- Flexibility to choose only appropriate parts for students who are already professionally active;
- Modules can easily be adapted to fit specific training environments (e.g. advanced geography, geodesy, environment, IT specialists, GIS users, governmental employees – planners and decision makers).

5. THE INTER-INSTITUTIONAL CHALLENGES OF DEVELOPING AND DEPLOYING THE SHARED COURSES

In our projects, we established very productive cooperation and developed new educational components for both programs at OC and in VIU departments.

What did we gain and learn in the projects discussed above?

- Students in both institutions enjoyed the visiting instructors from different institutions. CS students have obtained more information in the application area of CS, but also Advanced Diploma Program in GIS Applications students have obtained more information about GIS server side applications.
- We developed a strong basis for the collaboration between our institutions and departments in different areas of expertise.
- We gained experience and expertise, which helped us with other projects and helped the projects to be recognized not only in BC, but across Canada and internationally.

However, we did not realize all of our goals:

- We have not established effective interdepartmental collaboration at OC, in spite of very effective inter-institutional collaboration. This is partly due to the small size of the Geography Department and the limited number of students. Growth is expected in the area of GIS within the Geography Department as more options become available.
- As was mentioned in section 2.1.2., the last goal of the First Project "To simplify the transfer of BCIS students to VIU after the completion of the BCIS degree" is not yet implemented. We still are working in that direction.

6. CONCLUSION

In the present paper, we discussed four inter-institutional educational projects. Inter-institutional developing and deploying shared courses is a challenge for many institutions, especially within different areas of teaching. Although the paper focuses on developing and deploying a GIS program, the solutions can be generalized to other institutional and inter-institutional projects.

GIS is a challenging learning subject [2], but also a rewarding opportunity for both faculty and students, and it has proven to be an excellent course for a CS elective. The students gained a new knowledge and practical experience about GIS applications, Geodatabases and other related tools. Some students already had previous experience with GIS and expressed a desire for more GIS courses at BCIS programs. At least several students in BCIS program expressed their readiness to apply for the postgraduate program at VIU after finish the BCIS program at OC.

7. ACKNOWLEDGMENTS

Our thanks to Okanagan College Program Development Fund, College Administration, BC Campus program and Head of Computer Science Department of the OC for supporting us and our joint educational projects. The authors like to thank the reviewers for their valuable comments, which have helped to improve this paper.

8. REFERENCES

[1] Geographic Information Systems, http://en.wikipedia.org/wiki/GIS

[2] Zhang, W. and Beaubouef, T. 2008. Geographic information systems: real world applications for computer science. SIGCSE Bull. 40, 2 (Jun. 2008), 124-127. DOI= http://doi.acm.org/10.1145/1383602.1383650

[3] Zhang, M., Lundak, E., Lin, C., Gegg-Harrison, T., and Francioni, J. 2007. Interdisciplinary application tracks in an undergraduate computer science curriculum. In Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (Covington, Kentucky, USA, March 07 - 11, 2007). SIGCSE '07. ACM, New York, NY, 425-429. DOI= http://doi.acm.org/10.1145/1227310.1227457

[4] Helfert, M. and Duncan, H. 2007. Evaluating information systems and business informatics curriculum. In Proceedings of the 2007 international Conference on Computer Systems and Technologies (Bulgaria, June 14 - 15, 2007). B. Rachev, A. Smrikarov, and D. Dimov, Eds. CompSysTech '07, vol. 285. ACM, New York, NY, 1-5. DOI= http://doi.acm.org/10.1145/133059.1330676

[5] Sharma, P., Govorov, M., Khmelevsky, Y., and Dhanjal, S. 2004. Oracle 9iAS Portal as a platform for Geographic Information Science distance and flexible learning at the University of the South Pacific. First International Conference on The Internet Society: Advances in Learning, Commerce and Security (formerly E-Business 2004), Skiathos, Greece, May 12-14 in book: "Human Perspectives in the Internet Society", 489-500, ISBN: 1-85312-726-4

[6] Govorov, M., Khmelevsky, Y., Ustimenko, V., and Khorev, A. 2005. Security for GIS N-Tier Architecture, Book Developments in Spatial Data Handling, Fisher, P. (Editor), Springer Verlag, ISBN: 3-540-22610-9, 71-83.

[7] ArcGIS Server, http://www.esri.com/software/arcgis/arcgisserver/index.html

[8] Spatial database, http://en.wikipedia.org/wiki/Spatial_database

[9] Beconytë G., Govorov G., Ningal T.F., Paršeliūnas E., and Urbanas S. 2008. Geographic Information E-Training Initiatives for National Spatial Data Infrastructures, Technological and Economic Development of Economy, Baltic Journal on Sustainability, 14(1): 11–28, ISSN 1392-8619 print / ISSN 1822-3613. http://www.tede.vgtu.lt/upload/ukis_zurn/2008_01_psl_11_28.pdf

A Learning and Collaboration Platform Based on SAGE

Javier Delgado
Coll. of Engineering and Computing
Florida International University
Miami, FL, USA
1(305)348-4106
javier.delgado@fiu.edu

Mark Joselli
Instituto de Computação
Universidade Federal Fluminense
Rio de Janeiro, Brazil
55-21-88828788
mjoselli@ic.uff.br

Silvio Stanzani
Lab. of Architecture and HPC
University of São Paulo
São Paulo, Brazil
55-11-30915617
silvio.stanzani@yahoo.com.br

S. Masoud Sadjadi
Coll. of Engineering and Computing
Florida International University
Miami, FL, USA
1(305)348-3549
sadjadi@cs.fiu.edu

Esteban Clua
Instituto de Computação
Universidade Federal Fluminense
Rio de Janeiro, Brazil
55-21-26592646
esteban@ic.uff.br

Heidi Alvarez
C. for Internet Augmented Research
Florida International University
Miami, FL, USA
1(305)348-2006
heidi@fiu.edu

ABSTRACT
In this paper, we describe the use of a tiled-display wall platform for use as a general purpose collaboration and learning platform. The main scenario of emphasis for this work is online learning by users in different countries. We describe the general efficacy of this platform for our purposes and describe its shortcomings for this purpose empirically. We discuss its advantages and also the shortcomings that we found. We also describe an enhancement made to make it more viable for our target usage scenario by implementing an interface for a modern human interface device.

Categories and Subject Descriptors
K.3.1 **[Computers and Education]:** Computer Uses in Education – *collaborative learning, computer-assisted instruction, distance learning*

General Terms
Performance, Design, Human Factors

Keywords
Cyberinfrastructure, interdisciplinary, collaboration, e-learning

1. INTRODUCTION
Globalization has had a profound effect on the development process of many industries. For example, information technology businesses now globalize the development of software and even managerial positions are no longer limited to members in the home country of a company. The fast-paced growth of the Internet has been fundamental in this movement, as it has made remote collaboration tasks more efficient. However, many obstacles are still faced on a daily basis by workers in globalized environments. In order for future workers to be prepared for this kind of workforce, it is beneficial to get trained in a learning environment that utilizes the tools used by companies. In this work, we describe some of the challenges and requirements for this domain and provide an analysis of the usage of an integrated platform that can be used for collaboration and learning.

This paper describes our experience with SAGE as a learning and collaboration platform. We discuss ease-of-deployment, advantages and disadvantages compared to other solutions, and its general efficacy for this purpose. We describe two specific enhancements that we found necessary in order to make SAGE a more viable platform for collaboration. One was the remote desktop performance and the other was the human-computer interface, which we addressed by implementing an interface for a modern input device. To analyze the remote desktop performance, we performed tests to find where the bottleneck is and we report this. This project itself was carried out by members in different countries and different tools were used throughout our collaboration process, which allows us to provide a frame of reference for the shortcomings of other approaches, which were used extensively throughout this endeavor.

Several products have been released to facilitate collaboration activities that are carried out on a persistent basis by people in globalized workforces. These same tools can be used for remote learning. In this paper, we describe the use of a platform previously not fully exploited for these kinds of tasks. We use the Scalable Adaptive Graphics Environment (SAGE) [7] as our base platform. The SAGE software distribution was not necessarily designed for remote learning, but includes software that can be used for this task.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE '09, May 1–2, 2009, Burnaby, B.C., Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we present the learning and collaboration requirements. In Section 4 we describe the features of SAGE in terms of deployment as a collaboration platform. In Section 5 we give an overview of the efficacy of SAGE as a learning and collaboration environment. In Section 6 we discuss the remote desktop sharing. In Section 7 we present our input device implementation for SAGE. And finally in Section 8 we present the conclusions.

2. RELATED WORK

Skype [14] is an instant messaging application which provides teleconference facilities involving Skype clients, VoIP compliant devices and regular phones. Skype does not support video conferences involving more than two participants, which makes it impractical for lecture sessions; it is very effective for synchronous communication of project activities.

Enabling Virtual Organizations (EVO) [3] is a Java application which enables video conference meetings. EVO can be used to support lecture sessions and project group meetings. EVO enables a high level of interaction among remote participants by providing several collaboration modalities. A drawback to EVO, as with any application designed to write to a single display output, it the limit that exists in the output display resolution. As the number of remote participants increases, it becomes difficult to accommodate all the shared windows on a single display. Even if a large projector display is used, there may come a point in which the display area becomes saturated. In our GCB class, the lecture session consists of a professor and a group of four to eight students attending the class locally, and at least two remote sites with one to four students in each site attending the class via EVO. This results in at least three different cameras to be displayed on the local site (the local room and the rooms of each of the remote sites). In this sense, the learning interaction would be better if the students could count on the support of a tiled display wall, where all the cameras could be displayed, in high resolution, on a single large (integrated) display. This makes interaction easier as the number of remote participants increases. SAGE can fit this requirement.

3. Remote Collaboration and Learning Requirements and Facilitators

A collaboration and learning environment was created in the context of the Global Cyberbridges (GCB) program [4]. This program aims at enabling a collaboration infrastructure for e-Science using Cyberinfrastructure (CI). Currently, GCB consists of a number of distributed teams of two to three collaborators, with at least one person at each site in a specific team. The collaborators attend lecture sessions about special topics in high-performance and grid computing, and participate in subgroups where they conduct research and perform experiments alongside Cyberinfrastructure (CI) research scientists. The requirements for collaboration and learning were modeled based on the experiences of the GCB project.

The collaboration in GCB involves the following activities: a semester-long lecture on scientific computing, project task planning, project discussion, reporting, and result publication. Considering that all the participants work in different physical locations, and in different parts of the world (where time zones are different), the support for synchronous and asynchronous communication is mandatory. As a result, the functional requirements are the support for teleconferencing, video conferencing, display sharing, and the support for asynchronous activities, such as web forums, email lists, web logs, etc. The non-functional requirements are the Internet connection quality and display resolution quality.

During the learning activity of GCB, synchronous and asynchronous e-learning communication tools are used. The combination of both improves the interaction level and keeps the students motivated, as has been shown in [6]. The synchronous e-learning communication tools, such as video conferencing software, maintain a good interaction level, because it is possible to monitor the receiver's reaction to a message, improving the interaction. The asynchronous e-learning communication tools such as web forums increase the ability to process information, because the person involved in the learning activity can read, comprehend the message, research the topic, and then answer, ask more question(s), and/or make comment(s) [6].

Having taken the class and worked on the project, we were able to evaluate several communication technologies [5]. Teleconferencing tools are used for project planning and discussion and the message forum was used to discuss the results and specific issues on the tasks assigned to each participant. As the forum tool is used to keep track of activities, it can also be used to evaluate the progress of the project. The tools used for the support of synchronous and asynchronous communication are *Moodle* web forums [10], *Skype* for teleconferencing and instant messaging and EVO teleconference and video conference support.

The non-functional requirements comprise Internet connection quality and display resolution quality. The former directly affects the quality of interaction performing the synchronous collaborative tasks. The latter can improve the learning quality, which can take advantage of the quality of media used to support learning. In the context of the GCB experience, the Internet connection quality represented a tremendous drawback in the interaction. Desktop sharing, for example, was rarely used. Instead, the presentations (which were the most typical shared applications) and other materials were sent to the audience prior to the meeting. During the presentations, lecturers had to consciously dictate their position in the document (e.g. slide number in a presentation).

3.1 Specific Examples

3.1.1 Distributed Lectures

Online courses have become very popular, but with current commodity technology, conducting these lectures can still be burdensome for the lecturer. It is often necessary to use multiple tools at once. For example, separate tools may be needed for remote desktop sharing and for transmitting video of the lecturer and/or blackboard. Lecturers should ideally be able to see their entire audience (including those in other sites); this allows them to see if someone has a question or to observe facial expressions. For this, having an integrated teaching platform is beneficial.

It is typical to display graphics when teaching. In many cases, high resolution is not needed; for example, when showing flowcharts. However, there are cases in which higher resolution may help the audience to better analyze the graphics being shown, e.g. showing aerial images of a city for a lecture on geography. If these types of images are shown by means of a projector, image quality suffers. A solution to this is to use a tiled display wall, which can be composed of standard (high-resolution) LCD monitors.

Another issue is scalability. As the remote audience size increases, more display space is needed for the lecturer to be able to see them all. Ideally, the audiences from all remote sites should be viewable from a single display. It also helps if the lecturer is able to control the entire wall from one place. Controlling the wall should require as little effort as possible.

3.1.2 Project Collaboration

Computer science classes typically have components in which a number of students collaborate on a project. In a class with a distributed audience, such as the GCB class, the members of each team will end up in different parts of the world. This is where the need for efficient collaboration tools is most apparent. The team, which may consist of students of different disciplines, not necessarily computer related, benefits from having a means of efficient collaboration using intuitive tools. Many of the same features described for lecturers are critical: high-resolution display, friendly interface to interact with whatever is being displayed, etc.

4. EVALUTATION OF SAGE FEATURES AND DESIGN

A collaboration platform should allow its users to efficiently and easily collaborate. This is seldom the case. Many integrated solutions exist for collaboration, but they are still prone to errors that make them difficult to use and operate. It is often necessary to have an experienced technician present during a “meeting” to set up, monitor, and troubleshoot hardware and software related issues. Some of the issues we faced with the software that we were previously using were broken remote desktop sessions and inexplicable connection problems.

SAGE is a large scale visualization platform developed by the Electronic Visualization Laboratory at the University of Illinois at Chicago. The desktop sharing, live video streaming, and high resolution image displaying software that comes with SAGE allows it to be used as a powerful tool for the support of synchronous collaboration and learning. SAGE provides the following key benefits:

- (*Scalable) High-resolution*. SAGE is one of the tiled-display wall deployment technologies currently available. Multiple screens from different systems can be joined together to form one large display. A novel feature of SAGE that differentiates it from, for example, XDMX [2], is that it allows applications to be written such that each system renders its own part of its display, while still allowing multiple applications to be open at a time. SAGE tiled displays have been shown to scale to over 50 screens [9] while maintaining good visual quality. This allows the viewing of multiple simultaneous shared desktops and/or video conference feeds, up close analysis of rendered images, and other collaborative tasks. In Section 3.1.1, it was explained how, as the size of the remote audience grows the display space needed to show the entire audience also grows. A scalable display modality makes it possible to stream video feeds for several audiences simultaneously, whereas the display area projected by a single projector can only project the audiences of 2 to 5 sites, depending on the size of the audiences. In our target environment, having a large amount of screen space allows users to view several applications at once.
- *Collaboration software included*. The SAGE distribution includes two video players capable of playing back streaming media; both can be used for video conferencing on a SAGE display. It also includes a Virtual Network Computing (VNC) client to allow collaborators to share their displays. This can be used for showing presentation slides, work being done on a remote user's computer in real time, etc.

4.1 SAGE Architecture

SAGE is separated into several components. It has a basic window manager, the “Free Space Manager” or *FSManager*, which keeps track of which applications (i.e. windows) are open and their location on the display. The fundamental difference between the *FSManager* and a traditional window manager is that the *FSManager* is capable of spanning multiple screens by mapping the physical “tiles” to specific sections of the wall. The communication is performed between the *FSManager* and “SAGE Receivers” which run on each node. If a node has multiple graphics outputs, it may run more than one tile, but only one receiver is required per tile.

Drawing functions in SAGE are limited to basic pixel and rectangle manipulation. Porting applications that rely on higher-level drawing functions, such as widget creation, filling enclosed regions of arbitrary shape, etc. to work on a SAGE display is not trivial. It is necessary to modify the application and/or the lower layer drawing libraries it uses.

To provide abstraction between individual SAGE applications and the window/display management, SAGE provides a separate interface, the SAGE Application Interface Library (SAIL). This component obtains pixel information from the application, determines which nodes need to draw which parts of the application's display and sends the appropriate pixel information to the SAGE receivers at those nodes. The SAGE receivers then send the pixel buffers to the tile(s). A message passing paradigm such as MPI can be used to specify which node is to render which part of the screen.

SAGE provides a distinct user interface model wherein the position, orientation, and size of the windows being shown on the screen can be changed from a separate system running one of their *UI clients*. These clients also handle actions generated by input devices. This decoupling of display from display interface allows multiple interfaces to control the same display. Also, multiple pointers may exist, which can be beneficial in collaboration scenarios.

4.2 Ease of Deployment

SAGE is available for the three most common desktop platform operating systems: Linux, Windows and OSX. Installation on Linux requires several third-party libraries, such as Simple Directmedia Layer (SDL), portable audio, and GLUT. These are generally not difficult to install for experienced system administrators. Packages are available in the repositories of most major Linux distributions, and binary versions are available for Windows. SAGE can either use the entire display of a system or just a portion of it. If the whole display is used, another system must be used for user interaction. This paradigm may be difficult for some users, but it is beneficial when the display wall becomes very large, since all the windows in the display are consolidated into one application window in the workstation. It also allows multiple users to manipulate the windows of the display wall.

SAGE also requires the QUANTA toolkit, which is also developed at EVL. The compilation processes for QUANTA and SAGE are similar to other typical software applications. SAGE may require post-installation configuration as well. For example, access to the graphical displays of the tiles through SSH is necessary.

4.3 Cost

A tiled display wall requires a considerable investment, especially if a cluster is not already available. Several LCD monitors are required. Mounting hardware needs to be bought or fabricated to place the monitors. As with EVO, no specific video nor audio hardware is necessary. The graphics hardware used for the tiles should be powerful for best performance.

4.4 Drawbacks

Since a different display paradigm is used for SAGE, standard applications (which are designed to write to the underlying graphics library, such as X11 or Win32) do not work without modification. To fully exploit the scalability of SAGE, applications must be rewritten to work in parallel (i.e. parallel rendering and parallel display).

5. EFFICACY OF SAGE FOR COLLABORATION AND LEARNING

In the previous section, we described some of the general aspects of SAGE that make it attractive for remote collaboration and learning. The next step in our process involved actually using it for remote collaboration tasks. The available software was adequate for collaboration, for the most part. The quality of video conferencing and high resolution image rendering were good. However, we found the performance of two key features to be lacking. One of the features that was lacking was the VNC client. The human interface device (HID) support of the platform also seemed to be lacking for the purpose of collaboration, particularly for giving lectures.

A high-performance remote desktop client is essential for our target environment. One typical use case is for applications that display data that is viewed by all collaborators, e. g. students in different parts of the world watching slides associated with a presentation. The VNC protocol was designed for simplicity [13] and as a result, its performance over wide area networks suffers for some applications [1][8]. Part of our research consisted of testing the SAGE VNC Viewer using a presentation application.

A high-performance remote desktop modality also makes it possible to show non-SAGE applications running on a different virtual desktop of the local host. Since there is no network propagation bottleneck, good performance is potentially achievable, as long as no bottlenecks exist in the translation from the local display to the SAGE display. This is also analyzed.

When working with large displays, traditional input devices can become inefficient and difficult-to-use. The most common input device for manipulating windows in regular workstations, the mouse, is particularly unsuited for tiled displays since there may not always be a surface to put it on and it becomes cumbersome with large displays. To address this, we implement an interface for a modern pointing device.

6. REMOTE DESKTOP PERFORMANCE

The computer screen is very effective for depicting information, but projecting one's thoughts onto a computer is often not. When collaborating remotely, the audience physically present is instantly able to see what the presenter is showing. Virtual collaborators are not able to do so unless a desktop sharing solution is provided. In terms of collaboration, the ability to create something once and redistribute it instantaneously is very efficient. Combining a SAGE desktop with a remote display paradigm furthers the efficiency. But it is important for the software to perform adequately. A speaker should assume that whatever depiction (e. g. a slide from a presentation) is being explained is visibly available at the collaborating institution(s). Since sharing presentation slides is the most common desktop sharing task, we use this as a benchmark for assessing the quality of the remote desktop experience in terms of collaboration. This application requires short bursts of high-bandwidth data transfer (e.g. to show quick animations, such as slide transition effects).

6.1 Existing Remote Desktop Solution

In addition to the already-mentioned shortcomings of the protocol, the SAGE VNC Viewer client is not optimized for performance. A single node renders the image and SAIL propagates the changes to the affected display nodes. As the window gets larger (i.e. uses more screens), the performance degrades. The performance issue is partially mitigated by employing the “tight” implementation by Constantin Kaplinsky [9] to reduce the amount of data transferred. However, protocol-related optimizations do not address the problem of scaling to many screens.

Compared to other remote desktop paradigms, VNC performs very well when showing the display of systems connected through high-speed (e.g. gigabit) links. VNC has been shown to provide faster performance than more sophisticated remote desktop protocols, such as NX, in some cases [11]. VNC is considered a “low-level” desktop sharing approach, since it is pixel based. The other type of desktop sharing approach (“high-level”) is based on processing drawing instructions. In [11], the authors suggest that “high-level” protocols can be slower than the “low-level” protocols since the instructions used (and/or the protocols themselves) were designed for local displays; the fact that low-level protocols do not have to synchronize the state of the desktop is one possible reason why this is the case. However, to achieve adequate performance over a wide area connection, it is necessary to overcome the shortcomings of the VNC protocol.

Another advantage of VNC is that the protocol is based on image transfer. This makes it easier to port it for use with SAGE, since the design of SAGE is also based on simple bit maps. NX, for example, which is an open source alternative remote desktop paradigm, uses a more complex design. It uses a proxy, *nxproxy*¸ at the server side, to compress messages and an agent, *nxagent*, at the client side that caches and displays the images. It displays the images in a window similar to the *XNest* [17] nested X server. Since *XNest* is essentially an X server that is a client of a parent X server, porting the NX client would involve porting most functions of the X Desktop API. Doing so would be time consuming and error prone.

The SAGE *VNCViewer* is based on a standard VNC implementation. It transfers its pixels to the SAGE display using a SAIL-defined function that transfers a pixel buffer from the application to the SAGE display. This is performed at a specified interval, which is given by the invoker of the *VNCViewer* at run time. By default, it is one frame per second. The VNC protocol is well-suited for SAGE since it is based on similar, simple drawing primitives such as rectangle drawing. Also, as with SAGE, it knows to update only affected regions of the screen. The only VNC feature that is not available in SAGE is the efficient handling of window "move" events [12].

6.2 Performance Evaluation

A series of tests were performed to gain empirical performance results when with the use of *VNCViewer* in different collaboration scenarios. As a benchmark, a presentation rendered using *OpenOffice.org Impress* was played back. Regular presentations consisting of basic text and graphics typically do not require a large amount of bandwidth. However, if there is some kind of animation or slide transition being used to help the audience grasp a topic, some information may be dropped by the remote desktop paradigm. This is especially true of "pull" models such as VNC, wherein updates to the frame buffer are requested periodically, depending on available bandwidth, rather than the server "pushing" a set of frames as is done with other protocols. Conversely, the push model can lead to de-synchronization between what a presenter is saying and what is actually being displayed. The reference presentation we created consists of several effects, including fading (of graphics and text), moving objects, and changing of background graphics. We specifically noted that VNC did not provide animations as smooth as NX did when used over a commodity broadband link. To measure the loss in display quality quantitatively, slow motion benchmarking as described in [11] was employed. In this technique, a remote desktop trial is run at a regular pace and then in slow motion. The slow motion case ensures that more information is transferred. In the end, the total bandwidth transmitted is compared between the two runs.

The following systems were involved in the experiments:

- *Mind*: A 16-node, dual Xeon compute cluster connected to a 15-tile SAGE display wall. Un-accelerated ATI Radeon Graphics cards are used in each node.
- *Mileena*: A Pentium 4 workstation using Gentoo Linux and *tightvnc* 1.3.9
- *Athena*: An Intel Core Duo workstation using Gentoo Linux and *tightvnc* 1.3.9
- *Simiano*: An Intel Core Duo workstation using Fedora Core Linux, version 9 and *tightvnc* 1.3.9.

In all cases, Mind was the system acting as the VNC client. *Mind* and *Mileena* are located in the same laboratory at Florida International University, but they are not connected directly to the same switch. *Athena* is located 22 miles away from the University and uses a Direct Subscriber Line (DSL) connection rated at 3 Mbps and 384 Kbps download and upload rates, respectively. *Simiano* is located in the University of São Paulo. For each test, regular-speed and slow-motion benchmarks were executed with Mind running a regular VNC client and with the SAGE VNCViewer at 1 fps and 20 fps. A video was created of the presentation being executed with automatic slide transitions every second, using *recordmydesktop* capturing at 24 frames per second. The slow motion video was generated by playing the video through *mplayer* at a rate of 1 frame per second and transfering the output to a separate file using the *yuv4mpeg* output. Bandwidth utilization was measured using *Wireshark*.

Figure 1 shows the average bandwidth utilized throughout the sessions. *Simiano* is not included since the performance was too slow. The average of five separate sessions were used, since the amount of data transfer varies due to there not being any specified update interval in VNC. It can be seen that when the VNC server is far away, a lot of data is lost, which results in poor remote desktop playback quality. Visual assessment agrees with this: the interactive features of the presentation were lost when *athena* was the VNC server. The slow motion benchmarks always transfer approximately the same amount of data regardless of the frame rate of the SAGE client. With non-local runs, a lot of information is discarded when there is a low frame rate on the SAGE client.

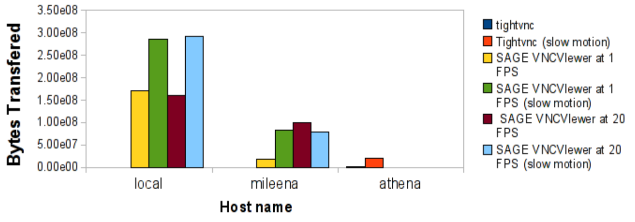


Figure 1. Amount of VNC-specific data transferred while playing a pre-recorded screen capture of our reference presentation. Data are shown for different hosts at regular speed and slow motion.

When comparing runs with different frame rates and with using the SAGE *VNCViewer* at regular speed, the amount of bytes transferred is not changed much in the local execution, whereas for remote desktop executions, having a faster frame rate results in more bandwidth being transferred. This was unexpected, since intuitively the lack of a network bottleneck on the local side would offset the advantage from having a higher frame rate. Conversely, when the VNC server was at a separate client system, several times more bytes were transferred when a higher frame rate was requested. This indicates an inability of the client to process all of the updates[1]. Another experiment was carried out at the University of São Paulo, with similar results. For this experiment, two systems connected through a LAN were used to measure the amount of bytes transferred during a VNC session. The following systems were involved in the experiments:

- *Simiano*: An Intel Core Duo workstation using Fedora 9 Linux Running SAGE Display on a single screen.
- *Willykit*: A Pentium 4 workstation using WindowsXP and RealVNC 4.1.2.

1 The VNC server pools update requests and combines them into one when it detects that the client is unable to handle each one

Figure 2 shows the amount of data transferred when using 1 and then 5 frames per second (FPS). Besides the larger initial peak for the 5 FPS run, there is no sign of it using more bandwidth overall. A test conducted afterward using a Secure Copy (SCP) transfer (Figure 3) showed that the link is capable of much higher bandwidth than was used by VNC, indicating a bottleneck at the client. The fact that CPU utilization increased supports this point. This indicates that VNC could benefit from a way of maximizing the bandwidth utilization.

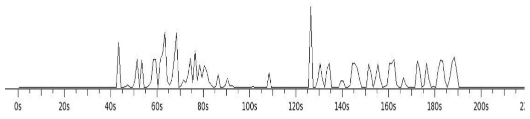


Figure 2. Two VNC sessions transferring identical content. The first was at 1 FPS (from 40 to 100 seconds) and the second was at 5 FPS

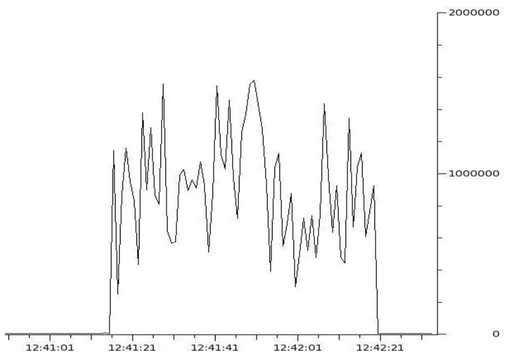


Figure 3. Using the same hardware setup as was used for the plot shown in Figure 2.

7. A MODERN INPUT DEVICE FOR DISPLAY WALLS

SAGE provides a new kind of platform and it benefits from new kinds of user interaction in order to exploit its capabilities. It supports traditional user interaction through the use of commands, a graphical application, or joysticks. None of these interactions are very useful for lectures when they need a direct interaction with the display and are not close to the computer. In order to have this kind of interaction, SAGE requires support for a new input device.

This section presents a new kind of input device that is more ergonomic than the traditional input devices supported by SAGE for working with the tiled display. A modern input device that fits this application is the Nintendo Wii™ Remote ("wiimote"). It provides a more natural interface to the display wall that does not require a surface on which to be placed. It also provides a similar interaction experience on an arbitrary display to touch-screen devices, which by experience have been very successful.

7.1 The Wiimote

The wiimote communicates with the Wii™ console or the PC though a Bluetooth wireless link. This link follows the Bluetooth Human Interface Device (HID) standard, which is the same interface used by traditional Bluetooth input devices, such as mice and keyboards. The wiimote sends state information with a frequency of 100 Hz and is composed of eleven input buttons, plus a shutdown button. It also has a special input though an infrared camera on top of it. This input has a resolution of 1024x768 and a field of view of about 45 degrees, and is able to "see" up to four IR points' positions. In order to use the IR input an IR source must be provided. In the case of Wii™ it uses the sensor bar, composed of two infrared light emitting diodes (LEDs). Based on the position information, the system creates a pointer device.

The wiimote has another special input, important for this work, that consists of a 3-axis linear accelerometer that detects movements along the three axes (in the three-dimensional Cartesian coordinate system) in the range of 3 g's (gravities) with a precision of 10%.

7.2 Implementation

The *wiiuse* [16] open source library was used for performing the low-level connection between the UI Client system and the wiimote. Both the IR sensor and the accelerometer sensor can be used with the SAGE display wall. The IR sensor would be the most intuitive pointer for the wall, but because a SAGE wall can potentially span several meters, the small field of view of the wiimote's sensor (45°) may be inadequate as a pointer for SAGE. As a result, we use the wiimote accelerometer and the wiimote buttons. The following inputs were defined and implemented: the accelerometer data of the X and Y axis are used in order to control the cursor on the screen. The "A" button is used to initiate the movement of application windows though the display. The "+", "-", are used to scale the size of the windows and the "home" button to rotate the applications. Also the cursor buttons can be used to navigate through the SAGE displays, providing a fast way to move across the wall, which can be difficult for normal joysticks when the wall is very large.

The functionality was implemented using two threads. The first one is responsible for getting the messages from the *FSClient* (a client of the *FSManager*) and process the message to manipulate the SAGE objects (e.g. the wall resolution, the SAGE applications positions, etc.). The second thread is responsible for getting the wiimote state, through *wiiuse*, from the bluetooth device, and sending messages to the *FSManager,* according to the wiimote input, as can be seen in Figure 4.

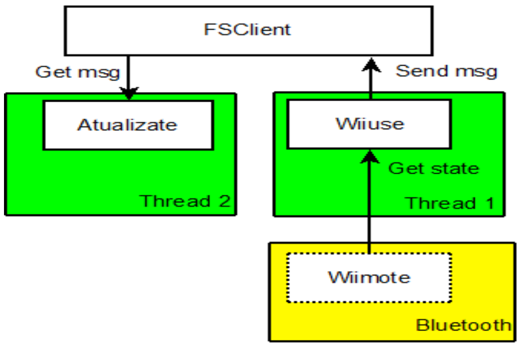


Figure 4. Two components, running in separate threads, interact with the *FSClient* and with the wiimote itself to form the interface to the display wall.

This application can also process input of messages through the keyboard, making it functionally similar to *FSConsole* (which is an application for the management of SAGE windows that provides a text-based interface to input commands to send messages to the *FSManager*). This is useful for the execution/shutting down of applications.

7.3 Evaluation of the Wiimote Interface

The use of the Wiimote in the SAGE environment provides an improvement in the degree of freedom when working with the SAGE display wall. Previously, the most common way to interact with the display wall and applications was through the use of the *SAGEUI* and the *FSConsole.* The SAGEUI is a graphical application that has the same functionalities as the *FSConsole*, but provides a more intuitive and user-friendly GUI. However, the *SAGEUI* requires that the user interact with a computer showing the current status of the wall (i.e. placement of windows). Referring again to the example in Section 3.1.1, the lecturer in this case would need to be in front of a computer showing the *SAGEUI*, which would give very little freedom in the case of needing a direct interaction with the wall if he/she is far from a computer.

The other way to interact to the wall is with the use of joysticks. These joysticks can only move and zoom applications. Also, normally these devices have wired interfaces to the computer, requiring the lecturer to be close to a computer. In addition, these devices are not made for user interaction. They normally need the use of both hands, and normally are not very ergonomic when used standing up. The use of a wiimote gives a new degree of freedom for the user interaction of SAGE. It is ergonomic, it only needs one hand for interaction, it is wireless, it allows the user to interact directly to the SAGE wall, and is capable of performing all the possible application windows transformations supported by SAGE.

8. CONCLUSIONS AND FUTURE WORK

Based on our evaluation, we see SAGE as a viable platform for remote learning and collaboration. With some optimizations to the remote desktop implementation, it could be used more effectively for this purpose. The *wiimote* interface gives users an improved interface to the display. Compared to other integrated collaboration solutions, it has the distinct advantage of supporting high-resolution graphics display.

Another feature we would like to add is a native web browser. Since many applications are written with a web browser as their graphical user interface, having a native SAGE web browser would allow users to use many more applications. This would be beneficial for collaborators and/or lecture sessions requiring special software that cannot be trivially ported to run on SAGE.

9. ACKNOWLEDGMENTS

The authors appreciate the support from Global CyberBridges, which is under the National Science Foundation office of cyber-infrastructure, CI-TEAM OCI-0636031. The NSF PIRE program, grant number 0730065, is also appreciated as it will help extend this research as well. Javier Delgado appreciates research grants HRD-0833093, CNS-0426125, CNS-0520811, CNS-0540592 and IIS-0308155.

10. REFERENCES

[1] Baratto, R.A., Kim, L.N., and Nieh, J. 2005. Thinc: a virtual display architecture for thin-client computing. In SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles, volume 39, pages 277–290, New York, NY, USA, December 2005. ACM Press.

[2] Distributed Multihead X Server (XDMX). http://xdmx.sourceforge.net

[3] EVO. http://evo.caltech.edu/evoGate

[4] Global Cyberbridges Project. http://www.cyberbridges.net/

[5] Hang, X., Villegas, D., Sadjadi, M., and Alvarez, H. 2007. Formative assessment of the effectiveness of collaboration in gcb. In Proceedings of the International Conference on Information Society (i-Society 2007), Merrillville, Indiana, USA, October.

[6] Hrastinski, S. Asynchronous and Synchronous E-Learning. EDUCAUSE Quarterly, vol. 31, no. 4 (October–December 2008). http://connect.educause.edu/Library/EDUCAUSE+Quarterly/AsynchronousandSynchronou/47683?time=1227443959.

[7] Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J. Johnson, A., and Leigh, J. 2006. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In proceedings of the ACM/IEEE conference on Supercomputing, page 108, New York, NY, US.

[8] Lai, A. M. and Nieh, J. 2006. On the performance of wide-area thin-client computing. ACM Trans. Comput. Syst., 24 (2):175–209.

[9] Leigh, J. et. al. The global lambda visualization facility: an international ultra-high-definition wide-area visualization collaboratory. 2006. Future Gener. Comput. Syst., 22(8): 964–97.

[10] Moodle. http://moodle.org

[11] Nieh, J., Yang, S.J., Novik, N. 2003. Measuring thin-client performance using slow-motion benchmarking, ACM Transactions on Computer Systems (TOCS), v.21 n.1, p.87-115, February 2003. DOI=10.1145/592637.592640

[12] Remote Framebuffer Protocol. http://www.realvnc.com/docs/rfbproto.pdf

[13] Richardson, T., Stafford-Fraser, Q., Wood, K.R., Wood, A.H. 1998. Virtual Network Computing, IEEE Internet Computing, vol. 2, no. 1, pp. 33-38, Jan/Feb., 1998.

[14] Skype. http://www.skype.com

[15] Tightvnc. http://www.tightvnc.com.

[16] WiiUse. http://www.wiiuse.net

[17] XNest. http://www.x.org

Introduction to Team-Based Learning

Patricia Lasserre
Computer Science, UBC Okanagan
3333 University Way
Kelowna, BC, V1V 1V7
(00) 1 250 807 9502
patricia.lasserre@ubc.ca

ABSTRACT

The generation Millennium is team-oriented, and very well versed on technology. Team-based learning (TBL), a teaching technique successfully used in Medicine since the late 80s, exploits that knowledge by enforcing individual accountability, emphasizing team work, and using any possible method (technological or not) to provide the immediate feedback this generation expects. After experiencing the technique hands-on, participants will understand how students achieve a deeper understanding by getting more actively engaged in their learning. TBL can be implemented gradually and flexibly, making it a very attractive technique as an alternative to traditional lecture format.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education*. K.3.1 [**Computers and Education**]: Computer uses in education – *Collaborative learning*.

General Terms

Experimentation.

Keywords

CS1/2, Active Learning.

1. INTRODUCTION

1.1 Session Summary

1.1.1 Target Audience

The session is targeting instructors who want to take group activities to the next level, or are curious about what team-based learning (TBL) is.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

1.1.2 Session objective:

In this one-hour workshop, the objective is to discover how TBL works in a short experiment. After a quick presentation introducing the technique, participants will experience hands-on team-based learning.

1.2 Background

Team-based learning (TBL) is a teaching technique which has been explored by Larry K. Michaelsen, Arletta B. Knight, and L. Dee Fink [4]. Since their book appeared in 2004, a few other disciplines then medicine [3,6,7] have started to introduce TBL in their curriculum including engineering [5], business [8], nursing [2], and computer science [10].

TBL emphasizes the application of concepts as a means to learn. The class time shifts from a lecture-type format to a practice-based format where students apply concepts learned prior to coming to class. Class time serves to test the understanding of concepts, close any gaps in understanding that students might have, and apply those newly acquired concepts on more and more complex problems. TBL uses the fact that peer discussion is recognized as valuable to students' success in understanding concepts, even when none of the peers initially understood the concepts [9]. Class time is used to have teams tackle problems, after all key concepts have previously been reviewed by individuals. Accountability and immediate feedback are two key ingredients of TBL. Accountability is ensured by evaluating first individuals and then teams. Competition between teams is also used to trigger interest and accountability. Immediate feedback is provided through various means ranging from traditional (cards, scratch tests, color pins, small boards) to technological (spreadsheets, in-class scanner, clickers [1]). Students are also encouraged to deepen their understanding through an appeal process to challenging the instructors' questions if they discover errors, or ambiguities.

After describing the technique, participants will experience hands-on its benefits. TBL makes students accountable for their own learning; it promotes team work as a means to deepen commitment and learning; it enforces immediate feedback which brings dynamism into the classroom, to provide a more engaging learning experience.

1.3 Biography

Dr. P. Lasserre is an associate professor in Computer Science at UBC Okanagan. Her research interests include student engagement and how to interest young children in sciences. Her preliminary use of Team-Based Learning on a first semester programming class (considered difficult by students) have convinced her of its benefits even on lower-level courses where students' accountability can be a challenge.

2. ACKNOWLEDGMENTS

This work has been financially supported by the Irving K. Barber Endowment Fund.

3. REFERENCES

[1] Ducan D. 2005. Clickers in the Classroom. Addison-Wesley

[2] Klose M. P., Fuchs S. B. 2007. Utilizing Team-based Learning in Nursing Education: How do you incorporate peer evaluation into assessment? TBL conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#e)

[3] Levine R. E., Sandor K., Sierpina V. S. et al. 2007. An Interdisciplinary Course in Spirituality and Clinical Care: comparing Team-based Learning Strategies with Lecture and Standardized Patient Scenarios. TBL Conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#k)

[4] Michaelsen L. K., Knight A. B., and Fink D. L. 2004. Team-Based Learning: A Transformative Use of Small Groups in College Teaching. Stylus Publishing. Sterling VA.

[5] Ostafichuck P., Hodgson A. 2007. Standing on Our Heads: How Teaching Engineering Design Looks Different from a Team-based Learning Perspective. TBL Conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#h)

[6] Parmelee D. 2007. Experiencing TBL as a Health Professions Educator. TBL Conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#4)

[7] Pasley J., Petty M., Jennings M. et al. 2007. Enhancing a Sim-Man Experience in Cardiovascular and Respiratory Physiology for First Year Medical Students through the Implementation of Team-based Learning. TBL Conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#7)

[8] Robert A. H. 2007. Team-Based Learning and the Business Strategy Game used simultaneously in a Business Strategy course; are the two compatible? TBL Conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#5)

[9] Smith M. K., Wood W. B., Adams W. K., et al. Why peer discussion improves student performance on in-class concepts questions. 2009. Science Magazine. 323 (5910). 122 – 124 (January 2009). DOI = 10.1126/science.1165919

[10] Whittington K. J. 2007 Understanding the TBL divide; examining similarities and differences between writing and programming. TBL conference. Vancouver (June 2007) (http://tbl.apsc.ubc.ca/conferences/2007/#1)

Masterclass on Using OPNET IT Guru

Karam Singh Khokra
School of Computing, Technology and Communications
Waiariki Institute of Technology
Mokoia Drive
Rotorua, NZ 3021
0064-7-3468651
Karam.khokra@waiariki.ac.nz

Richard Midgley
School of Computing, Technology and Communications
Waiariki Institute of Technology
Mokoia Drive
Rotorua, NZ 3021
0064-7-3468684
Richard.midgley@waiariki.ac.nz

ABSTRACT

Simulation Modeling is becoming an increasingly popular method for network performance analysis. OPNET IT Guru is a graphical user interface (GUI) based network simulation tool developed by OPNET Technologies which is simple to install and use. It provides a virtual environment for modeling, analyzing and predicting the performance of IT infrastructures, including applications, servers and networking technologies. OPNET Technologies provides OPNET IT Guru academic edition free to universities for both teaching and research purposes. It is designed to complement specific lab exercises that teach fundamental networking concepts. Thousands of commercial, government organizations and universities worldwide are using OPNET. This workshop will demonstrate how to use OPNET IT Guru for teaching basic networking concepts for undergraduate students.

Categories and subject descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Performance, Design, Study

Keywords

Data Communications, Networks, Computer Science Education Research, Teaching, Lab exercises, Software Tools, Simulation, Modeling, OPNET

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09 May 1-2,2009 Burnaby,BC, Canada
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

INTRODUCTION

This interactive practical workshop covers some of the features of OPNET IT Guru Academic edition. This software is used to teach data communication and networking concepts to computing undergraduates studying data communication papers in New Zealand computing degrees. A brief overview of why network simulation for data communications is required will be introduced. This will be followed by a brief review of how OPNET IT Guru was used in teaching certain aspects of computer networking in undergraduate papers in a typical British computer science degree. Best practices of the learning from both domains have been combined to ensure that future students benefit from using this tool to augment their existing methods of study.

A practical demonstration will be given of using a rapid tool within the software, to allow a simple network topology to be easily configured. It will also cover how appropriate network traffic can be added to this simple model and from this graphical simulation evidence gathered as output. The session will then demonstrate how simple iterations can be made to gather further graphical evidence. This allows a direct comparison between different scenarios to be undertaken.

The workshop will conclude with a question and answer session. Finally all attendees will be told how to obtain further resources to assist their investigation in to this network simulation tool.

INTENDED AUDIENCE

No previous experience with OPNET IT Guru is required. It requires a minimal knowledge of network devices and protocols. The session will be geared towards the computer science and information systems faculty.

INSTRUCTORS'S BACKGROUND

Karam Singh Khokra

Karam Singh Khokra is currently working as a Senior Lecturer in the School of Computing, Technology and Communication at Waiariki Institute of Technology, Rotorua, New Zealand. Karam holds a Masters Degree in Computer Engineering from

Kurukshetra University. He has been teaching networking subjects in the Bachelor of Computing System Degree at the Waiariki Institute since 2003. Karam has been in the computing industry since 1989 and has broad experience in managing and administering big networks. Karam has started using OPNET IT Guru for his teaching in 2008.

Richard Midgley

Richard Midgley is currently working as a Senior Lecturer in the School of Computing, Technology and Communication at Waiariki Institute of Technology, Rotorua, New Zealand. Richard holds Masters Degree in Computer Engineering from the University of Central Lancashire. Richard taught networking subjects on the Bachelor of Computing and eBusiness Systems Degree at Birmingham City University between 2001 and 2007. He has experience of using OPNET IT Guru Academic Edition since 2004.

Panel

Computational Thinking: Special Sauce or Snake Oil?

Yan Xu
Microsoft Research
Redmond, WA, USA
yanxu@microsoft.com

Joseph Peters
Computing Science
Simon Fraser University
Surrey, BC, Canada
peters@cs.sfu.ca

Arthur Kirkpatrick
Computing Science
Simon Fraser University
Burnaby, BC, Canada
ted@sfu.ca

Kevin O'Neil
Computing Science
Thompson Rivers University
Kamloops, BC, Canada
koneil@tru.ca

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer science education.

General Terms
Algorithms, Design, Theory.

Keywords
Computational Thinking.

1. SCOPE OF PANEL
Computer scientists are in wide agreement that computational thinking is destined to become a crucial topic in computer science departments. Advocates tout computational thinking as an essential tool for informed citizens of the information age, allowing those citizens to understand the digital infrastructure underlying their lives and also to solve longstanding problems using new computational approaches.

This broad range of application may result from an overly vague definition of the topic. Many activities have been grouped under the heading, "computational thinking". It is not clear if there is a consistent underlying core or if different authors use the same phrase to refer to different things. These potentially contradictory definitions in turn determine how widely or narrowly the ideas of computational thinking can be taught and applied.

Is computational thinking the "special sauce" that will spice up our major courses and our service courses, a topic that can and should be taught to everyone---or is it "snake oil", an evanescent idea with little applicability to the lives of students outside the technical departments directly using computation? This panel will present several points of view on computational thinking. Participants will present their definitions of the topic, which fields of study they believe it is most likely to enhance, and how broadly it can (and should) be taught across the university.

* Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
ACM 978-1-60558-415-7

Keynote Talk

Programming in an undergraduate CS curriculum

Bjarne Stroustrup
Texas A&M University
bs@cs.tamu.edu
www.research.att.com/~bs

Abstract

This note argues for a fairly classical undergraduate computer science (CS) curriculum where “software” (programming and related topics) takes a bigger role than is often the case. The discussion is based partly on experience with an undergraduate curriculum change at Texas A&M University and with developing a new freshman programming course. That freshman course is the central topic of this note. Based on industrial experience, it is argued that the primary aim of a university education in the area of “software” is to be a foundation for professional work. The primary design criterion for the freshman (first year) programming course is to make it a good start at that. Caveat: the opinions expressed about the needed improvements of and directions for software education is based on personal experience rather than hard data.

1 Introduction: Problems

My perspective is that of an industrial researcher and manager (24 years at Bell Labs; 7 of those as a department head) who has now spent 6 years in academia. I see a mismatch between what universities produce and what industry needs (not just what industry says it needs). When I say “industry” I obviously simplify, but I base my simplification on talks with dozens of representatives of (primarily, but not exclusively, US) industry a year over the last 30 years or so.

Industry wants computer science and computer engineering graduates to build systems consisting largely of software (at least initially in their careers). Many graduates have essentially no education or training in that outside their hobbyist activities. In particular, most see programming as a minimal effort to complete homework and rarely take a longer-term view involving use of their code by others and maintenance. Also, many students fail to connect what they learn in one class to what they learn in another. That way, you can (and often do) see students with high grades in algorithms, data structures, and software engineering hack solutions in an operating systems course with total disregard for data structures and algorithms, resulting in a poorly performing unmaintainable mess.

For many, “programming” has become a strange combination of unprincipled hacking and invoking other people’s libraries (with only the vaguest idea of what’s going on). The notions of "maintenance" and "code quality" are at best purely academic. Consequently, many in industry despair over the difficulty of finding graduates who understand “systems” and “can architect software.”

This is not all or even primarily the fault of the students. The academic curriculum is crowded with topics of undisputed importance and in the resulting competition for time, practical issues and the development of time-consuming practical skills (such as design for testing) lose out to more classical academic subjects (such as mathematical analysis of algorithms) or fashionable research topics (such as subsurface luminosity). However, to remain an applied discipline – as it has been from its inception – computer science must emphasize software development and CS programs must allocate time for student skills to mature. If we don’t, we are like a music department that does not require musicians to practice before a concert or an athletics department that “trains” its athletes primarily through lectures.

For the good of both industry and academia, we must do better.

2 What do we want?

A university education should lead to a well-rounded personality, provide a solid grounding in an academic field, and be a reasonable preparation for a job (or grad school entry). I consider a basic knowledge of history, art, math, and science (e.g., biology and physics) essential, so I’m not going to argue for more CS at the expense of a liberal education. I just wish the liberal arts curricula similarly went out of their way to avoid producing scientific ignoramuses.

The majority of CS graduates do not become professors, so I consider preparation for a career in industry crucial. I use the term “industry” in its broadest sense, including non-profit organizations and government. Industry does not want “scientists” in large numbers (I have heard repeated reports of recruiters denying interviews to “computer scientists” for

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and / or a fee.
WCCCE ’09 May 1-2, 2009, Vancouver, British Columbia, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

that flawed reason), engineers maybe (for some definition of "engineer"), but they definitely want "developers." Consequently, I consider producing software professionals (for some definition of "professional") the primary aim. Hardware is of course also essential and should be part of every curriculum, but in this context, hardware is "someone else's problem" so I won't consider it further.

I (emphatically) do not suggest that universities should simply produce what the majority of industry recruiters ask for: developers trained in the latest fashionable languages, tools, and methodologies. That would do harm to both academia and industry. Fashions come and go so rapidly that only a solid grasp of the fundamentals of CS and software development have lasting value. Training developers – rather than educating computer scientists (under whichever specific label you prefer) – would lead to a stream of employees being disposed of as fashions changed. That is neither moral nor cost effective. The alternative is a focus on fundamentals combined with the development of practical skills based on them.

The days where you could learn all that you needed for a career during four years in a university are long gone. A university education is just one stage (although an important stage) in a life-long education. Thus, there has to be an emphasis on providing and reinforcing dedication to and skills for self-education.

3 Computer science

Programming obviously isn't all there is to CS, however popular that misconception is among many people in the wider community. In fact, few tasks in software can be done well without a reasonable knowledge of algorithms, data structures, and machine architecture. For many tasks, we must add operating systems, networking, human-computer interfaces, graphics, and / or security to the requirements. And what about theory, compilers, math, etc.?

I find that CS professors often overreact to the inaccurate popular image of the software developer as a lonely guy with "no life" hacking code all night. To counter, they cry "Computer Science is *not* programming!" That's true of course, but that reaction can lead to a serious weakening of programming skills as some adopt the snobbish attitude "we don't teach programming; we teach computer science" and leave practical software development skills untaught.

Programming is the primary means of making ideas into reality using computers. Everything that runs on a computer is directly or indirectly expressed in a programming language. Most people don't have to program; their needs are well served by pointing, clicking, dragging, writing HTML (usually using tools), etc. However, to have a solid understanding of computer systems you need a practical and theoretical understanding of programming, programming languages, and other tools supporting the development of trustworthy code. Preferably, this understanding extends to several kinds of languages (e.g., declarative, scripting, general purpose, and close to the hardware) and several kinds of applications (e.g., embedded systems, text manipulation, small commercial application, scientific computation); language bigots do not make good professionals.

4 Software and the software curriculum

I see software as an artifact that is interesting in its own right (and not just for what it does) with a structure that can aid or hinder its growth and usability. I'm not alone in that view so parts the new Texas A&M University CS curriculum reflects it.

The study of "software" includes

- software engineering on a small group scale. The design, implementation, and maintenance of million-line systems are beyond our "software" curriculum and left for people who care to specialize in that and especially for industry to handle. A balanced 4-year undergraduate program can do little on this scale; though it must try to prepare students for scaling up.
- the use of programming languages (note the plural). However, most language implementation, language theory, and comparative languages studies are left for more specialized courses. For some of us that is a great loss, but something has to give.
- individual and group projects done at each level starting in the first programming course and repeated with increasing difficulty in every software course. These projects are central to teaching the beginnings of the "higher level" project management and software engineering skills. This is where tools such as test frameworks and source control systems find their natural homes.

The fundamental notion is that programming has a theoretical basis but is also a craft, like playing the violin. We need to teach both. We have seen too many examples of unnecessarily ghastly code – expensively and laboriously produced by bright, well-educated individuals who just never were given guidance in how to structure a program for further development and maintenance. Conversely, talented, mostly self-taught, programmers struggle for lack of a theoretical foundation for their work.

This leads to the question of what to teach first: the theoretical basis or the craft? Since there is no way a student can appreciate the problems of writing correct software or maintaining large code bases without having programming experience, we have to start by writing code. Only through trying to write code and debugging it do people get to appreciate the magnitude of difficulty in producing correct software. Furthermore, only by facing the problems of evolving an existing code base do people appreciate the value of clean design, supporting tools, and testing. Long lectures on software engineering to people with a weak software background are at best ineffective and at worst instill a dislike for programming as a low-level activity unworthy of serious attention ("a mere implementation detail") and / or of software engineering as "irrelevant and abstract." So we initially approach programming as a craft with an emphasis on how to get real-world code sufficiently good for people's lives to depend on it. That implies a constant attention to requirements of correctness and practical techniques of how to meet them. That way, the more abstract principles emerge naturally from concrete needs.

Our "software curriculum" is a sequence of courses:

- Introductory programming (using C++[2])
- Data structures and algorithms
- Programming languages (at least two that are not C++)
- Design studio (using at least two languages out of the three taught)

The "design studio" is primarily project based and aimed at pulling together what is taught in the other courses. Most topics are taught repeatedly in increasing levels of detail and rigor. These four courses are ideally fitted into the first two years of study and are designed to provide a student the basic knowledge and expertise to qualify for and benefit from an industrial internship.

This is of course just part of a broader curriculum offering machine architecture, discrete mathematics, human-machine interfaces, compilers, and more advanced and specialized courses. The software program is often completed through a project-based "capstone design course" taken in the final year.

5 The first programming language

The choice of a first language is always controversial. TAMU used Java for a few years and the experiences were mixed and the comments from industry interviewing and/or hiring students discouraging. This fitted a pattern I had seen in many places in industry: a desire for greater knowledge of "systems" and "machines" combined with an emphasis on performance that didn't match what the students had been led to expect. A further constraint was that the Electrical Engineering department and the Computer Engineering faculty insisted on "not a teaching language, but something close to the hardware, preferably C/C++."

It is often pointed out that the choice of programming language is less important that the choices of programming philosophy, design methodology, and teaching approach. Changing programming languages is at best part of a solution. However, a programming language carries with it a whole host of assumptions, attitudes, and an emphasis on select application areas. Part of the task of teaching programming is to make such cultural factors explicit and to use the initial language to its best effect in areas where it is suitable. Teachers must try to take the focus away from programming language technicalities and focus on more general issues, such as software development. However, a first language must be chosen and used well.

Given my background, our choice of C++ is unsurprising, but we try to avoid language bigotry by insisting that all CS graduates learn at least three languages. C++ has the strengths of being very widely used, having an ISO standard[1], being well supported on all platforms (including the embedded systems platforms), and supporting the major programming styles (paradigms). Its obvious weaknesses are complexity, archaic features, and the dangers of accidentally violating the type system. We decided to compensate for the weaknesses by an emphasis on type safety, encapsulation of "dangerous features," and use of libraries. Every line of code presented (except examples of what never to do, but including GUI) runs on all major platforms. Every time we have given the freshman course the students have used a mix of Windows, Macs, and (other) Linux systems.

A language, its compiler, and basic development platform is only the first and most basic tools for a software developer. The first programming course does not progress beyond this (unless you count downloading, installing, and using a software library that is not part of the standard). My view is that using a general-purpose programming language (rather than a simpler "teaching language") and common industrial programming environments (rather than specialized educational environments) give students as much initial exposure to tools as they can handle. Add more tools, and many will get distracted from the code itself and develop an unhealthy dependency on ill-understood, non-standard, and often proprietary facilities.

6 The introductory programming class

My basic idea for the design of the freshman programming class was to work backwards from what is required to start a first project aimed for use by others. That list of requirements defines the ideal set of topics to be covered. Naturally, we can't completely cover all that (even assuming suitable supervision for that hypothetical next project). You couldn't train a plumber in three months let alone an acceptable high-school violinist. To compare, learning the basics of a natural language takes upwards of three years. Yet, we succeed in assembling a toolset of concepts and techniques. Students have reported that they have put what they learned to good use on their first real projects.

The freshman programming class is based on a book that I developed concurrently with the course and refined based on experience with the course. The book is entitled *Programming – Principles and Practice using C++*[3]. Its preface, table of contents, lecture slides, and other supporting materials can be found here: `http://www.stroustrup.com/Programming.html`.

Our approach is "depth first" in the sense that it quickly moves through a series of basic techniques, concepts, and language supports before broadening out for a more complete understanding. The first 11 chapters/lectures (about 6 weeks) cover objects, types and values, computation, debugging, error handling, the development of a "significant program" (a desk calculator) and its completion through redesign, extension of functionality, and testing. Language-technical aspects include the design of functions and classes. Finally, interactive and file I/O are explained in some detail. The data types used are bool, char, int, double (a double-precision floating point number type), string (a variable length sequence of characters), and vector (an extensible container of elements). That's "the basics." At this point, the students can (in principle) do simple computations on streams of numbers and/or strings – they are by now dazed and need a break.

There is a constant emphasis on the structure of code (invariants, interface design, error handling, the need to reason about code to ensure correctness, etc.). This is hard on the students, but seems to bear fruit and stop them from obsessing over language details. Concepts and techniques are presented through concrete examples followed by the articulation of an underlying general principle. Typically, the students have a hard time grasping the importance of the principles, which are seen as “abstract,” so repeated application to concrete examples is essential. The style of the concrete examples reflects the principles and can – when imitated by the students – lead to later understanding.

At the end of this part of the course, the students should have no problems with simple exercises like this:

```
// produce the sum of the integers in "data.txt"
ifstream is("data.txt");
if (!is) error("data file missing");
int sum = 0;
int count = 0;
int x;
while(is>>x) {         // read into x
        sum+=x;
        ++count;
}
cout << "the sum of " << count
        << " elements is " << sum <<"\n";
```

They should also (often somewhat hesitantly) begin to define simple types (classes and enumerations) to simplify their code.

That “break” after “the basics” takes the form of 5 chapters / lectures (about 3 weeks) on graphics and GUI. This actually does refresh the students, increases their level of interest, and makes them work harder. It’s mostly graphics (as opposed to GUI) and the students do not perceive it as difficult. Class hierarchies and virtual functions are introduced. That is, the fundamentals of object-oriented programming are presented as a simple response to an obvious need. This is a technique we use repeatedly. After all, much of a first programming course is simply to supply a strange formal notation for what the students learned in primary school, or even before that (“if the light is green, cross the road; otherwise, wait” is not rocket science). By relying on such student knowledge where we can, we gain time needed to deal with concepts and techniques that are genuinely novel to the students. For example, we don’t spend much time of simple control structure, arithmetic, or (really) basic geometry. The student *do* know that. The time we gain can be spent on, say, input validation, error handling, and the value of user-defined types (classes and enumerations).

A typical simple exercise would be to define and use a simple graphical class:

```
// define a shape class "Arrow":
class Arrow : public Shape {
   Arrow(Point p1, Point p2)
        // construct an arrow from p1 to p2
        // with default head and tail
   {
        check(p1,p2);  // long enough?
        add(p1);       // store tail point
        add(p2);       // store head point
   }

   int check(int len)
   {
        if (length(p1,p2) < min_lgt)
           error("arrow length too small");
        return len;
   }

   void draw_lines() const;

   int length() const;
   Point point() const; // head point
   void set_point(Point x);

   // ...
private:
   const int min_lgt = 10; // minimal length
   Head hd, tl;            // head and tail
};

// make a few shapes:
Point center(0,0);
Arrow a0(center,Point(20,50));
Vector_ref<Shape> vs;  // container of Shapes

vs.push_back(a0);
vs.push_back(new Arrow(Point(20,50),Point(50,50)));
vs.push_back(new Rectangle(oo,Point(50,50)));
```

Larger examples includes the animation of consecutive approximations of an exponential function and graphing data sets from a file.

The third part of the course (about another 3 weeks) has two sections:

- The first is a detailed explanation of how the C++ standard-library vector is implemented and the second an introduction to containers and algorithms using the STL (including that vector). The STL is the ISO standard library facilities providing containers and supporting computations (algorithms) on sequences of data. The sequences of data can be either conventional input / output or containers of elements (such as the elements of a vector). Explaining vector involves the introduction of pointers, arrays, and C-style manipulation of memory. We did not consider it responsible to leave that out. This section has been expanded and refined over time based on requests from “consumers” of our students.
- The second section introduces the basics of generic programming. The STL provides a contrast to the low-level pointer and array manipulation from the first section with a much higher level approach to algorithms on data structures. We use vectors, lists, sets, and maps with algorithms such as find, sort, and accumulate. Most modern programming languages supports roughly equivalent facilities, even if they are typically either built-in (rather than provided in a library) or less general. I consider a working understanding of such basic data structures and algorithms in their colloquial form for a chosen language essential. There have been requests for expansion of this section also, but I don’t see how that could be done without cutting something else.

A typical simple exercise at this stage would involve the use of STL containers and algorithms:

```
// print unique words from iname to oname in order
istream is(iname);    // input from iname
ostream os(ofname);   // output to oname
if (!is || !os) error("couldn’t open file");

istream_iterator ii(is);
ostream_iterator eos;
ostream_iterator oo(os,"\n");

set<string> b(ii,eos);      // read words from input
                            // into a set
copy(b.begin(),b.end(),oo); // copy words to output
                            // ordered by the set
```

The combination of the STL for storage and algorithms with graphics is a good base for exercises and projects. However, the students have also tried conventional C-style techniques:

```
char* cat(const char* p, char c, const char* q)
   // concatenate p and q separated by c
{
   int lp = strlen(p);
   int lq = strlen(q);
   char* r = new char[lp+lq+2];
   strcpy(r,p);
   r[lp] = c;
   strcpy(r+lp+1,q);
   r[lp+lq+1] = 0;
   return r;
}

char* p = cat("someone",'@',"somewhere");
cout << p;         // "someone@somewhere"
delete[] p;
```

The variety of programming styles can be confusing, but it is minor compared to what is found in real-world code. To help the students cope, we frequently refer back to fundamental principles, to commonly useful styles of code (reflecting those principles), and try to offer guidance about preferences. Style matters.

Usually, there is time for just one more lecture: a presentation of ideals for software followed by a quick trip through the history of programming languages giving examples of how languages have increasingly supported those ideals. This lecture complements an introductory lecture on applications of software and the importance of software and its developers to society. This kind of motivational material is essential as the students are pretty clueless about the role of software in the world. We try to complement it with a sprinkling of “news flashes” in the individual lectures.

This material is very extensive for a first course and the pace quite high, but most students succeed. In addition to the final 3-person 3-week project (running in parallel with lectures) which all students do, we have noticed many students experimenting with private (not homework) projects such as a “catalog” of friends with contact information, comments, and photos. It is not uncommon to find elements in the final projects that were not taught, thus demonstrating student interest in and ability to go further. It would have been nice to give the students sufficient tools for a web interface for such projects, but that’s beyond us for now.

There is a fourth part to the book aimed partly as support for projects, partly to support people learning on their own, and partly for extended versions of the course: text manipulation (including standard-library regular expressions), numerics (mostly an N-dimensional matrix class to save people from the horrors of C-style multidimensional arrays), embedded systems programming (mostly low-level memory manipulation and bitfiddling), testing, and a survey of C (there is a *lot* of C code “out there”). Sometimes, we manage an extra lecture or two based on these chapters. That depends on how a class progressed, how many review sessions had to be included, etc.

7 Problems: Execution

What happens when these ideals, ideas, and plans meet a class of 240 freshmen of mixed abilities, aims, and backgrounds? My experience is based on a mixture of EE, CE, and CS students, 60% of whom have programmed before (mostly in high school "advanced placement" CS courses) and 40% have never seen a line of code. TAMU is a good public university, but not an elite institution with the ability to reject most applicants. This course has now been given nine times by a variety of professors to about 1500 students. We are reasonably convinced that the approach scales. A College of Engineering survey showed that the students on this course worked 25% longer hours than average for our engineering school freshman classes, yet reported 20% higher satisfaction. However, we can’t provide meaningful quantitative measures of success.

The most common complaint about the course has been that the order of topics is confusing and illogical. This primarily comes from students who have programmed before and have a firm idea of what should be taught and in which order. Typically, “bottom up” or “all C language features first” is seen as “natural” whereas the ordering based on programming needs and principles rather than language features is seen as “wrong and unnatural.” To contrast, students who have never programmed before do not have a problem with our early use of standard library facilities (such as iostreams, string, and vector) and do not find the early absence of pointers and arrays strange. Appendices presenting C++ and its standard library in conventional manual order aim to address these comments.

The second most common complaint is that the repeated statements of principles (to achieve correctness, maintainability, etc.) is “over our heads” and “irrelevant for programmers.” The latter comment proves the need for an emphasis on professionalism. We address this problem primarily through a close tie between concrete examples (code) and statement of principles. In particular, we (also) present examples of errors to teach the students to recognize both “silly errors” and violations of principles. The students do seem to make fewer “stupid errors.”

Textbooks: Before starting to design the freshman programming course, I surveyed a couple of dozen introductory C++/programming textbooks and saw some patterns I found disturbing. Most could fairly be criticized for teaching C++ more than programming and they tended to do the student a disservice by presenting a very complete and detailed bottom-up view. For example, some present all variants of C++ built-in data types and control structures before presenting meaningful examples of their use. Others present all features present in C before “the extensions” provided by C++ and avoid C++ standard library facilities. In the hands of an uninspired (or weakly prepared) teacher, this bores a good student to tears, presents programming as an unending sequence of obscure technical details, gives a view of C++ an unnecessarily complicated variant of C, and presents crucial higher-level concepts essential for industrial use (e.g., the

STL, ways of defining classes, and realistic uses of inheritance) late labeled "advanced" (and consequently typically avoided). In a one-semester course based on such books, students never reach useful programming techniques and are never faced with meaningful challenges. My response was to teach based on notes and turn those notes into a book, which is now the basis of the course as described above. Other software courses face similar challenges. Too many textbooks are either dumbed down, dissociated from real-world practice, or simple how-to-guides that don't expose students to principles.

Students: Mostly, the students are not a problem. Exceptions include students who have programmed before and insist on showing off and to teach other students "cleverer, more efficient, ways" of programming than what is taught, such as

```
char buf[128];
gets(buf);   // Nifty! efficient! simple!
```

rather than

```
string buf;
getline(cin,buf);  // professor's boring stuff
```

(Quick test: Why is that `gets(buf)` a disaster waiting to happen?[1]) If not carefully instructed, TAs can add to this problem by regurgitating what they have absorbed over the years rather than following the rules for the course.

Having not taught freshmen before, my greatest surprise was that a major component of the course became teaching the students the need to work and to give guidance on how to do that. Many had breezed through high school and expected to do well reading notes or listening to the lectures (not both!) and going to "the labs." The idea of having to concentrate during the lectures and then spend 10 hours a week outside class re-reading the notes and working through exercises was alien.

When we went from using notes to using the finished textbook and also made the complete set of lecture slides available on the web (primarily for the benefit of readers who are not TAMU students), we experienced a most unfortunate drop in attendance. Apparently, many students think that listening to a lecture as well as reading a chapter is a waste of time. I see the redundancy as necessary reinforcement, so I expect this drop in attendance to offset much of the gain from the better material. Since almost all students miss important points when doing a single pass over new material, they will pay for their "saved effort" with longer debug sessions.

Projects: The final 3-person 3-week project (alongside lectures) is essential. That's where everything comes together and where the students get their first taste of project management. It's hard to come up with a sufficiently long non-repeating series of such projects. Examples so far include scrabble, sudoku, and whack-a-mole. Games make good projects because they are perceived as interesting and are open-ended (e.g., improve the GUI interface, add "cleverness" to the computer "player," etc.).

Exercises: Most exercises are of the "write a program" variety. So far, we have always been short of good exercises. This problem is decreasing over time, but finding exercises that are challenging (but not too challenging) for the students and also present problems that the students can relate to is hard. In particular, we need more exercises of the sort where the teacher provides a larger program and the student fixes bugs and/or extends functionality. Many exercises involve improving solutions from previous chapters, thus hopefully reinforcing the lesson that code is an artifact that needs a structure to ease modification (a.k.a. "maintenance").

Grading and testing: Obviously, grading of homework of the "write a program" variety is time consuming and tedious. It is also unavoidable and essential to provide students feedback on their coding style. In this, CS has much to learn from the humanities. Inadequate feedback caused by lack of time and attention can undermine the key message of "correctness and systematic testing of code." The fact that we – due to lack of funds and personnel for alternatives – regularly have to use multiple-choice testing is a major problem. Many students will study for the test, and multiple-choice tests are ill suited for testing higher-level skills. Imagine basing a large part of a pianist's or a soccer player's ranking on a multiple-choice test! I consider this analogy between programmers, artists, and athletes fair. My ideal world has much feedback from teachers and no multiple-choice tests. But in the real world, the student/teacher ratio often limits what can be done.

The exact details of grading have varied over time, but this should give an idea.

- 30% Homework and drills; a drill is a simple multi-step programming exercise to ensure that a student has mastered the basic practical aspects of a lecture.
- 45% Three (closed book) multiple-choice exams (Yuck!)
- 20% Final (group) project
- 5% Attendance and class participation (verified by pop quizzes).

We plan to increase the weight given to pop quizzes (to increase attendance) at the expense of the – quite likely counterproductive – exams. This system of assessment is probably the weakest aspect of the course, but we are heavily constrained by local culture.

Teaching assistants: Typically, TAs are grad students that arrive back on campus on the same day as the undergraduates. This makes it really hard to be ready for the "Hello, World!" program on day 1. Departmental machines have to be ready and the TAs gathered together to be minimally trained. It is hard not to stumble and immediately get the homework out of sync with the lectures. Though fundamentally not the TAs' fault, this has been a major problem.

Another problem is that not all TAs attend a lecture and also carefully read the chapter before seeing the first student. Some try to "wing it" by relying on earlier knowledge of C/Java/C++/programming. However, what we teach is not "your father's C++" and our approach to software development is less tolerant of "messy code" than they have been used to, so sometimes they end up sending the message that less rigorous approaches to programming are acceptable.

[1]The input may overflow the fixed-size buffer; this used to be the single most common security hole in C code.

Professors: For scaling a new approach, we are the weak link. Any mistake by the professor gets amplified by the TAs and the students. In particular, falling back to a slower pace, explaining more of the basics early on, and giving the students a break on exercises are obvious dangers. Students live up to expectations, but are also quite willing to be convinced that a course is "a waste of time" and "too difficult." By slowing down, a professor can significantly delay the point in time where students gain satisfaction from completing a program that actually does something interesting. By slowing down, a professor can seriously delay the point where a motivated student feel the satisfaction of mastery of new knowledge or skill. By slowing down, a professor can trap motivated students into a pattern of inattention, too little work, and alternative activities (not leading to higher-level programming skills). Whatever we do, we should take care not to demotivate the students most likely to become the best software developers and computer scientists. Rather than slowing down, we should – and do – add TA support and review sessions (really catch-up session) for the tail end of the students.

Software: We allow students to use any computer with a reasonably up-to-date C++ compiler. That has worked pretty well. The major problem is that C++ does not have an ISO standard GUI. Instead I had to choose from among the couple of dozen available C++ GUI libraries and toolkits. I chose to use FLTK[4] because it was portable, reasonably simple, not particularly controversial in the community (having several GUIs creates confusion and competition that is not always polite), and relatively easy to install. That "relatively easy" can be hard for someone who has never downloaded and installed a library before. Obviously, we install a version for students who use university computers, but more adventurous students sometimes have to be helped by the TAs. Setting up the correct compiler / IDE settings to use the GUI can also be quite frustrating.

Other software is just standard libraries or header-only libraries which cause no significant problems. To simplify the earliest classes, we use a header file that includes the necessary standard headers and an error() function; that way, we don't get embroiled in discussions about header files, system interfaces, and namespace management until those topics natually fit into the sequence of topics (e.g., Chapters 6, 8, and 12)

Non-problems: It is widely believed that any teaching of C++ is associated with endless searches for "buffer overruns" and misused pointers. We relegated such problems to the status of a minor nuisance through the systematic use of the standard library (in particular, range-checked vectors and strings) and the restriction of the use of pointers and arrays to the innards of classes.

8 Etc.

The ideal CS curriculum consists of so many topics that are fundamentally important, interesting, and in high demand by "consumers" (that is, industry and grad schools) that no student can complete it in four years – especially not if they take a reasonable load of humanities and science courses that they need to grow as individuals and to interact with non-CS people in the workplace. There is a serious information overload. My suggestion is to make a Master's degree the first degree considered suitable for a software development job. This view is hardly revolutionary: that used to be the view in Bell Labs and is the traditional view in many European countries. However, in a US university, this would be a culture change and is beyond the scope of a single department or a single university. Independently of that, we need to increase the level of professionalism, rather than pandering demands for

- "better trained" (but not educated) students from industry,
- "easier and more exciting courses" from students,
- "things done the way we are used to" from teachers unwilling to face a serious challenge,
- "more concrete, simpler-to-grade material" from TAs selected for their scholarly abilities rather than their software development skills or teaching experience,
- "more advanced / scientific / theoretical courses" from professors wanting Ph.D. students.

I am no fan of monocultures. I consider it essential that universities offer a variety of undergraduate courses with widely differing emphases in the area of software (and within other areas of computer science). Ideally, different software programs will emphasize different levels of "the software stack" and target different "consumers" of graduates (e.g. with emphasis on different programming languages and different application areas). Within the TAMU undergraduate CS curriculum we cater to the need for diversity of subjects by giving the students a choice of "tracks" with "software" being just one alternative.

Our emphasis on code, correctness, and classical CS topics may seem to go against the obvious and necessary aim of attracting more students to the computing profession. I don't think it does. It has been suggested that such an emphasis especially discourages women and minorities from studying computer science. On the contrary, offer anybody something they perceive as valuable and they'll come. Medicine did not go from having essentially no women to almost 50% women by offering cuddly soft choices. It did so by offering solid knowledge, professional status, good careers, and an obvious way for an individual to benefit society. Our freshman programming course contains a repeated emphasis on the benefits to society provided by software developers and explanations of the wide variety of applications areas. For freshmen, appeals to idealism are fortunately still more effective than quoting salary statistics (though the stellar statistics for CS, CE, and EE majors also help). Other courses and lecture series follow up on this. Whether that can achieve anything against "Hollywood"'s persistently negative image of programmers and engineers is doubtful, but it is a start.

9 Acknowledgements

Thanks to Walter Daugherity, Jaakko Järvi, Teresa Leyk, Ronnie Ward, and Jennifer Welch for constructive comments on drafts of this note and / or discussion about the first programming course. They all teach parts of the TAMU CS software curriculum. Also thanks to the WCCCE reviewers for constructive comments and hard questions.

10 References

[1] *Standard for the C++ Programming Language*. ISO / IEC 14882-2003.

[2] B. Stroustrup: *The C++ Programming language*. Addison-Wesley 2000. ISBN 0-201-70073-5.

[3] B. Stroustrup: *Programming – Principles and Practice using C++*. Addison-Wesley 2008. ISBN 978-0321543721.

[4] FLTK: *Fast Light Tool Kit*. `http://www.fltk.org/`.

WICS @ SFU: Assessing the Impact and Outcomes of a Women in Computing Science Student Group at the College Level

Kathleen Tsoukalas
Department of Computing Science
Simon Fraser University
8888 University Drive
Burnaby, BC, Canada
kjtsouka@cs.sfu.ca

Winona Tin Wing Wu
Department of Computing Science
Simon Fraser University
8888 University Drive
Burnaby, BC, Canada
winonaw@cs.sfu.ca

ABSTRACT

In 2002, Women in Computing Science at Simon Fraser University (WICS) was established first as an email list and later as a full-fledged student group for individuals with a wide diversity of backgrounds with the following goals: first, to promote women in Computing Science; second, to support women throughout their study of Computing Science; third, to build a strong network of friendly faces for women in Computing Science, and four, to challenge the biases and myths faced by women in Computing Science [1].

In this paper, we assess the impact of WICS on its members and its value in SFU's CS Department through the results of a survey and statistics on WICS activity. We also propose future directions for the group as well as recommendations for those wishing to start similar groups.

Categories and Subject Descriptors

K.3 [**Computers and Education**]: Computer Science Education; K.4.2 [**Computers and Society**]: General

Keywords

Attrition, Gender, Retention, Undergraduate Education

1. INTRODUCTION

Women have long been under-represented in the field of Computing Science, both in academic [9] and industrial settings [10]. Many theories exist as to why this is the case; some attribute these lower levels of involvement to gender differences in experience, motivation [4] [6], mathematical ability [7], or academic preparation [5], while others point to departmental factors within academic institutions as causing low retention rates for female students [8].

To address these issues, much focus has been placed on improving the situation in academic institutions. For example, Coohoon[2] provides recommendations for recruiting women, such as "Promote interaction among classmates, and develop learning communities and other forms of peer support" and "Mentor undergraduates", as well as recommendations for retaining women, such as "Provide female role models". It can be daunting to address each of the many recommendations in the literature individually, but one way to maximize the benefits to students is to create a group to support female students.

Given that groups for female students are a good way to improve recruitment and retention in Computer Science programs, Simon Fraser University (SFU) has its own version, called "Women in Computing Science" (WICS). WICS began in 2002 and is still a vibrant and highly active group, but surprisingly the value and impact this group has on SFU's Department of Computing Science and its community of students has never been measured or documented. Therefore, here we attempt to assess WICS impact and outcomes and propose future directions for the group, as well as recommendations for others attempting to start and build similar groups in their own institutions.

2. BACKGROUND: WICS

In this section, we discuss the diverse nature of WICS' membership and the strategy it has adopted for sustaining itself while also meeting the needs of the female students in CS at SFU.

2.1 WICS Membership

WICS members have a variety of backgrounds and occupy diverse roles in SFU's Computing Science Department, including undergraduate students, graduate students, faculty, alumni, and members external to SFU, such as high school students, parents, and industry workers. WICS membership is open to male and female members, but the executive positions are reserved for female students only.

It should be noted that WICS events do overlap with those of the Computing Science Student Society (CSSS), which is part of the Simon Fraser Student Society (SFSS). However, WICS not only caters to the unique interests of women in the department, but provides more events concentrated on professional development. Also, the CS Department does

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM.
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
Copyright 2009 ACM 978-1-60558-415-7/09/0003...$5.00

have technical talks and industry speakers, but these are more centered around the graduate school and many undergraduates don't feel comfortable attending. WICS events often take place in smaller groups with fewer numbers (rarely more than 30 attendees), so WICS members are more likely to participate.

2.2 WICS Strategy: Diverse Events for Diverse People

Given the diverse nature of WICS' membership, over the past year (2007-present) WICS adopted a policy of diversifying the types of events held each semester. These events can be broken down into the following categories: Technical Events, Social Events, Discussions, Industry-Related Events, and Conferences.

2.2.1 Technical Events

Technical events provide students with the opportunity to expand their CS-related skills such as programming or problem-solving in a low-pressure, supportive setting. WICS has held various technical events, such as a "Ruby on Rails Workshop", a "Java to C++ Workshop", and a "Technical Inteview Preparation Workshop". At least one technical event per semester is held and the information and skills provided are generally complementary to those provided in the classroom setting.

2.2.2 Social Events

Being a small minority in a large department can make it difficult for members to connect and meet new connections within the department, whether they be with other students, faculty, or staff. WICS' social events provide members with an opportunity to network and enjoy a break from studying. WICS' generally starts each semester with a 'Kick-Off Lunch' and ends the semester with an outing off-campus.

2.2.3 General Meetings and Discussions

WICS has been a long-time supporter of SCWIST[1] and holds regular 'Brown Bag Lunch' events, panel discussions and presentations on various topics. WICS holds bi-monthly meetings, with each meeting ending with a puzzle or game. Prizes consisting of donated books and other inexpensive items are always provided as incentive!

2.2.4 Industry-Related Events

Industry-related events provide students with the opportunity to gain insight into working in industry. WICS participates in external events such as SCWIST's XX Science World [2] and organizes industry tours such as an SAP Business Objects Tour & Q/A Session.

2.2.5 Conferences

Members of WICS have been attending the Grace Hopper Conference since 2002, and in 2008 a delegation of 8 students was organized. There are similar plans for a delegation to the 2009 conference. Although some staff members of WICS do send out information about this conference each year, WICS provided an organizer and concentrated meetings and emails on recruiting delegates, as well as organizing flights, accomodations, and ground transportation. Without WICS heavy involvement with this recruiting, it is unlikely that so many students would have attended in 2008.

[1]Society for Canadian Women in Science and Technology: http://www.scwist.ca/index.php

[2]http://www.scwist.ca/index.php/main/entry/xx-evening-march-11-2009/

2.3 WICS Recruitment and Growth

Since WICS' inception in 2002 there has been a recurrent danger of the group's dissolution due to graduation of core members. However, by employing the strategy of diversifying events, as well as a personalized recruitment strategy, we have increased the average number of attendees at events and meetings in 2007-2008 to approximately 20.

Employing new media has helped WICS reach current and new members, and WICS is now using Facebook [3], LinkedIn[4], as well as incorporating a blog on its website[5] and the original email list. From 2007-2008, WICS' Facebook group grew from 10 to 82 members, and this has been the most effective method for reaching members. LinkedIn, a network centered more on professionals than students, has been more successful in pulling in WICS' alumni than current members. Seven members belong to our group there. From our experience, if events and meetings are advertised solely on the email list without the support of the Facebook group, attendance is much lower.

3. EVALUATING WICS

An anonymous survey was conducted of 29 past and present WICS members. Participants were recruited both through our email list and individually, and conducted the online survey using SFU's internal survey tool, 'SFU WebSurvey'.

The survey is presented in Appendix A, and has four parts. First, we asked about our participants' demographics, such as gender and status as student, faculty, staff, or external. Second, we asked for participants' roles as well as the level and duration of their involvement in WICS. Third, we ask for feedback regarding how participants heard about WICS, WICS' events, and areas of impact WICS has had on them. The final section asks respondents to describe what their experience in CS would be like if WICS did not exist. We present the results in the following sections.

3.1 Demographics

In 2002 WICS was primarily an undergraduate organization, but we have since seen a growth in the number of graduate student members, especially from 2007-2008. This increased involvement provides excellent mentoring opportunities for undergraduate students. This increased involvement is reflected in our survey demographics, with 8 graduate students participating out of 29 total participants. In addition, 16 participants were undergraduate students, with one of these being a continuing studies student (this is an umbrella term including Certificate, Post Baccalaureate, and Second De-

[3]http://www.facebook.com

[4]http://www.linkedin.com

[5]http://wics.cs.sfu.ca

gree students[6]). There were 2 faculty members and 3 staff members. Finally, 27 survey participants identified themselves as female, while 2 identified themselves as male.

3.2 Member Roles and Involvement in WICS

Section 2 of the survey covers participants' roles as well as levels and duration of involvement in WICS. The numbers of respondents with each type of role (Executive Member - Past, Executive Member - Present, Regular Member - Past, Regular Member - Present, Just on Mailing List, and Not a Member) are shown in Table 1, which shows a distribution of participants who are members past and present, and who range from a variety of different demographics.

Interestingly, we can see from the survey results that the majority of participants began their involvement in WICS in 2007 or 2008 (16 participants), with the next largest year being 2003 (8 participants). 2004-2006 had low participation, and we see the lowest number of survey participants from those years (5 total).

3.3 WICS Events, Activities, and Areas of Impact

In this section we asked participants to rate which events they found valuable (more than one may be selected) and to also rate areas of WICS by level of importance to them.

We found that technical events (23 responses), social events (21 responses), and industrial events (19 responses) were rated as the most valuable to a majority of participants. Interestingly, conferences (15 responses) rated slightly lower than these three events, but slightly higher than discussions and meetings (14 responses). 4 responses indicated 'other'. These provided alternative activities such as networking, outreach for younger students, a games night for WICS members, and mentoring programs for high school students.

In terms of the types of events and activities held by WICS, participants were asked to rate individual aspects of WICS on a scale of zero to five, with zero being 'not important at all' and five being extremely important'. The results show that technical and social events had the highest average rating at 3.4 each, while meetings were rated less important at 3.1. Industry events were rated at 3.3. The average rating of WICS' importance was 3.9.

Finally, we asked participants to select all areas in which WICS had impacted them. The largest area of impact (24/57 responses) was the Social aspect of WICS, with the next largest area being Leadership (18/57). The Professional area came next at 12/57, with Academics and Other at 6/57 and 3 'No Impact' responses. For these last two, participants were permitted a free response. A participant who responded 'Other' said, "A senior WICS member told me in 2nd year that the courses would become much more interesting and fun in 3rd and 4th. Basically she told me to hang in there when I thought CS wasn't for me (I wanted to quit because of a bad grade in stats or architecture, I think).". A 'No Impact' response indicated that WICS events were at inconvenient times and as such this person could not attend at all.

[6]These programs are offered by SFU's Computing Science department and generally require a Bachelor's degree in either CS or another field. They are intended to help students upgrade their skills in preparation for either industry or graduate studies. More information can be found here: http://www.cs.sfu.ca/undergrad/Advising/programs.html

3.4 Life in CS Without WICS

In this section we ask participants to speculate on how they would find being a female student in their Computing Science program if WICS did not exist. Specifically, the question we asked was: "Imagine WICS did not exist. Would your experience in SFU's Computing Science program be different? If so, how?"

The responses show the impact of WICS on students, opening doors and presenting new learning and professional opportunities. For example, one participant wrote, "...I learned about amazing events through WICS specifically the Grace Hopper Celebration for Women in Computing. I have gone twice and hope to go again even after graduation". Another wrote, "WICS brings us a lot of information. Without WICS, I would not attend CRA-W meeting."

Technical and industrial aspects may be important to participants, but one can not ignore the social aspects of WICS and their importance to the group and its members. For example, one participant writes, "Without WICS, I would have little to no social connection to the CS program and its students. It is likely I would have pursued another major option." Another writes: "...It's hard to say what choices I would have made without WICS; I might have completed my degree in CS and gained leadership experience only in the CSSS, with lots of male friends. But with WICS I've made a huge number of female friends who continue to inspire me and give me advice till this day."

Finally, it is interesting to note WICS' impact from the perspective of faculty and staff. One writes, "Before WICS, women were less active in the CSSS (and the CSSS common room). WICS has also helped more women become known to faculty and staff.". Another says, "Yes. WICS is integral to the CS community. I would most likely see a decline in female CS students and graduates if not for WICS."

4. CONCLUSIONS

We can see from the results that members highly value the industry-related, social and technical events WICS provides. These events provide professional and academic development in a relaxed social setting, a successful strategy also employed in [3]. Here we present recommendations specific to WICS as well as some future directions for the group. We also summarize our recommendations for those looking to duplicate WICS' successes in their own groups.

4.1 Recommendations for WICS

Recommendations stemming from our survey results are as follows:

1. Increase the number of technical events per semester: these were rated the most valuable WICS activity, and more members may be attracted if more of this type of event were provided each semester.

Table 1: WICS Member Roles by Demographic

Member Status	Executive Members		Regular Members		Just on Maillist	Not a Member
	Past	Present	Past	Present		
Undergraduates	3	5	1	1	4	1
Master's Students	1	1	1	0	0	0
PhD Students	2	0	1	0	1	0
Staff	0	0	0	3	0	0
Faculty	0	0	0	0	1	0
External	0	0	0	0	0	0

2. Continue organizing delegations for and promoting the Grace Hopper Celebration of Women in Computing Conference: these events rated at least as highly as WICS Discussions, but are rich in potential. Students may find internship opportunities, potential graduate supervisors, learn about cutting edge research and issues such as diversity and retention in industry and academia.

3. Put less energy/resources into discussion events: although rated the same as conferences overall, these events did not garner any special mention in the long responses and have traditionally been less well-attended than other types of events.

4. Continue the strategy of 'Diverse Events for Diverse People': diversifying our events has continued to work over the past two years. We should continue to have at least one type of each event, but concentrate WICS' resources on those rated more highly by survey participants

5. Focus recruitment efforts on undergraduate students

4.2 Future Directions for WICS

Future directions for WICS include more outreach and recruiting programs for students at earlier stages in their education. WICS is a strong supporter of SFU's ChicTech[7] program and generally supplies all or most of the volunteers that organize it each year. WICS will continue to support ChicTech but will also explore new opportunities centered on recruitment of these students. This position is supported by our survey results, in which some participants suggested that they would like to see more such events, and this is an important means of both recruiting more female CS students as well as ensuring WICS' group renewal and survival. Other participants also expressed their support of mentoring opportunities, which are a key part of events like ChicTech.

One new idea we would like to pursue is a 'Shadow Day' where highschool students would be matched with a WICS member. The students would have one-on-one time with the WICS member, attending classes and regular daily CS activities with them.

4.3 Recommendations for Other Groups

WICS' strategy of personalized recruitment (inviting potential members by word-of-mouth, information from friends, and recruitment at Open Houses, Clubs Days, and other face-to-face events) as well as new media such as Facebook, LinkedIn, and blogs has been very effective and we recommend new groups to use such a multi-faceted approach to recruitment. Most survey participants (21/29) heard about WICS from either a friend or from faculty and staff, but our online presence as well as consistent appearances at school events and classes form a good support for the word-of-mouth methods.

A diversified approach to events allows members to pick and choose what they want to participate in, providing something for everyone and including as many potential members as possible. This strategy has maximized student involvement in WICS and has kept the group active. We recommend that other groups employ a similar strategy and maintain consistency in the types and number of events held each semester, so members come to expect them and look forward to attending.

5. ACKNOWLEDGMENTS

The authors would like to thank Dr. Ted Kirkpatrick of Simon Fraser University's Computing Science Department for sponsoring the research for this paper and for his helpful suggestions regarding our survey.

6. REFERENCES

[1] WICS Constitution.

[2] J. M. Cohoon. Recruiting and retaining women in undergraduate computing majors. *SIGCSE Bull.*, 34(2):48–52, 2002.

[3] C. Frieze and L. Blum. Building an effective computer science student organization: the carnegie mellon women@scs action plan. *SIGCSE Bull.*, 34(2):74–78, 2002.

[4] Katz, S., Allbritton, D., Aronis, J., Wilson, C., Soffa, M. L. Gender, achievement, and persistence in an undergraduate computer science program. *SIGMIS Database*, 37(4):42–57, 2006.

[5] Kersteen, Z. A., Linn, M. C., Clancy, M., & Hardyck. Previous experience and the learning of computer programming: The computer helps those who help themselves. *Journal of Educational Computing Research*, 4(3):321–333.

[6] S. Kesici, I. Sahin, and A. O. Akturk. Analysis of cognitive learning strategies and computer attitudes, according to college students' gender and locus of control. *Comput. Hum. Behav.*, 25(2):529–534, 2009.

[7] J. Konvalina, S. A. Wileman, and L. J. Stephens. Math proficiency: a key to success for computer science students. *Commun. ACM*, 26(5):377–382, 1983.

[7]http://cgi.sfu.ca/ wics//chictech/

[8] C. J. McGrath. Departmental differences can point the way to improving female retention in computer science. In *SIGCSE '99: The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 198–202, New York, NY, USA, 1999. ACM.

[9] G. Scragg and J. Smith. A study of barriers to women in undergraduate computer science. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 82–86, New York, NY, USA, 1998. ACM.

[10] K. Todd, L. Mardis, and P. Wyatt. We've come a long way, baby!: but where women and technology are concerned, have we really? In *SIGUCCS '05: Proceedings of the 33rd annual ACM SIGUCCS conference on User services*, pages 380–387, New York, NY, USA, 2005. ACM.

APPENDIX

A. SURVEY INSTRUMENT

A.1 Consent Form

Title: WICS & You: Assessing the Impact and Outcomes of a Women in Computing Science Student Group at the College Level

Investigator Name: Kate Tsoukalas Investigator Department: Computing Science

This short survey will take about 5-10 minutes of your time and provide us with invaluable data on how WICS has benefited you, our members, or what we can do better. Your input will help us continue to improve WICS and provide better services to the CS community.

By filling out this questionnaire, you signify that you have read this document, which describes the procedures, whether there are possible risks, and the benefits of this research study, that you have received an adequate opportunity to consider the information in the documents describing the study, and that you voluntarily agree to participate in the study.

Procedures: You will be asked to complete an anonymous online survey regarding your experience in the Department of Computing Science and with the Women in Computing Science student group. The survey uses SFU's WebSurvey tool, which means that your responses will be given over a secure website.

Risks to you: We will record your responses to the survey, which may include your email address should you choose to disclose it. This information will be kept for two years after the study has been completed on a CD in a locked cabinet in the Computing Science office. For participants who choose to disclose their email address, there is some risk that this data may become public (in the event of theft of the CD).

You may obtain copies of the results of this study, upon its completion by contacting Kate Tsoukalas at kjtsouka@gmail.com.

The University and those conducting this research study subscribe to the ethical conduct of research and to the protection at all times of the interests, comfort, and safety of participants. This research is being conducted under permission of the Simon Fraser Research Ethics Board. The chief concern of the Board is for the health, safety and psychological well-being of research participants.

Should you wish to obtain information about your rights as a participant in research, or about the responsibilities of researchers, or if you have any questions, concerns or complaints about the manner in which you were treated in this study, please contact the Director, Office of Research Ethics by email at hal_weinberg@sfu.ca or phone at 778−782−6593.

If you do not wish to continue with the survey, please click here to exit (http://wics.cs.sfu.ca/).

A.2 Questions

1. Would you like to specify your gender?
   - Male
   - Female
   - Prefer not to specify
2. I am/was a:
   - Undergraduate Student
   - Continuing Studies Student (Second Degree, Post-Baccalaureate)
   - Graduate Student - Master's Graduate Student - PhD
   - Staff
   - Faculty
   - External to SFU
3. My role in WICS is best described as:
   - Executive Member - Past
   - Executive Member - Present
   - Regular Member - Past
   - Regular Member - Present
   - Just on Mailing List
   - Not a Member
4. When did you first become involved with WICS (i.e. by joining the maillist or coming to meetings/events)?
   - 2003
   - 2004
   - 2005
   - 2006
   - 2007
   - 2008
   - 2009
5. How long have you been involved with WICS? If you are not currently involved in WICS, for how long were you involved?
   - $<$ 1 year
   - 1 - 2 years
   - 2 - 3 years
   - 4 years or more
6. I attend(ed) meetings and events:
   - Always
   - Often

- Sometimes
- Never

7. How did you hear about us? Please select all that apply.
   - Friend
   - Clubs Day Booth
   - Classroom Visits
   - Flyers/Pamphlets
   - Staff/Faculty
   - Other (please list in the next question)
8. If you selected 'Other', please describe how here:
9. WICS has many different events each semester. Please select which event(s) you find valuable (select all that apply):
   - Technical Events (Interview Preparation, Language Workshops)
   - Social Events (Lunches, Dinners, Outings)
   - WICS Discussions (Panels, Brown Bag Lunches)
   - Industry-related (Industry Tours, Panels, Networking)
   - Conferences (Grace Hopper Conference)
   - Other
10. If you would prefer to see other types of events or have comments on previous events, please let us know here:
11. Please rate how important WICS is to you in the following areas. (0 = not important at all, 5 = extremely important)
    - WICS Meetings
    - WICS Social Events
    - WICS Technical Workshops
    - WICS Industry Events
    - WICS Overall
12. In which area(s) has WICS impacted you? You may select more than one.
    - Leadership (Gain experience giving presentations, managing projects, organizing events)
    - Academics (Improved grades, study buddies, learned new skills at technical workshops)
    - Professional (Industry networking, internship/coop opportunities, full-time employment)
    - Social (New friends, contacts, connections)
    - Other (please detail in the next question)
    - No impact (please detail in the next question)
13. If you selected 'Other' or 'No Impact', please explain here.
14. Imagine WICS did not exist. Would your experience in SFU's Computing Science program be different? If so, how?
15. We may be interested in interviewing members about their experiences. If you would like to participate in these interviews, please enter your email address here:

Graduate Follow up as a Vehicle for CSIT Curriculum Assessment and Improvement

Amos O. Olagunju, Matthew J. Cameron, Richard Mowe
Computer Networking and Applications Program
St. Cloud State University
aoolagunju, remowe@stcloudstate.edu

ABSTRACT

Internal and external program reviews are useful for targeting areas of Computer Science and Information Technology (CSIT) curriculums for revision and improvement. The successful working graduates from CSIT programs are perhaps the knowledge experts in the areas of curriculum that need to be improved. This paper argues in favor of surveying practicing CSIT graduates in industry to gather pertinent information for revamping curriculum. The paper discusses the results of a survey used to revise the instruction and core curriculum of the degree programs in Network Information Security (NIS) and Computer Network Modeling and Simulation (CNMS) at St. Cloud State University. The paper also presents the elements of a generic tool for a comprehensive student learning outcome assessment.

Categories and Subject Descriptors
k.3.2 [**Computers and Education]**: Computer and Information Science Education *–computer science education, curriculum*.

General Terms: Design, tool

Keywords: Student learning outcome, program review, assessment tool

1. INTRODUCTION

Computer Science and Information Technology departments are periodically required to conduct internal and external program reviews to satisfy accreditation requirements [8]. Today, there are available resources for the philosophy of assessment [3], experiences gained from practical assessments [4], valuable pedagogy for student projects [5], and a step-by-step guide for assessments [21].

Student learning outcome assessment (SLOA) is the organized compilation of information about student learning for decision making on how to improve instruction and learning. Today, accrediting agencies require SLOA. Ideally, every major course in a degree program should have student learning outcomes (SLOs) that map into the program's SLOs. The systematic collection and processing of data about student learning from a variety of courses entail cumbersome processes. Unfortunately, practical self-contained tools to facilitate meaningful graduate follow-up and curriculum assessment at course and program levels are rare. Consequently, we have designed a comprehensive SLOA for the NIS and CNMS degree programs with the following goals:

- To operationally define all competencies and student learning outcomes for all CNMS and NIS courses.
- To review the detailed topics for all CNMS and NIS courses and map them to SLOs.
- To implement a databank for systematically aggregating the overall learning outcomes in CNMS and NIS.
- To develop and administer instruments for assessing the learning impacts of the CNMS and NIS programs on current students and the graduates.
- To develop and administer an instrument for assessing the congruency between the requirements of industries/organizations and the learning outcomes of the CNMS and NIS.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE '09, May 1–2, 2009, British Columbia, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

2. PROJECT ACTIVITIES

We have defined the program outcomes for the CNMS and NIS. The seven specific SLOs established for the NIS degree are:

NIS1: Create, analyze, and modify security policy.
NIS2: Design and implement secure network architecture based on security policies.
NIS3: Demonstrate understanding of security laws and ethics.
NIS4: Identify and correct security weaknesses in operating systems, networks, and applications.
NIS5: Identify and respond appropriately to security breaches.
NIS6: Demonstrate understanding of cryptography and other theoretical foundations of security.
NIS7: Demonstrate knowledge of security protocols used at each layer of the TCP/IP model and how they relate to form layered security architecture.

Presently, students complete core courses in operating systems [6], computer networking, open systems interconnection layer security [10, 15, 17], firewall security [2, 18, 23], computer network security [20], applied cryptography [14, 19], applied public key infrastructure [1,7] and network intrusion detection [9, 12], calculus, discrete mathematics, and applied probability and simulations to satisfy the goals of the NIS SLOs.

The six SLOs defined for the CNMS degree are:

CNMS1: Configure operating system and network programs, using the language of a software package or a general-purpose programming language.
CNMS2: Write correct, well documented and readable programs.
CNMS3: Describe and use microcomputer operating systems.
CNMS4: Identify security weaknesses within open systems interconnection layers and implement security controls for operating systems and communication networks.
CNMS5: Use performance requirements and simulations to design data networks.
CNMS6: Use probability models and simulation to solve problems.

Presently, students complete core courses in operating systems, computer networking, web authoring and administration [11, 13, 16], computer network security, network performance analysis, data network design [22], calculus, statistics for physical sciences, SAS programming, regression analysis, applied probability and simulations, and six hours of additional statistics electives to fulfill the goals of the CNMS SLOs.

We have operationally defined all competencies and SLOs for the CNMS and NIS courses. We have mapped the detailed topics of these courses to SLOs. The general learning outcomes and rubric of the CNMS and NIS have been mapped to the student learning outcomes of the degree programs. For example, Appendix A contains the mapping of the general learning outcomes and rubric of OSI Layer Security (CNA432) to the SLOs of the NIS degree program.

We have implemented a generic database system (GDS) for systematically aggregating the overall learning outcomes in CNMS and NIS. The relational GDS was implemented with macros in Microsoft Access. The database design and descriptions are provided in Appendix C. Appendix D provides a sample of the menus of the GDS equipped with graphical user interfaces. The database supports data entries at individual course level, and queries and reports such as:

a. For each course and each rubric, generate the student assessment report, along with the descriptive statistics (mean, standard deviation). In the report, an asterisk (*) is displayed besides the record of each student who scored two standard deviations below the mean.

b. For each program's student learning outcome (each NISID, CNMSID), generate courses (CORID) that assess the skills. Provide also a summary report of the frequencies and totals.

c. For each program's student learning outcome, generate just those courses (CORID) and rubric (RUBID), and identify students who scored two standard deviations below average. A rubric is the set of specific activities students perform to demonstrate the knowledge of a learning outcome. Each learning assessment rubric has one or more clearly defined grading scales. Appendix A provides examples of assessment rubric and rubric scales.

We developed and administered the survey instrument of Appendix B to assess the learning impacts of the CNMS and NIS programs on the graduates, and to assess the congruency between the requirements of industries/organizations and the learning outcomes of the CNMS and NIS. Although we mailed out over 75 surveys, we have only received 20 responses to date. The GDB system we implemented has an interface for data entry and automatic analysis of the survey responses.

3. GRADUATE SURVEY RESULTS

3.1 Profile of Graduates

Of the twenty survey responses, 11 (55%) graduated from the Network Modeling and Simulation (NMS) major, 8 (40%) indicated Computer Networking and Application (CNA) as their major, and 1 (5%) double majored in NMS and NIS.

The graduates work at different small, medium and large businesses and organization including: Boston Scientific Corporation, Convey, City of Willmar, GeoComm Inc, IBM, ISECURE trac, Ingusal Rand, Park Nicillet Healt Services, Relco LLC, SCSU, Secure Computing, 3 Sigma Developer, Skyline Displays, St. Jude Medical, Target, Thomson West, US Bank, Video Furnace and Woltas Kluwer.

The graduates work as application and software developers, network and software engineers, IT technicians, IT support specialists, systems and database administrators, systems analysts, and application support engineers.

3.2 Graduate Responsibilities, Tools and Environment

The numbers and percentages of graduates who indicated programming, network administration, networking monitoring, technical support and network security as parts of their responsibilities were 11 (55%), 7 (35%), 7 (35%), 7 (35%) and 6 (30%) respectively.

Twelve or 60% of the graduates are involved in only one type of duty, and eight or 40% are engaged in two to five types of duties.

The graduates are programming in AJAX, C, C#, Java, Javascript, C Shell Scripting, VB Scripting, VB.Net, XML, and SQL.

The graduates work with Visual Studio and Net Beans. They work with DB2, Oracle, and MYSQL databases, and MS SQL servers.

The graduates work on Windows XP, Windows Vista, Windows Server 2003 and 2008, Linux, UNIX, AIX, Solaris and Mac OS X platforms.

3.3 Graduate Perception of CNA Instruction

On a scale of 1 (least helpful) to 5 (most helpful), the average ratings of lectures, in-class discussion, in-class computer activities, in-class group activities, out-of-class group projects, and out-of-class individual projects by the graduates were 3.25, 3.95, 4.35, 4.1, 4.25 and 4.58 respectively.

The graduates perceived group projects, blend programming and theory, and hands-on lab activities as most helpful.

The least helpful educational experiences identified by graduates include: the traditional lecture style classes, too much theory, too much statistics, insufficient courses in the major, and unenthusiastic professors.

3.4 Graduate Recommendations

The graduates recommended we provide more real-world and group exercises and projects, offer classes or tailor selected courses toward the preparation of graduates for certification, introduce more database concepts and programming, and provide more hands-on network administration and programming.

3.5 Departmental Observations and Recommendations

A significant number of the graduates are engaged in programming. The Department now recognizes the need to develop and implement strategies for introducing more programming into CNA courses.

The graduates rated the in-class discussion as the least helpful for their training. The Department now understands the need to implement strategies for making classroom interactive. More CNA courses would be offered with more hands-on projects in electronic classrooms and laboratories.

The graduates recommended fewer statistics courses and more CNA courses in the major. The Department previously already recognized the need to revise the Network Modeling and Simulation degree requirements to include less statistical theory and more security courses.

The graduates would like to be prepared for certification upon graduation. The Department now sees the need to include topics tested in certification exams in selected CNA courses, to offer a course targeted toward certification, or to implement a self-paced online certification course.

4. CONCLUSIONS

The SLOA project offers new opportunities for faculty members to systematically connect tests, quizzes, assignments and projects in CNMS and NIS courses to program and course learning outcomes. The SLOA project offers the department a unique opportunity to begin to use a database to target specific skill areas of CNMS and NIS where students are deficient in learning.

Beginning in fall 2007, the CNMS and NIS programs started implementing the detailed learning outcomes and rubrics into individual courses. Since 2002, students have been questioning the relevance of the many statistics courses required in the CNMS degree program. We were not able to respond to this issue because we relied on the faculty strength in statistics and had no fully developed network security courses to use as substitutes for the required statistics courses. Armed with the graduate follow-up survey results in fall 2008, the CNA faculty reviewed the curriculums and teaching styles in the CNMS and NIS programs. Recognizing that students should be skilled in security design to protect core information while accomplishing the network design and management goals of an organization, we replaced the majority of the statistics courses with career-oriented network information security courses.

The SLOA project has helped us to develop a model for implementing student outcome assessment at the course and program levels. The project has enabled us to develop a generic database that supports assessment in a variety of programs.

5. REFERENCES

[1] Adams, C. and Lloyd, S. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, 2003

[2] Anderson, R. J. Security Engineering – *A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001

[3] Astin, A. W. *Assessment for Excellence: The Philosophy and Practice of Assessment and Evaluation in Higher Education.* American Council on Education Series on Higher Education, Phoenix: Oryx Press (1993)

[4] Astin, A. W. "Involvement in learning revisited: lessons we have learned." *Journal of College Student Development*, 37(2), 123-134 (1996)

[5] Batra, M. M., Walvoord, B. E., Krishnan, K. S. "Effective pedagogy for student team projects." *Journal of Marketing Education*, 9(2), 26-42 (1987)

[6] Barrett, D. J., Silverman, R. and Loukides, M. *SSH the Secure Shell: The Definitive Guide*. O'Reilly UK, 2001

[7] Boyd, C. and Mathuria, A. *Protocols for Authentication and Key Establishment*. Springer-Verlag, Berlin, Germany, 2003

[8] Computer Science Accreditation Board and ABET, http://www.csab.org/

[9] Doraswamy, N. and Harkins, D. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall, 1999

[10] Ford, W. and Baum, M. S. *Secure Electronic Commerce*. Prentice Hall, 1997

[11] Garfinkel, S. and Spafford, G. *Web Security, Privacy & Commerce*. O'Reilly, Cambridge, MA, 2001

[12] Kosiur, D. *Building and Managing Virtual Private Networks*. John Wiley & Sons, 1998

[13] Lawrence, E., Corbitt, B., Fisher, J., Lawrence, J. and Tidwell, A. *Internet Commerce*. John Wiley & Sons, 1999

[14] Mao, W. *Modern Cryptography – Theory & Practice.* Prentice Hall, New Jersey, USA, 2004

[15] Merkow, M. S., Breithaupt, J. and Wheeler, K. L. *Building SET Applications for Secure Transactions*. John Wiley & Sons, 1998

[16] Rubin, A. D., Geer, D. and Ranum, M. J. *Web Security Sourcebook*. John Wiley & Sons, 1997

[17] Ryan, P. and Schneider, S. *Modeling and Analysis of Security Protocols*. Addison-Wesley London, UK, 2001

[18] Sherif, M. H. *Protocols for Secure Electronic Commerce*. CRC Press, 2000

[19] Singh, S. *The Code Book*. Fourth Estate, London, UK, 1999

[20] Stallings, W. *Cryptography and Network Security: principles and practice*, Prentice Hall, Upper Saddle River, N.J., 2006

[21] Walvoord, B. E. *Assessment Clear and Simple: A Practical Guide for Institutions, Departments, and General Education.* John Wiley & Sons, Inc., San Francisco, CA (2004)

[22] Winfield Treese, G. and Stewart, L. C. *Designing Systems for Internet Commerce*. Addison-Wesley, 2002

[23] Zwicky, E. D., Cooper, S., Chapman, D. B. and Russell, D. *Building Internet Firewalls*. O'Reilly, UK, 2000

APPENDIX A: Mapping of CNA432 General Learning Outcome and Rubric to CIS SLOs

CNA 432 OSI Layers Security Outcome Assessment

The Course Description

Security models and protocols for each Open System Interconnection layer; network and web security implementation, monitoring, intrusion, recovery and countermeasures

Prerequisite: CNA 426 or BCIS 353 or consent of instructor.

NIS SLO and CNA432 SLO Mapping

NIS1	CNA4321
NIS1	CNA4322
NIS5	CNA4323
NIS7	CNA4324
NIS7	CNA4325
NIS2	CNA4326
NIS4	CNA4327
NIS4	CNA4328
NIS4	CNA4329
NIS3	CNA43210
NIS5	CNA43211

The General Learning Outcomes

CNA4321: Understand CIA and PPP triads, risk assessment, security policies and security resources
CNA4322: Understand the importance of security education, advisory and issue management, risk and incident management
CNA4323: Identify attacks to IT and Office, and treat taxonomy
CNA4324: Install network devices, addressing and defense in depth
CNA4325: Evaluate packet sniffers, threats and solutions to TCP/IP and wireless networks
CNA4326: Design IDS, NIDS, HIDS and Honeypots architecture and methodologies
CNA4327: Manage preventive, detective and corrective security features for a Linux LAN
CNA4328: Install preventive and detective measures for UNIX
CNA4329: Assess preventive and detective features of Windows
CNA43210: Apply policy verification, security standards and audits, audit process and action
CNA43211: Understand legal concerns, defense probing, and exploitation of security vulnerabilities

The Assessment Rubric

RUB4321: Compute annualized loss expectancy, annual rate of occurrence and total costs of ownership for alternative business assets → Use **RS1**
RUB4322: Build Red Hat Linux security resource center → **RS2**
RUB4323: Use business intelligence as an attack against a company → **RS3**
RUB4324: Install and configure IP tables to simulate a Firewall → **RS2**
RUB4325: Assess wireless network vulnerabilities at SCSU, and for SNMP → **RS3**
RUB4326: Design and implement a protected network → **RS2**
RUB4327: Administer security for servers running Linux → **RS2**
RUB4328: Install OpenSSH client and authentication → **RS2**
RUB4329: Investigate Firewall on Windows Server 2003 → **RS3**
RUB43210: Evaluate TCSEC and ISO requirements and audit reports for Windows → **RS3**
RUB43211: Configure and use nessus utility to uncover vulnerabilities → **RS2, RS3**

The Rubric Scales

RS1:
- Excellent (A) = the student achieves an average of 9 or 10 points in three ALE, ARO, TCO exercises.
- Good (B) = the student achieves an average of 7 or 8 points in three ALE, ARO, TCO exercises.
- Fair (C) = the student achieves an average of 5 or 6 points in three ALE, ARO, TCO exercises.
- Poor (D) = the student achieves an average below 5 points in three ALE, ARO, TCO exercises.

RS2:

- Excellent (A) = the student achieves 9 or 10 points in the installation, testing and evaluation of the project.
- Good (B) = the student achieves 7 or 8 points in the installation, testing and evaluation of the project.
- Fair (C) = the student achieves 5 or 6 points in the installation, testing and evaluation of the project.
- Poor (D) = the student achieves below 5 points in the installation, testing and evaluation of the project.

RS3:

- Excellent (A) = the student achieves 18, 19 or 20 points in a test designed to assess mastery of the terms, concepts and issues.
- Good (B) = the student achieves 14, 15, 16 or 17 points in a test designed to assess mastery of the terms, concepts and issues.
- Fair (C) = the student achieves 10, 11, 12, or 13 points in a test designed to assess mastery of the terms, concepts and issues.
- Poor (D) = the student achieves below 10 points in a test designed to assess mastery of the terms, concepts and issues.

Appendix B: Computer Networking and Applications Graduate Survey

1. What year did you graduate? ________________________

2. What was your major?
____

3. What is your current job title:
____

4. Employer:
____

5. Please select your appropriate job functions from the list below:

a) Networking
- Network administrator
- Network design and analysis
- Network monitoring, Network management
- Security

b) Programming and software development
- What programming language do you use?
- What development environment do you use?

c) System administration
- What tools do you use?
- What databases do you use?
- What Operating System do you use?

d) Technical support
e) Others (please specify)

6. If you were starting now with a CNA major, which styles of teaching do you think would best help you in grasping computer skills and concepts? Please rate each teaching/learning method on a scale of 1–5, with 5 representing most helpful.

	Rating
Lecture	___
In-class discussion	___
In-class computer activities	___
In-class group activities	___
Out-of-class group projects	___
Out-of-class individual projects	___

7. Overall, what was most helpful in your CNA major?

____

8. What was least helpful?

____

9. What should be added to the CNA major?

____

10. Did your CNA major help you get a job? yes no If so, how?
____

____

11. (Optional). We are also interested in getting the perspective of employers and supervisors in assessing the CNA major. If you would feel comfortable doing so, please give the name and complete address of your current supervisor or employer.

____

____

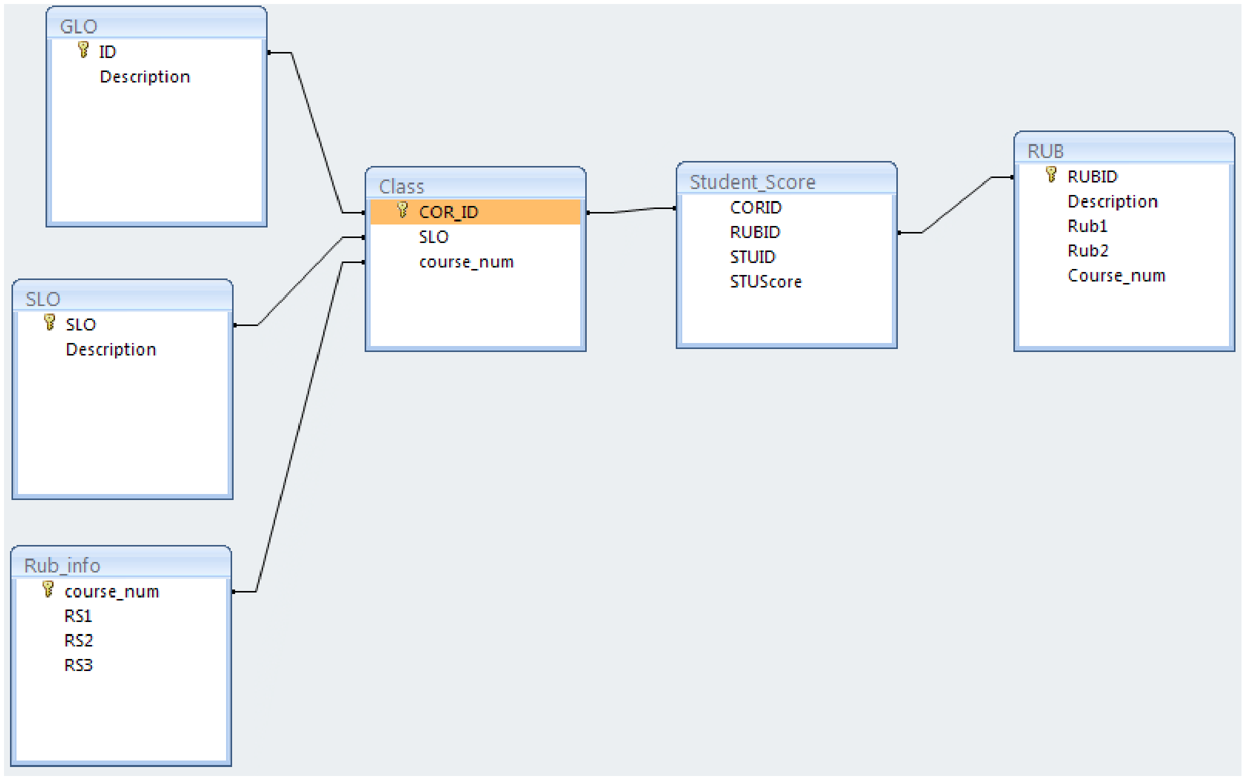
____

____

____

____

Appendix C: Database Layout and Relationships



GLO Table
This table stores the information about General Learning Outcomes (GLO). It contains the primary keyID, and the description of the GLO.

SLO Table
This table stores the information about the Student Learning Outcomes (SLO). This table uses field SLO as the primary key and also contains the description of the SLO.

Rub_info Table
This table stores the grading information of each Rubric. The memo fields RS1, RS2, and RS3 are the rubrics. This table also contains the course ID used to link the rubric scales (RS fields) back to a class.

Class Table
This table has 3 fields, Cor_ID, SLO and course_num. The Cor_ID field is connected to the GLO table and the student_score table. The SLO field is linked to the SLO table to obtain the description, and the course_num is connected to the Rub_info table.

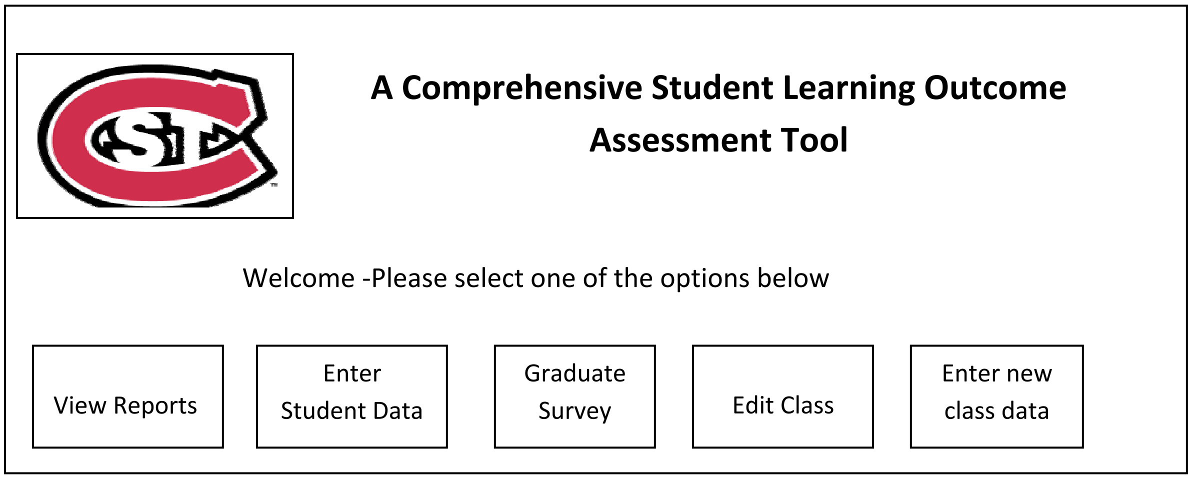
Student_Score Table
This table stores all information of the students, including the scores. It has CorID, RubID, StuID, and STUScore fields. The CORID field is linked back to the Class Table; the RubID field is linked to the Rub table. The StuID and StuScore fields are used to store the information about each student and associated scores.

Rub Table
This table stores information about each Rubric. The only Key, RubID is connected to the Studen_Score table. It contains the description of each rubric; Rub1 and Rub2 fields are used to store information for processing multiple rubric scales.

Macros
The macro codes of any form can be viewed in the design view, including comments on its functionality.

APPENDIX D: Graphical User Interface of the Generic Database for SLOA

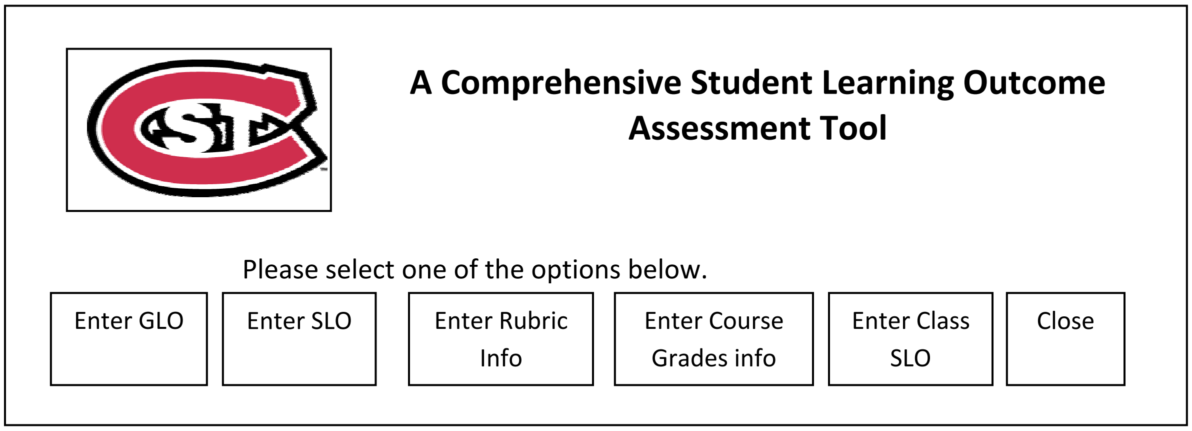
Figure 1 Main Menu of the Learning Outcome Assessment Tool



A Comprehensive Student Learning Outcome Assessment Tool

Welcome -Please select one of the options below

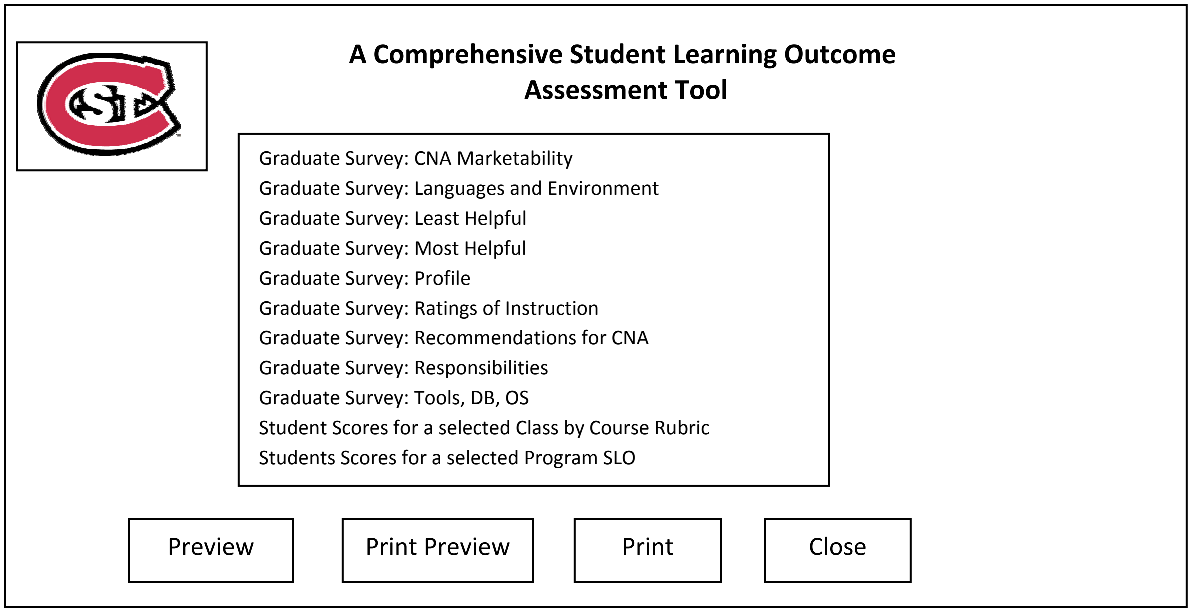
Figure 2 Menu for Data Entry of Student Learning Outcome



A Comprehensive Student Learning Outcome Assessment Tool

Please select one of the options below.

Figure 3 Menu for Printing Student Learning Outcome



A Comprehensive Student Learning Outcome Assessment Tool

Graduate Survey: CNA Marketability
Graduate Survey: Languages and Environment
Graduate Survey: Least Helpful
Graduate Survey: Most Helpful
Graduate Survey: Profile
Graduate Survey: Ratings of Instruction
Graduate Survey: Recommendations for CNA
Graduate Survey: Responsibilities
Graduate Survey: Tools, DB, OS
Student Scores for a selected Class by Course Rubric
Students Scores for a selected Program SLO

An experiment with online instruction and active learning in an introductory computing course for engineers

JiTT meets CS1

Paul Carter
Department of Computer Science
University of British Columbia
Vancouver, Canada
pcarter@cs.ubc.ca

ABSTRACT
This paper describes an experiment with online instruction and active learning in an introductory computing course for engineering students. A series of online screencasts was developed to replace the traditional component of lectures for a one week period in the middle of the course. Students were asked to review the screencasts before coming to class and were assessed on that material at the beginning of class. A "just in time" lecture was provided in response to the assessment, as needed. The remaining class time was devoted to associated active learning exercises through peer instruction with formative assessment. Student feedback on this preliminary experiment was positive suggesting that further investigation is warranted.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms
Human Factors

Keywords
Online instruction, screencasts, active learning, JiTT, peer instruction, CS1, pedagogy

1. INTRODUCTION
We describe an experiment that combines just-in-time teaching (JiTT)[8] with peer instruction [3, 7] in an introductory computing course for engineering students at a large research intensive university. In brief, JiTT uses online learning technologies to supplement in-class activities. Typically, students engage in some kind of pre-class activity, such as textbook readings, coupled with web-based assessments. These assessments allow the instructor to focus subsequent class time on concepts that students find particularly difficult. Traditional lecture is significantly reduced allowing for more time to be devoted to active learning. Peer instruction is an example of one such learning activity in which students are presented with concept problems that they work on in small groups. These problems typically take the form of multiple choice questions followed by immediate feedback by way of a show of hands or more sophisticated technology. This feedback allows the lecture to be further tailored to student learning. JiTT and peer instruction were originally developed for use in introductory Physics courses. In this paper we describe an adaptation for use in a CS1 course.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7…$5.00.

The use of JiTT and peer instruction in Computer Science education is not new. However, JiTT appears not to have been widely adopted in the Computer Science domain, especially in CS1. Bailey et al. [1] reported on the use of JiTT and peer instruction in a CS0 course where students were asked to complete reading assignments and warm-up exercises as a form of pre-instruction. Davis [4] noted positive experiences with JiTT in systems and design courses. Pargas [9] reported on the use of pre-lecture reading assignments and peer instruction in a sophomore-level data structures and algorithms course. The experiment at hand examines the use of JiTT with online screencasts, rather than reading assignments, as the form of pre-instruction in a large CS1 class.

The course under consideration provides an introduction to software development in C with applications in data acquisition and analysis. This experiment was conducted during a one week module on arrays. Online screencasts were developed that present core factual information about arrays and a minimal set of examples to illustrate how they are coded in C. Students were asked to study the screencasts prior to coming to class and were told that they would be assessed on their comprehension of the material at the beginning of class using a Personal Response System (PRS). These systems allow students to respond to multiple choice questions using a small electronic device, often referred to as a "clicker". Aggregate responses can be shown to the class in real time, often sparking heated debate in cases where a question has split the class vote. Such systems serve to increase the level of engagement, especially in large classes where some students may be reluctant to respond to questions verbally. The results of the PRS assessment were used to guide a "just-in-time" lecture. This initial assessment consumed about 20 - 25% of class time allowing the remaining time to be devoted to associated active learning exercises, formative assessment and additional mini-lectures as needed. The online pre-instruction therefore allowed for a dramatic shift away from lecturing and towards active learning.

Given the dynamic nature of program execution, we believe that students are likely to learn more effectively when multimedia presentations are used to complement static text. Code animations were used to illustrate how to iterate over an array, for example. Online screencasts as a form of pre-instruction in an introductory physics course have been reported by Stelzer et al. [11]. They compared the use of multimedia with traditional texts as a form of pre-instruction and reported that students who received multimedia instruction performed significantly better on immediate post-tests and on tests of retention conducted two weeks later.

A survey was conducted at the end of our experimental module and slightly more than 50% of students who expressed an opinion indicated that they preferred the new approach to teaching and learning. Given that the experiment was conducted part way through the second term of the year and that students often need time to adapt to new techniques, this result is encouraging.

2. MOTIVATION

The course is required in all engineering programs resulting in an overall enrolment of between 300 and 400 in each of two terms. The students are distributed across two sections in each term. The majority of students in the course are first year. The course is co-taught by the departments of Computer Science and Electrical & Computer Engineering with both sections of the course in a given term taught by the same instructors. The first eight weeks of the course are taught by the Computer Science department and the experiment at hand was conducted during this segment of the course.

The author was one of the original course developers and has taught the course on a regular basis since its inception in 2004. Increasing levels of disengagement by students have been perceived, especially in recent years, with many students (around 25%) not attending lectures on a regular basis or even at all. This is in spite of the author's best efforts to provide an interactive lecture that included the use of a PRS to promote student engagement, that allowed for questions from students on a regular basis and that incorporated occasional in-class problem solving exercises.

In this study we attempt to assess student reaction to the use of JiTT with online pre-instruction and in-class peer instruction for a one week period in the middle of the course.

3. EXPERIMENT

3.1 Design

The course is split into two parts with the author teaching the first eight weeks of the thirteen week course. A midterm break, called "Reading Week", occurs after the first six weeks of the course and the experiment was conducted the week following the break.

A series of six screencasts was developed that presented the concept of one and two dimensional arrays, the corresponding syntax in C and the use of `for` loops (not previously introduced in the course). Each screencast was two to five minutes in duration and consisted of a screen capture of a PowerPoint presentation complete with voice-over. The presentations included animations and a minimal set of examples. The screencasts were made available on the course webCT site in two formats: Flash video and mp4 (for iPod). The Flash version included closed captioning for students who were using computers without an audio system. This also helped to ensure that the online materials were accessible to hearing impaired students. At this point the reader is encouraged to view the introductory screencast on arrays available at: `http://www.cs.ubc.ca/~pcarter/WCCCE09/arrays.html`

A set of in-class exercises was also developed that allowed students to explore the application of concepts presented in the online screencasts to specific problems. These exercises were not available to students before class and were presented in hard copy format in class. Exercises for each topic typically began with simple code comprehension and code completion tasks, and advanced to writing short programs or functions from scratch.

3.2 On the use of PRS

The PRS was used on a voluntary basis in the course. Answers to PRS exercises throughout the term contributed to 5% of the overall course grade for students who elected to use the system. Each PRS question received 2 marks if answered correctly and 1 for any other answer. At the end of the term, students received the higher of the two marks computed by (i) counting PRS exercises as 5% of the overall mark and (ii) counting PRS exercises as 0% of the overall mark and transferring the 5% weight normally assigned to PRS to the final exam. In the term under consideration, 262 of 340 students scored at least 50% on the PRS component in the first segment of the course indicating that a large number of students were actively involved in this voluntary activity.

3.3 Implementation

We describe the implementation of the experiment in a class that meets twice a week for 75 minutes. Students were asked to study three of the six screencasts prior to attending each of the two classes. Each class started with five PRS questions that aimed to assess students' understanding of fundamental aspects of arrays (e.g., declaration, indexing, arrays as parameters). Students were asked to respond individually without discussion. The instant assessment offered by the PRS system allowed for immediate intervention if the results suggested that students had failed to understand a concept. Sometimes the intervention consisted of nothing more than working through a solution to the specific assessment exercise. This initial assessment and intervention occupied about 20 minutes of class time.

The average score on the assessment was 70.8% for the first class of the week and 65.7% for the second class. These scores represent correctness only and do not include marks for participation. Given that no traditional instruction had taken place at that point, the results suggest that many students had prepared for class, presumably by studying the online screencasts.

The remaining class time (about 55 minutes) was devoted to peer instruction through active learning exercises with additional mini-lectures, as needed. Students were asked to work together in groups of two to four. The groups were self-selected and usually consisted of students already sitting in close proximity. The exercises were intended to lead students to think about arrays on a deeper level. The peer instruction model described in [7] asks students to first think about a problem individually and then discuss the answer with a neighbour. Given that problems involving code writing and comprehension can take at least 10 minutes to complete, students were asked to work as a group from the start. While students worked on problems, the instructor had the opportunity to circulate through the class, check on the progress of various groups and answer questions.

Some of these exercises consisted of, or were coupled with, multiple choice questions that were immediately assessed with PRS. Four PRS questions were asked during each of the two classes with students scoring over 78% on average for correctness. The results of the assessment guided the need for further instruction.

In addition to helping the instructor determine the need for intervention, the PRS questions offer formative assessment that helps students to gauge their level of understanding. As noted by Bransford et al. "Feedback is most valuable when students have the opportunity to use it to revise their thinking as they are working on a unit or project." [2, p.141]. Students who answer a question incorrectly realize that there's an important concept that they don't understand and are more likely to pay attention to, and extract relevant information from, the subsequent mini-lecture. Mini-lectures were delivered to the entire class but students who had answered correctly were encouraged to start working on the next question rather than focus on the lecture.

Questions that ask students to write code from scratch are more difficult to assess with a PRS. In these cases, students were encouraged to compare answers with neighbouring groups and to ask for help from the instructor when necessary. Solutions to some of these problems were presented in class.

Different groups tended to work at different rates. Those that finished a problem quickly were encouraged to move on to the next problem. In some cases, students finished all the problems before the end of class and approached the instructor for more.

3.4 Examples

The questions asked at the beginning of class were intended to assess only the rudimentary aspects of arrays, as presented in the online screencasts. The following is an example of a question that revealed fundamental misconceptions, not only with the topic at hand but also with a topic covered earlier in the course.

> What is the output from the following code segment?

```
int data[] = { 4, 3, 1, 7, 3 };
int index = 1;
while( index < 4 )
{
    printf( "%i", data[ index ] );
    index++;
}
a)  4317
b)  3173
c)  431
d)  317
e)  none of the above
```

Only 50.5% of students chose the correct answer while 25.2% of students chose (c) (indicating that they had failed to grasp the concept of zero-based indexing) and 14% chose (b) (indicating an off-by-one error while tracing the loop). A further 6.5% of students chose (a) suggesting a combination of the aforementioned problems. This is an example of a situation where an intervention in the form of a mini-lecture was used, in this case to clarify zero-based indexing and to trace through the code with the students.

The active-learning exercises used during the rest of the lecture were intended to guide students beyond the basics presented in the online screencasts. For example, one question asked students to trace a piece of code and was intended to determine whether or not students understand the difference between passing primitive types vs. arrays as parameters. A PRS assessment indicated that 72.6% of students determined the output from the code correctly when working in small groups. Given that the concept of passing an array as a parameter had been presented to students only in the online screencasts, this again suggests that students had engaged in pre-class activities.

4. STUDENT REACTION

4.1 Methodology

A survey was conducted at the end of the last class during the week in which this experiment took place. Responses were collected using PRS but the results were not shown to students until the following class. This was done so that results from earlier questions in the survey would not impact responses to later questions. The PRS used does not have an anonymous response option and this must be kept in mind when interpreting the data collected. Students were asked to respond to statements using a Likert-like scale: Strongly Agree (SA), Agree (A), Disagree (D), Strongly Disagree (SD) and No Opinion (NOP).

In addition, an anonymous open forum was created on webCT that allowed students to provide free-form comments on their experience with the new pedagogy. The response rate was very low (only 15 students posted) but some of the comments received are interesting.

4.2 Survey Results

We present the combined results of the survey conducted in both sections of the course. Approximately 250 responses were received to each question. Recall that 262 of the 340 students registered in the course were actively using the PRS. The results are shown in Table 1.

Table 1: Survey results.

Statement	SA	A	D	SD	NOP
For the rest of the course, I'd prefer that lectures be delivered in the format used before Reading Week rather than in the format used this week.	20.9%	22.5%	22.9%	25.7%	7.9%
I learn more when you use the class time to give a lecture (as we did before Reading Week) than by watching online videos and working on problems in class (as we've done since Reading Week).	17.3%	21.3%	26.9%	22.5%	12.0%
The online videos did not help me understand the concept of arrays and how to use them.	5.8%	12.1%	45.9%	31.1%	5.1%
The exercises given in class this week did not help me understand the concept of arrays and how to use them.	3.6%	7.6%	35.6%	48.4%	4.8%

The first survey statement attempts to assess students' preferences when comparing the new pedagogy to the earlier more traditional pedagogy. We observe close to a 50-50 split with slightly more students favouring the new approach. Given that the experiment took place after six weeks of conventional instruction and in the second term of the academic year, it is encouraging that so many students looked favourably on the new approach.

The second statement attempts to assess students' perceptions about learning when comparing the new approach to the more traditional pedagogy used earlier in the course. Again, we observe that slightly more than half of the students who had an opinion indicated that they believe they learned more with the new approach.

The remaining two statements attempt to assess the perceived effectiveness of the online screencasts and in-class exercises with regard to learning. There is a strong indication that students believed that these elements helped them to understand the concept of arrays and how to apply those concepts in practice.

One concern is that this approach appeals only to stronger students. To test this we coded the survey responses SA, A, D and SD with the scores 1, 2, 3 and 4, respectively. Given that survey results were not anonymous we were able to compute a correlation coefficient between the coded survey response and the overall course grade for each survey statement. Students who responded "No Opinion" were excluded. The correlation coefficients were small (ranging from 0.018 to 0.344) suggesting that survey responses that favoured the new approach did not come from one particular group of students with respect to overall course grade.

4.3 Anonymous Forum Results

We now turn to some of the comments posted in the anonymous open forum. As mentioned earlier, the response rate was very low but some interesting comments were received.

> "I liked the videos because they gave you a chance to absorb the material and new techniques before we had to discuss them in class. Using the class to go through examples also strengthened our understanding of the key concepts, as opposed to just listening to some guy talk about them..."

This student's comment articulates many of the anticipated benefits of the experimental approach. It must be noted that no commentary had been provided to students about these expected benefits, so this is not a case of the student repeating something that had been presented by the instructor.

> "I really liked the style of teaching with the videos. The videos themselves were useful so that I could go at my own pace... ...I also found the follow up that we did in class helpful. It allowed me to confirm what I knew and practice it, while learning a few more things that I may have missed. "

This student makes the important observation that the experiment consisted of two significant parts from the student perspective: preparation before class and in-class active learning exercises. Other students also commented on the fact that the videos allowed them to go at their own pace.

> "...the classes were much more interesting doing examples. I know that I would often fall asleep in the regular lectures, but I was awake and alert throughout the whole class in this format (which isn't an easy task at 8 am). "

This comment is certainly in keeping with observations made during class where students seemed to be much more responsive to the active learning approach and energy levels in the classroom were noticeably higher.

> "I think it would be more effective if class time was split so that half the time we have the lecture portion and the other half we have in-class exercises. I found that doing in-class exercises helped me understanding things more because I learned from my mistakes. But at the same time I found spending the whole lecture working on exercises really draining and at times overwhelming when there is a time limit."

This student expresses the fact that the new pedagogical approach requires more energy and effort from students. The student recommends a combination of traditional lecture and in-class exercises, presumably disposing of the need to do the preparatory work and reducing the "active" part of the classroom experience. It is not surprising to receive this kind of feedback. In fact, any new pedagogical approach that requires more from students is bound to be met with some resistance. The question of whether the benefits outweigh the costs in terms of student learning will be investigated in a future study.

4.4 Student Learning

The goal of this study was to examine student reaction to a different pedagogy. However, the ultimate goal is to improve student learning. Given that this experiment took place during a one week segment of the course, it is difficult to assess the impact on student learning given the high dependence on material taught earlier in the course. That said, we provide a few data points that, at the very least, suggest that the new approach did not have a detrimental effect on the students' comprehension of arrays.

A question similar to the one presented at the beginning of section 3.4 was asked on the final exam. This exam was closed book, so students did not have access to notes or textbooks. Recall that when used as an assessment of pre-learning, only 50.5% of students answered this question correctly. On the corresponding final exam question, 88.8% of students answered correctly.

On the final exam, a Parson's puzzle [10] presented students with a set of paired statements. Students were asked to select one statement from each pair to construct a function that determines if a particular item is found in a given array. The class average on this question was 84.8%. An analysis of Parson's problems in comparison to code tracing and code writing questions by Denny et al. [5] indicated that students performed better on a Parson's problem than on the other two styles of questions. This is in part due to the fact that simple guesswork may account for a portion of the marks received. We must keep this in mind when interpreting the high scores earned by students on this question.

One of the in-class active learning exercises asked students to write a function that returns true if a given array is in ascending order and false otherwise. A follow up to this exercise was asked in the form of a multiple choice question on the final exam. The question was a translation from Java to C of the "sorting" question presented by Lister et al. [6] and also used in the study by Tew et al. [12]. Students were asked to select code that represented the body of a function that returns true if the data in a given array is in descending order and false otherwise. This represents a slight modification to the question asked in class and to that presented by Lister which asked students to select code that determines if the elements were in *ascending* order. Lister's multi-institutional study reported that 42% of students chose the correct answer. Tew found that when the question was used as a pre-test at the beginning of the second course in computing, 26.22% chose the correct answer, while 44.30% chose the correct answer when the question was used as part of a post-test at the end of the course. In the current study, the class average on this question was 49.3%. It must be noted that in the other studies, the extent to which this particular problem had been studied during the course was not stated, whereas in the current study the problem had been explicitly presented to students in class as an active learning exercise.

While these isolated data points are not sufficient to draw conclusions about the effect on student learning, they do suggest that the approach was not detrimental and that further investigation is justified.

5. CONCLUSIONS AND FUTURE WORK

The results from this experiment are encouraging. We have seen positive student reaction to the pedagogical approach described. We intend to develop materials necessary to teach the entire segment of the course presented by the Computer Science department using this new approach. In addition to a further assessment of student perceptions, we will also engage in quantitative research to assess the impact on student learning.

6. ACKNOWLEDGMENTS

This study was conducted with support from the Carl Wieman Science Education Initiative at the University of British Columbia. I would like to express my thanks to Carl Wieman and Beth Simon for encouraging me to publish this work and for providing valuable feedback on early drafts.

7. REFERENCES

[1] T. Bailey and J. Forbes. Just-in-time teaching for CS0. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 366–370, New York, NY, USA, 2005. ACM.

[2] J. D. Bransford, A. L. Brown, and R. R. Cocking. *How People Learn, Expanded Edition.* National Academy Press, Washington, D.C., 2000.

[3] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–977, September 2001.

[4] J. Davis. Experiences with Just-in-Time Teaching in Systems and Design Courses. In *SIGCSE '09: Proceedings of the 40th SIGCSE technical symposium on Computer science education*, 2009, to appear.

[5] P. Denny, A. Luxton-Reilly, and B. Simon. Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on Computing education research.* pages 113–124, New York, NY, USA, 2008. ACM.

[6] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Mostrom, K. Sanders, O. Seppala, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin*, 36(4):119–150, December 2004.

[7] E. Mazur. *Peer Instruction: A User's Manual.* Prentice Hall, New Jersey, 1997.

[8] G. M. Novak, E. T. Patterson, A. D. Gavrin, and W. Christian. *Just-In-Time Teaching: Blending Active Learning with Web Technology.* Prentice Hall, New Jersey, 1999.

[9] R. P. Pargas. Reducing lecture and increasing student activity in large computer science courses. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 3–7, New York, NY, USA, 2006. ACM.

[10] D. Parsons and P. Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australian conference on Computing education*, pages 157–163. Australian Computer Society, Inc., 2006.

[11] T. Stelzer, G. Gladding, J. Mestre, and D. T. Brookes. Comparing the efficacy of multimedia modules with traditional textbooks for learning introductory physics content. In *American Journal of Physics*, 77(2):184-190, February 2009.

[12] A. E. Tew, W. M. McCracken, and M. Guzdial. Impact of alternative introductory courses on programming concept understanding. In *Proceedings of the 2005 international workshop on Computing education research*, pages 25–35. ACM, October 2005.

Teaching the Unifying Mathematics of Software Design

Emil Sekerinski
McMaster University
Hamilton, ON, Canada
emil@mcmaster.ca

ABSTRACT

We report on our experience on teaching the mathematics of reliable software design as a unifying force for various elements of software design, rather than as an additional element of software design. This is in line with the use of mathematics in traditional engineering disciplines, but in contrast to teaching a "formal method" optionally after an "informal" exposition to software design or teaching a formal method only with specific applications in mind.

1. INTRODUCTION

Whereas in the design of a mechanical device breaking design rules would quickly lead to recognizable failure, one can very well break rules of software design and still get a "sufficiently functional" and marketable product. Qualities of software are not as evident or measurable as qualities of physical products; *design qualities* are even harder to judge than the qualities that can be observed of a product. Students follow the rules of software design because they are told so and not because they would experience the consequences of not doing so. Students grow up with unreliable software to the extent that they consider such poorly working software to be normal or unavoidable. Job advertisements suggest that programming skills are sufficient to write software and students take software design courses with the expectation of preparing them for the job market. All this makes is difficult to convince students that software can be better designed, that it is worth doing so, and that it is worth learning the mathematics for doing so.

Typical textbooks on software design, or software engineering as they are sometimes called, devote few chapters on formal specification and verification, with the rest of the book not depending on those. Curricula in computer science and software engineering may include an upper level elective on specification, verification, or semantics. Students perceive the mathematics of software design as optional to their education and largely as a burden.

We report on our experience in teaching a sequence of two courses in software design in which mathematics is used as the unifying force of all core elements of reliable software design. This is in contrast to teaching a "formal method" optionally after an "informal" exposition to software design: our students had only taken an introductory first year course in programming and a course in discrete mathematics. The approach is more in line with the use of mathematics in traditional engineering disciplines. The two courses cover all core topics in software design rather than a specific topic for which a dedicated formalism or tool exist:

Uniform Design Notation A uniform textual design notation is used, in order to emphasize the similarities among the concepts and help students interconnect these concepts, rather than making students switch to a new mindset due to the notational differences. Graphical notations like flowcharts, class diagrams, and statecharts are presented as appropriate and defined as alternative representations of specific aspects in terms of the textual notation

Uniform Mathematical Basis A mathematical basis for all design constructs is given. A typed logic using the same type system as a programming language is used. Equational reasoning is used for all proofs because of the familiarity from calculus. Weakest preconditions are used for statements.

Middle-out Sequencing of Topics Courses on software design are typically structured according to the phases in which software is developed. However, students who have only written small programs so far, do not see the need for, say, elaborate requirements. Instead, we start with writing and analyzing small programs and gradually move to topics to which students become motivated, approximately in middle-out order of a normal software development. This also gives enough opportunities for students to catch up with their understanding of programming techniques.

The material is distributed over Software Design 1, a one semester second year course, and Software Design 2, a one semester third year course. Both are required for Computer Science at McMaster. The courses were taught repeatedly since 1999/2000 and had a peak enrollment of 190 students. Many students perceive these two courses as the core courses for their career in software development. The same material was presented in a condensed form in a graduate course in 2005/06 and in 2008/09.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE '09 Burnaby, BC, Canada
Copyright 2009 ACM 978-1-60558-415-7 ...$5.00.

Program Annotations

$\{x \geq 0\}$
$z, u := 0, x$;
$\{$**invariant**
 $(z + u \times y = x \times y) \wedge$
 $(u \geq 0)\}$
while $u > 0$ **do**
 $z, u := z + y, u - 1$
$\{z = x \times y\}$

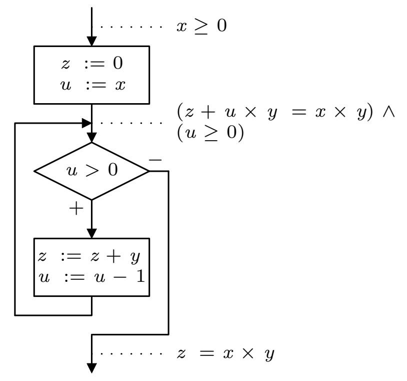


Figure 1: Excerpt from *Elements of Programming*

Statements with Partial Expressions

Assignment. Assume a: **array** N **of** T.

$$wp(x := E, P) = \Delta E \wedge P[x \backslash E]$$
$$wp(a(E) := F, P) = \Delta E \wedge \Delta F \wedge (0 \leq E < N) \wedge P[a \backslash (a; E : F)]$$

Conditional.

$$wp(\textbf{if}\, B \,\textbf{then}\, S, P) = \Delta B \wedge ((B \wedge wp(S, P)) \vee (\neg B \wedge P))$$
$$wp(\textbf{if}\, B \,\textbf{then}\, S \,\textbf{else}\, T, P) = \Delta B \wedge ((B \wedge wp(S, P)) \vee (\neg B \wedge wp(T, P)))$$

Repetition. If

$B \wedge P$	$\Rightarrow$	$wp(S, P)$	(P is invariant of S)
$B \wedge P \wedge (T = v)$	$\Rightarrow$	$wp(S, T < v)$	(S decreases T)
$B \wedge P$	$\Rightarrow$	$T > 0$	($T < 0$ causes termination)
P	$\Rightarrow$	ΔB	(B is always defined)

then

$$P \Rightarrow wp(\textbf{while}\, B \,\textbf{do}\, S, P \wedge \neg B)$$

Figure 2: Excerpt from *Elements of Programming*

The next section elaborates on the topics covered in both courses, in the order of presentation. We conclude with an evaluation of the courses and a discussion.

2. MIDDLE-OUT SEQUENCING OF TOPICS

Lecture 1: Elements of Programming

The first course starts with an introduction to basic control structures. The notion of (total) correctness is defined through the triple $\{P\}\, S\, \{Q\}$. For the purpose of proving, correctness is restated in terms of weakest preconditions as $P \Rightarrow wp(S, Q)$. Using the rules of wp, intermediate annotations can be derived automatically and annotations can be restricted to preconditions, postconditions, and loop invariants. Equational logic is introduced by appealing to the analogy with calculus. A formal definition of the syntax of programs and annotations and of typing is included; trying to skip this did lead to syntactically incorrect mixtures of programs and annotations. Flowcharts, like in Fig. 1, are used for explaining annotations. Trying to use them in exercises did lead to spaghetti charts; they are still retained as they are intuitive and as they are later used for explaining exceptions. Limitations of machine arithmetic and partial expressions are discussed and formalized as in Fig. 2. Further control structures (case, repeat, and for statements) and recursion complete the picture. Basic sorting and searching techniques, as they are taught in introductory and high school courses, illustrate the use of annotations. Quicksort and Boyer-Moore search are used to demonstrate the need for correctness arguments of more complex algorithms. Stepwise refinement is illustrated with the example of printing images [1]. Proper programming style (indentation, comments, naming) is discussed and from that point on enforced. The assignments are with paper-and-pencil only and force students to argue about programs without running and testing them. While this comes as a surprise to students, it puts students with and without programming experience on the same level. Verification is continued to be practiced throughout both software design courses: our observation is that one semester is not sufficient for students to feel comfortable with writing annotations.

Lecture 2: Program Modularization

The goals and principles of modularization are discussed and a notation for modules is introduced, see Fig. 3. The principles are supported by a discussion of the consequences of local module invariants (cohesion) and global module invariants (coupling). The KWIC example is used for illustrating the difference in qualities that arise from different modularizations [5]. The key point is that students learn that there is not the ideal modularization, but only one for—explicitly stated—anticipated changes. The need for robustness of modules is discussed and defined formally. Exercises continue to practice formally reasoning about programs, but also show how to map modules to the constructs found in programming languages, in particular how encapsulation is enforced in common languages.

Lecture 3: Abstract Programs

Four means of abstraction are presented: (1) multiple assignments, as a way of abstacting from an arbitrary ordering of individual assignments, (2) guarded commands, as a way of expressing bounded nondeterminism, (3) specification statements, as a way of expressing unbounded nondeterminism, and (4) abstract data types, for data abstraction. These are first illustrated with simple, abstract algorithms and then with examples as typically found in books on algorithms and data structures. Exercises practice the use of abstract data types for modelling information systems, as a preparation for object-oriented modelling. The techniques are then used for the specification of modules: the previous notation of a syntactic interface of a module is augmented by a complete abstract specification. Abstraction continues to be repeated throughout both courses. While we tried to start the first course with abstract programs instead of concrete programs, our experience was that students didn't have an understanding of what they are abstracting from; we find that it takes students significant time to appreciate for example nondeterminism.

Lecture 4: Testing

The role and need for testing are discussed. Testing is presented as complementing verification. Testing the internal consistency of modules is illustrated with checking module invariants. Specification based testing is used for both black box and white box testing. The wp calculus is used for de-

Why Modularization

Modularzation serves three purposes:

Comprehensibility We cannot understand a sizeable program unless we split it into manageable parts.

Maintainability We cannot make changes to a sizeable program unless the changes are confined to some parts.

Development We cannot develop a sizeable program in a team unless each team member develops their own part.

These goals necessitate the division of a program into *modules* with *interfaces* and *implementations*:

- Modules can be *used* based on their interface, without the need of understanding their implementation.
- Modules can be *implemented* based on their interface, without the need of knowing their use.

This way the *clients* (*users*) and the implementation of a module can be designed separately and can evolve (more) independently.

Module Invariants

A *module invariant* characterizes the possible states of a module. It is a predicate that holds after the initialization and after any subsequent call to the module. As the module invariant is an essential design decision of a module, we document the invariant as an annotation:

```
module BoxOffice
  public const CAPACITY = 250
  var seats : integer
  {invariant: $0 \le seats \le CAPACITY$}
  public procedure bookSeat
    require $seats < CAPACITY$ then $seats := seats + 1$
  public procedure cancelSeat
    require $seats > 0$ then $seats := seats - 1$
begin $seats := 0$
end
```

Figure 3: Excerpts from *Modularization*

riving test cases to achieve various types of coverage, as in Fig. 5. Test strategies are discussed. The assignments practice writing both implementations of modules and test suites according to formal specifications; in one assignment both implementations and test suites are run against (faulty) ones from other students. From that point on students are convinced of the need for precise specifications.

Lecture 5: Exception Handling

Failures, the need for exception handling despite verification and testing, and ways of reacting to exceptions are discussed. The raise and the try-catch statements, as the dominant exception mechanism, are introduced textually and graphically, and then defined formally through *wp*. Verification of exception handing is practiced with small examples. The correct design of exception handlers in modular programs is discussed. Students should understand that, for example, an exception handler is supposed to re-establish the module invariant and establish an alternative postcondition. Programming exercises from now on have to include proper exception handling.

Lecture 6: Functional Specifications

A formal model of programs in terms of relations is given and connected to *wp*. The use of predicative specifications is discussed. Tabular specifications are used to discuss the concepts of completeness of consistency of specifications [6], see

Two Nondeterministic Programs

Determining the maximal value in an array:

```
var $i$ : integer
begin $m, i := a(0), 1$ ;
  do $i < n \rightarrow$
    if $a(i) \le m \rightarrow$ skip
    [] $a(i) \ge m \rightarrow m := a(i)$
    fi ;
    $i := i + 1$
  od
end
```

Determining a location of the maximal value in an array:

```
var $i$ : integer
begin $k, i := 0, 1$ ;
  do $i < n \rightarrow$
    if $a(i) \le a(k) \rightarrow$ skip
    [] $a(i) \ge a(k) \rightarrow k := i$
    fi
  do ;
  $i := i + 1$
end
```

Both programs are nondeterministic, but the outcome of the first is unique.

Algorithmic Abstraction vs. Data Abstraction

- Multiple assignments, guarded commands, and specification statements provide *algorithmic abstraction*: they abstract from possible algorithms implementing them, but are expression in therms of the data structures of the program.
- *Data abstraction* abstract from possible data structures of the implementation by using abstract data types.

Example. Counting the number of distinct elements of array a : **array** N **of** T:

```
var $i$ : integer, $s$ : set of $T$
begin $i, s := 0, \{\}$ ;
  do $i < N \rightarrow s := s \cup \{a(i)\}$ ; $i := i + 1$ od ;
  $num := \#s$
end
```

Here we abstract from how elements of the set s are stored: they can be stored in an array, liked list, or hash table.

Figure 4: Excerpts from *Abstract Programs*

Fig. 7. Algorithmic refinement and data refinement are formalized with relations. Previous examples of module specifications and implementations are revisited and formally analyzed; the emphasis here is on understanding the concept of a refinement relation, as a preparation for class refinement.

Lecture 7: Object-Oriented Programs

Classes and inheritance are introduced textually and graphically. The object-oriented style is contrasted with the traditional style. A formal model of classes and inheritance is given that makes the additional complexity of object-oriented design explicit by translation into a model with variables and procedures only, see Fig. 8. Class invariants and class refinement are studied in this model. Class refinement is then used to justify "good" and "bad" used of inheritance. Small exercises enforce the understanding of the formalism whereas programming assignments practice class design without formal proofs.

Lecture 8: Object-Oriented Modeling

Object-oriented models are presented as an alternative, graphical way of specifying data structures. For this, class dia-

Path Coverage

Alternatively, we can derive a set of test cases such that all *paths* are covered. This includes coverage of all statements. In the example below, the paths are A-C, A-D, B-C, B-D:

```
if a(0) < a(1) then
  l := 1              – A
else
  l := 0 ;            – B
if a(l) < a(2) then
  l := 2              – C
else
  skip                – D
```

Test cases are determined by calculating the weakest precondition that excludes all alternative paths. For example, for testing the path B-C we calculate:

$$\begin{aligned}
&\neg(a(0) < a(1)) \ \wedge \\
&wp\ (\textbf{if } a(0) < a(1) \textbf{ then } l := 1 \textbf{ else } l := 0,\ a(l) < a(2)) \\
=\ &\text{"wp of } \textbf{if}\text{, logic"} \\
&(a(0) \geq a(1)) \ \wedge wp(l := 0,\ a(l) < a(2)) \\
=\ &\text{"wp of } := \text{"} \\
&(a(0) \geq a(1)) \ \wedge (a(0) < a(2))
\end{aligned}$$

From there, we pick arbitrary values that satisfy the precondition, for example $a(0) = 5,\ a(1) = 4,\ a(2) = 7$.

Testing Modules

Since modules may have private variables, we can neither set nor inspect their values directly.

- In order to set their values to a desired state, we have to call a sequence of *modifiers* (modifying public procedures).
- In order to inspect their values, we have to call one or more *observers* (observing public procedures).

With testing in mind, we should include sufficiently many modifies and observers in the interface. This leads to the requirement of designing modules for *testability*.

Figure 5: Excerpts from *Testing*

grams are extended with various forms of associations that are given a textual definition. The use of object-oriented models for a partial (abstract) view and for guiding the modularization according to data is practiced (repeating that lesson from *Modularization*). The transition from an object-oriented model to an object-oriented implementation is explained as a refinement step, in accordance with the earlier formalization of refinement, and practiced informally.

Lecture 9: Requirements Analysis

The need for formulating requirements in the "user's world" and the need for distinguishing user requirements from program specifications is discussed. The step of delineating the context of a software system is discussed with use cases and use case diagrams. The notion of the interaction of a software system with its environment is motivated with sequence diagrams. Proving the consistency of a specification with respect to sequence diagrams is discussed using *wp*, see Fig. 10. The derivation of test cases from sequence diagrams is discussed, thus connecting the topic of requirements analysis with specification, verification, and testing.

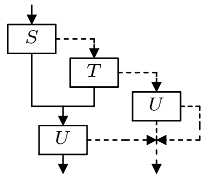
Lecture 10: Object-Oriented Design

Object-oriented techniques (like forwarding and delegation), design patterns, frameworks, subsystems, and components are discussed. Class invariants and class refinement are used in explaining class structures, but without formal proofs. Ten of the common 24 design patterns are selected. Assignments use the java.util framework for illustrating the use of design patterns when extending frameworks.

Finally Clause

A try-catch block may have a finally clause that is executed whether an exception occurred or not. Suppose S, T, U are statements:

try S **catch** T **finally** U



Either the catch or the finally clause is optional:

try S **catch** T = **try** S **catch** T **finally skip**
try S **finally** T = **try** S **catch raise finally** T

Weakest Exceptional Precondition

Recall that $wp(S, Q)$ is the weakest precondition such that statement S terminates and condition Q holds finally. We define:

$wp(S, Q, R)$ = weakest precondition such that S terminates and
– on normal termination Q holds finally
– on exceptional termination R holds finally

A statement that never raises an exception can be equivalently defined through weakest precondition or weakest exceptional precondition:

$$wp(S,\ Q) \ = wp(S,\ Q,\ \textbf{false})$$

The weakest precondition $wp(S, Q, R)$ is monotonic in both Q and R:

if $Q \Rightarrow Q'$ and $R \Rightarrow R'$ then $wp(S, Q, R) \Rightarrow wp(S, Q', R')$

Figure 6: Excerpts from *Exception Handling*

Lecture 11: Reactive Programs

The characteristics of reactive programs are contrasted with those of transformational programs. Statecharts are introduced as a dedicated formalism for reactive programs. The event-based approach to reactive systems is contrasted with the state-based approach. The elements of statecharts are first illustrated and then defined in terms of guarded commands, following [9, 10], see Fig. 11. Assignments practice the use of statecharts with *iState*, a statechart compiler that follows that definition and allows statecharts to be animated. In the fall of 2006, we experimented with adding the invariants to statecharts, with automatic verification in iState following [4].

Lecture 12: Software Development Process

Different software development processes are mentioned, without going into detail (for time).

Interlude: Software Tools

Additionally, the topic of *Configuration Management* was included at the beginning of Software Design 2 and subversion was used from that point on for all assignment submissions. The assignments used Pascal, Java, jUnit (for testing), and iState (for statecharts); introduction to these was provided in optional tutorials.

3. EVALUATION

Except for the topics of Configuration Management and Software Development Process, all other topics used a coherent notation and coherent mathematical basis. We did not

Completeness and Consistency with Tabular Specifications

When specifying complex systems, two fundamental issues arise:

Completeness: Does the specification cover all possible cases, or did we forget some cases?

Consistency: Are there contradictions in our specification, or is is consistent?

Both issues are addressed by *tabular specifications*: besides making it easier to check specifications for completeness and consistency, they make large specifications more readable and more appealing by the two-dimensional notation.

Properties of Tabular Specifications

Let a table with header $H = (p_1, \dots, p_n)$ be given.

	p_1	$\dots$	p_n
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- H is *disjoint* if $\neg(p_i \wedge p_j)$ for all $i \neq j$.
- H *covers* p if $p_1 \wedge \dots \wedge p_n = p$.
- H *partitions* p if H is disjoing and H covers p.

Figure 7: Excerpts from *Functional Specifications*

observe that students are in any sense math-phobic: they are sceptical towards the use of logic in software design as much as they are sceptical toward design patterns and configuration management systems; they haven't seen the need for any of these. In a series of assignments, the use of each concept of the course is practiced. At the end of the second course, students take the use of logic for granted. The topic that caused the most difficulties was object-oriented modelling: while the concepts are mathematically easy to explain, acquiring proficiency in practice would have taken more time than was available. The topic was dropped in later editions of Software Design 2.

Requiring students to take a course in logic and discrete math before Software Design 1 had only a moderate effect on their ability to use logic and abstract data types for the description of problems. Our explanation is that logic and discrete math courses traditionally teach a body of knowledge, and do not practice the use for description and do not practice proving. Additionally, the difference in notation, as for implication and for quantification, prevents students from seeing the connection even if there is an obvious one (one can easily find a dozen different notations for quantification; we doubt that calculus would have the same influence if that many different notations for addition or integration would be used). Operators that are useful in software design, like relational overwrite, are not taught in discrete math courses; usually (untyped) first-order logic is taught, rarely equational proofs. A good portion of Software Design 1 is spent—or rather wasted—with introducing notation for typed logic, data types, and equational proofs. One would wish that the field would have matured by now to standardize them.

We did not observe that students with previous experience in programming did better than those without. To the contrary, students with previous experience are being more dismissive of topics like modelling, design patterns, configuration management, except if they have already been exposed to those. In questions on verification, these students gener-

Definition of Inheritance

Methods correspond to procedures that take an additional parameter for **this**; the body must not assign to **this**. Self-calls are resolved to the methods of the class itself:

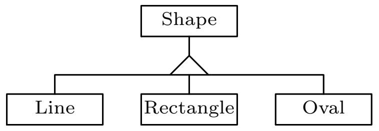
```
                        var C : set of Object
class C                 var a : map Object to A
  var a : A             invariant
  method s                nil ∈ C ∧ dom a = C − {nil}
    S               =   procedure C.s(this : C)
  method t                S[s, t \ C.s, C.t]
    T                   procedure C.t(this : C)
end                       T[s, t \ C.s, C.t]

                        var D: set of Object
                        var b: map Object to B
class D inherit C       invariant D ⊆ Y ∧
  var b : B               nil ∈ C ∧ dom a = C − {nil}
  override method t     procedure C.s(this : C)
    T′              =     S[s, t, u \ D.s, D.t, D.u]
  method u              procedure C.t(this : C)
    U                     T′[super.s, super.t \ C.s, C.t]
end                           [s, t, u \ D.s, D.t, D.u]
                        procedure C.u(this : C)
                          U[s, t, u \ D.s, D.t, D.u]
```

Forms of Inheritance

Object-oriented languages own much of their popularity due to the way in which inheritance can be used (or misused). Good uses of inheritance are:

- Inheritance for modelling
- Inheritance for specification
- Inheritance for code sharing

Inheritance for modelling is used to let the structure of the program reflect the structure of the problem: the program becomes easier to understand and to maintain. It applies when the one class is a *generalization*, or conversely a *specialization* of another class.



The essence is that each subclass must be a *refinement* of its superclass.

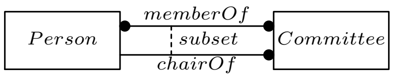
Figure 8: Excerpts from *Object-Oriented Programs*

ally performed poorer. One reason is that these students find it more difficult to think abstractly about programs. The other reason is purely notational: with = meaning assignment in some programming languages but equality in mathematics, they confuse these two notions and write meaningless correctness statements like $\{true\}\ x = x+1\ \{x = x+1\}$. They have initial difficulties distinguishing between a property of a state and a statement leading to that property. At the end of the second course the difference between students with and without previous experience disappears. Still, one would wish that programming languages would follow the tradition of mathematics in the use of =.

Students are required to complete a two-semester design project in their fourth year. Software Design 2 was consistently ranked as the most useful course in a questionnaire at the end of the project and Software Design 1 as the third most useful course. While that may sound encouraging, the projects rarely show a sufficiently systematic application of the techniques. That may be partly due to the explorative

Constraints

Constraints allow to express additional restrictions which are not captured by the diagrams on their own. Constraints can be attached to classes. Such constraints become an additional part of the invariant [example omitted]. Constraints can also be written next to associations. Dashed lines are used to connect the involved associations:

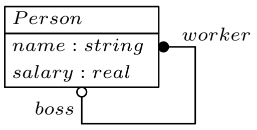


var *Person* : **set of** *Object*
  Committee : **set of** *Object*
  memberOf : **rel** *Object* **to** *Object*
  chairOf : **rel** *Object* **to** *Object*
invariant
  dom $memberOf \subseteq Person \ \wedge$
  ran $memberOf \subseteq Committee \ \wedge$
  ran $chairOf = Committee \ \wedge$
  $injective(chairOf) \ \wedge$
  $chairOf \subseteq memberOf$

Refining Class Diagrams

Class diagrams are a way of graphically expressing object-oriented system models. Some concepts, like classes with attributes and methods, can be readily implemented in programming languages, while others, like associations and qualification, cannot. In such cases, we have to refine our model, perhaps in several steps, until we arrive at a model that is sufficiently close to our programming language. The refined models can be expressed graphically as well.

Refining Associations by Pointers. Consider:



Suppose we decide to implement the association by adding an attribute *boss* to each person. That attribute could be **nil** or a pointer to the boss, reflecting the zero-or-one multiplicity.

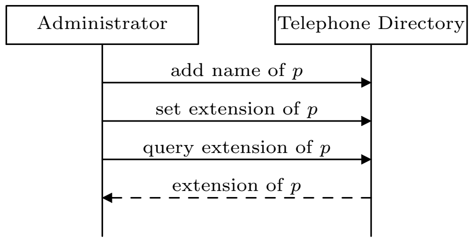
Figure 9: Excerpts from *Object-Oriented Modeling*

Sequence Diagrams

Scenarios can be described by *sequence diagrams* (*message sequence charts*) showing vertically an interaction sequence among actors and the system over time:

- Solid horizontal arrows indicate a message (interaction).
- Dashed horizontal arrows indicate a response to a previous message.

For example, for part of the Set Extension use case followed by the Query Extension use case:



Checking Specifications Against Scenarios

Consider the scenario:

> Setting the extension of a specific person and querying the extension of that person will return the same extension again.

To check if our specification allows this scenario, we analyze the sequence of calls

$S = setExtension(p, e1) \ ; \ queryExtension(p, e2, found)$

by determining its weakest precondition with respect to postcondition $found \ \wedge (e1 = e2)$:

$wp(S, found \ \wedge (e1 = e2))$

In general, the goal is to check:

$inv \ \wedge pre \Rightarrow wp(scenario, post)$

Note that if every procedure called by the scenario preserved *inv*, we also have:

$inv \Rightarrow wlp(scenario, post)$
$inv \ \wedge pre \Rightarrow wp(scenario, post \ \wedge inv)$

Figure 10: Excerpts from *Requirements Analysis*

nature of these projects. However, the author believes that this is mainly due to these concepts not being repeated and practiced elsewhere. In courses on databases, operating systems, compilers, user interfaces, networks, real-time and algorithms terms like invariants and robustness are not used, giving the impression that these notions are not universally relevant. To give evidence to this claim, we refer to the analysis of the five most popular algorithm textbooks in [11]: four books, with 550 to 770 pages, devote zero pages on correctness and one book with 790 pages devotes eight on correctness. One would wish that textbooks and instructors would acknowledge the usefulness of these notions more widely.

Over the years, in the course evaluations 30%–65% of the students report that 81%–100% of the course material seems valuable and 35%–50% report that 61%–80% seems valuable. The numbers were on the higher end in later years and for Software Design 2 (not all students continue with Software Design 2). The use of independent critical judgement was rated high, particularly in later years. The overall delivery of the course received mixed evaluations, because the material was not fully developed in earlier years, the material was not motivated well in earlier years, most teaching assistants were of little help to the students, and because students felt overloaded. Except in the first year, there were no complaints that the contents is overly mathematical.

4. DISCUSSION

We believe that we have successfully used mathematics as the unifying force for elements of software design into a two-semester courses in software design. The material is covered in 710 pages of lectures notes by the author plus a couple of original articles and book chapters (a course pack with the material is printed for students on demand). The mixture of mathematical and less mathematical topics gives students confidence that the use of mathematics is justified. We could not have done this with a single one-semester course.

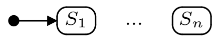
If teaching the mathematics of software design is to be useful, it has to be taught as early as possible, before students acquire "bad habits," a point that has been repeatedly made, e.g. [7]; we wish we could have started even earlier in the curriculum. We have deliberately not used a specific formal tool; we find those more appropriate for upper level courses. Gordon [3] also offers a two-semester course, also

Implementing Statecharts

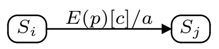
We now present a translation scheme for all elements of statecharts, except timing. A statechart is implemented by a module:

- States are represented by variables;
- Events are represented by procedures.

This way "generating an event" means "calling a procedure". (We note that a completely different implementation is possible in which events are treated as data.)



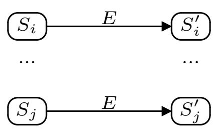
```
var $s$ : ($S_1$, ..., $S_n$) := $S_1$
```



```
procedure $E(p)$
  if $s = S_i \wedge c$ then
    begin $a$ ; $s := S_j$ end
```

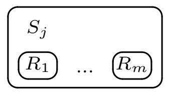
For brevity, we leave out parameters, conditions, and actions in subsequent rules. They have to be added according to above scheme.

Suppose there are several transitions on event E. Here $S_i, \ldots, S_j$ are not necessarily distinct states, with an overlap leading to nondeterminism:



```
procedure $E$
  if $s = S_i \rightarrow s := E_i'$
  [] ...
  [] $s = S_j \rightarrow s := S_j'$
  [] $s \neq S_i \wedge ... \wedge s \neq S_j \rightarrow$ skip
  fi
```

Hierarchy.



```
var $s$ : ($S_1$, ..., $S_n$)
var $r$ : ($R_1$, ..., $R_m$)
```

Figure 11: Excerpts from *Reactive Program*

using higher order logic as a unifying framework, but covers the logical aspects in more detail, and includes hardware verification. We have not tried to use any "light" method that makes formal techniques "invisible"; in our experience students appreciate being taught the theory in an isolated, minimal way, before seeing it applied with constraints.

Compared to the inverted curriculum, or outside-in order by Pedroni and Meyer [8], we do not teach class design before control structures, but we do teach programming before requirements analysis. Our way of introducing formal techniques is less gentle than theirs, but our reason is the same as for their outside-in order: to put all students on the same level and keep them motivated. We have succeeded with the first one, even if it comes by shocking the students in the first classes with mathematics, for which they were not prepared; we were less successful in keeping them motivated during Software Design 1.

While we believe that the courses influenced the way how students think about programs, the main obstacle for having a profound influence on their practice of programming is that concepts are not being repeated and practiced in other courses. We agree with Dijkstra's observation on computing science [2]:

> ... providing symbolic calculation as an alternative to human reasoning ... is sometimes met with opposition from all sorts of directions: ... 6. the educational business that feels that if it has to teach formal mathematics to CS students, it may as well close its schools.

If anything, with low enrollment numbers the pressure to eliminate mathematics has increased.

5. ACKNOWLEDGEMENTS

The author would like to thank David Parnas, Michael Soltys, and the two reviewers for their careful reading and thoughtful suggestions.

6. REFERENCES

[1] E. W. Dijkstra. Notes on structured programming. In O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structured Programming*. Academic Press, 1972.

[2] E. W. Dijkstra. On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12):1398–1404, 1989.

[3] M. Gordon. *Specification and Verification, Parts I and II*. http://www.cl.cam.ac.uk/~mjcg/, 2006.

[4] D. T. M. Le, E. Sekerinski, and S. West. Statechart verification with iState. In M. Chechik, editor, *Formal Methods 2006—Posters and Research Tools*, Hamilton, Ontario, 2006. http://fm06.mcmaster.ca/istate.pdf.

[5] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[6] D. L. Parnas. Tabular representation of relations. CRL Report 260, McMaster University, October 1992.

[7] D. L. Parnas. "Formal Methods" technology transfer will fail. *Journal of Systems and Software*, 40(3):195–198, 1998.

[8] M. Pedroni and B. Meyer. The inverted curriculum in practice. In *SIGCSE Technical Symposium on Computer Science Education*, Houston, Texas, USA, 2006. ACM Press.

[9] E. Sekerinski and R. Zurob. iState: A statechart translator. In M. Gogolla and C. Kobryn, editors, *UML 2001 – The Unified Modeling Language, 4th International Conference*, Lecture Notes in Computer Science 2185, Toronto, Canada, 2001. Springer-Verlag.

[10] E. Sekerinski and R. Zurob. Translating statecharts to B. In M. Butler, L. Petre, and K. Sere, editors, *Third International Conference on Integrated Formal Methods*, Lecture Notes in Computer Science 2335, Turku, Finland, 2002. Springer-Verlag.

[11] A. B. Tucker, C. F. Kelemen, and K. B. Bruce. Our curriculum has become math-phobic! In *SIGCSE Technical Symposium on Computer Science Education*, Charlotte, North Carolina, USA, 2001. ACM Press.

Training $\neq$ Education: Putting Secure Software Engineering Back in the Classroom

Michael L. Stamat
Air Force Institute of Technology
2950 Hobson Way, Bldg 641
Wright Patterson, Ohio 45433
michael.stamat@a t.edu

Jeffrey W. Humphries
Air Force Institute of Technology
2950 Hobson Way, Bldg 640
Wright Patterson, Ohio 45433
jeffrey.humphries@a t.edu

ABSTRACT

In the world of software engineering, security remains a critical issue. Companies lose billions each year because commercial vendors continue to produce exploitable applications. Over 8,000 vulnerabilities were cataloged by the Computer Emergency Response Team in 2006 alone. Despite this alarming statistic, companies still grip the same train-and-certify approach for cultivating security-minded programmers. However, exhibited by the prevalent vulnerabilities still appearing in cyberspace, a new ambitious plan for robust software development must be implemented. This paper addresses the inadequacy of training and encourages the academic community to adopt modern software security essentials into the undergraduate computer science curriculum. This paper also proposes a unique software engineering course targeted to senior-level computer science students that underlines design methods, tools, and standards applicable to writing secure code.

Categories and Subject Descriptors

K.3 [**Computers and Education**]: Computer and Information Science Education—*undergraduate curriculum*; D.2.8 [**Software Engineering**]: Design Tools and Techniques—*secure programming, static analysis, testing methodologies*

General Terms

Design, Security

Keywords

software engineering, software security, software development lifecycle, design for security, security education, computer science

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WCCCE'09, May 1–2, 2009, Burnaby, BC, Canada.
Copyright 2009 ACM 978-1-60558-415-7...$5.00.

1. INTRODUCTION

When the original designers of the Internet began development they did not have security in mind. In fact, when it was first built it was just a private network available only to the government and academia. The Internet was simply a conduit to relay information among disparate systems. It took several years before the architects realized its potential for commercial application. Today, this invaluable tool now impedes innovation, costs firms billions of dollars annually, and threatens national security.

Before ecommerce and email, the Internet's developers were only concerned about one thing—functionality. They wanted a robust architecture that was easy to use and flexible for expansion. The same can be said about the majority of today's programming languages. Major language paradigms, such as C, were created to enable humans to specify the behavior of a computer. Its syntax and semantics define instructions to express algorithms and solve problems.

At first, computer programming languages were intended to facilitate mathematical computation, information distribution, and other facets of software engineering still employed by industry today. Like the Internet, programming languages quickly evolved and, also like the Internet, its creators did not have security in mind. The rapid commercialization of software suddenly moved conventional programming from simple calculations to advanced applications. The private sector demanded functional and flexible programming mechanisms to streamline its software development and maximize profit. As a result, many programming languages lacked the necessary security to ensure reliable and robust code.

The current state of software security is unchanged. Although attempts have been made to integrate security "features" into popular languages, inherent relaxed coding standards only promote the widely adopted poor programming practices of today's software developers. These programming practices invite the exploitation of code defects—defects that if unchecked could lead to catastrophic results. The Blaster worm, for instance, infected nearly 330,000 systems worldwide and was successful because of a coding defect only two lines long [9].

Developing secure software is a difficult undertaking and can stress a project's design, usability, and budget. As expected, modern businesses often ignore these security practices in favor of a less demanding approach and continue to perpetuate this dilemma. Since 1998 the Computer Emergency Response Team (CERT) has seen a sharp rise in the number of reported security-related software vulnerabilities [5]. Figure 1 illustrates this increase over a ten year period.

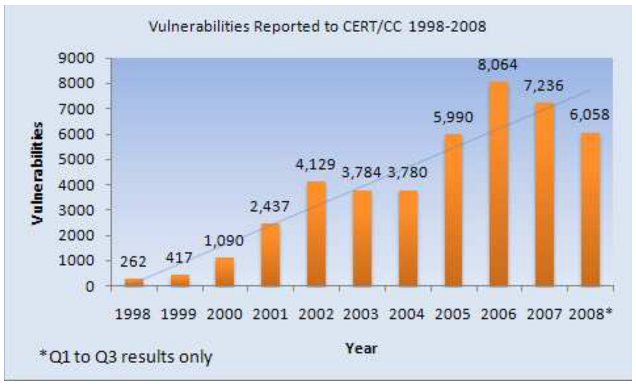


Figure 1: Vulnerabilities Report

These statistics confirm the need for a stronger promotion of secure code and software practices that include security-infused code design, risk analysis, and testing. However, industry still fails to implement this methodology. Instead, organizations favor the penetrate-and-patch approach primarily for its low initial cost in both research and training [6]. This in turn produces software that is not secure by design, but refined by trial and error.

A growing trend among businesses is to send their inexperienced programmers to security ”boot camps”. These boot camps provide an accelerated instruction medium for software developers to receive security training and certification. Although this is a commendable response it only sidesteps the issue. Companies have lowered their expectations of today’s college graduates and many now require all new hires to go through this additional training. This training only delivers a greater financial burden, both for the company and the consumer. If the development community wants to combat this problem then it’s going to have to start from the beginning, in colleges and universities.

The undergraduate computer science curriculum includes a variety of critical skills needed to graduate creative software developers. Most computer science courses involve cross-platform programming, object-oriented basics, and the fundamentals of the software development life cycle. But where is the security? Where are universities cultivating security-minded software professionals? Demonstrated by the prevalent vulnerabilities still appearing in cyberspace, a new ambitious plan for software security education must be implemented.

Essentially, computer science students are learning too little, too late. By the time they have entered the workplace, only a small percentage of graduates are exposed to software security [7]. This paper proposes a security-infused software engineering course to replace the traditional software engineering course taught in the undergraduate computer science curriculum. Its content underlines the design methods, tools, and standards applicable to developing secure software. The methods described can serve as a resource for educators to address this growing issue.

1.1 Prior Work

Software security by and large has become a hot topic in the computer science community. With software vulnerabilities costing nearly $60 billion annually, industry has started requiring its developers to know secure code design [15]. However, entry-level programmers just don’t have the education to support this demand. An informal survey of the top fifteen computer science universities revealed only four to include a software security course designed specifically for undergraduates [14]. Those colleges that did offer software security education did so through a focused technical elective [11].

The common components found in these electives include labs, research papers, and programming assignments—similar to most other computer science courses. Since the focus is specifically on software security, students are also introduced to various testing tools and methodologies used in the field. Although this approach does deliver a viable response to industry’s challenge, it fails to address software security at its core during development. Emphasis is on the technology not on the discipline of secure software engineering.

1.2 Objectives

Objectives benefit educators because they help design their instructional content and provide a tool by which success can be measured. The objectives below are for instructors to organize their efforts and clearly define the course’s characteristics and purpose. These should be stated at the beginning and reviewed periodically during the semester to ensure students are achieving the anticipated results. The objectives for this course are the following:

- Comprehend fundamentals of software security
- Identify common software vulnerabilities
- Learn defensive programming methods
- Utilize testing tools and techniques
- Implement the Secure Software Development Lifecycle
- Build analytical skills through peer review
- Produce effective communicators

2. COURSE CONCEPTS

2.1 Motivation

In the inaugural issue of IEEE Security & Privacy (January/February 2003), Matt Bishop said that ”security must be treated as a property to be engineered into every system component with the design, rather than viewing it as functionality to be added later [3].” For this approach to work, software security must become a multidisciplinary effort. Guided software security principles, coupled with case studies in design and analysis only scratch the surface of secure software development. Yet, students and industry alike expect this knowledge to be gained later through training. For reasons mentioned earlier, this is why security will continue to be a growing problem.

The course outlined here aims to close the gap between the ”bolt-on” techniques taught in computer security courses and the software development process taught in senior-level software engineering. By combining the use of software security best practices into the entire development life cycle, this course can effectively prepare students before they enter the professional work environment [2]. Considering over 60% of computer science graduates are recruited into software engineering, its influence would be significant [17]. For this reason, the course described in this paper engages the issue at its core. From its challenging and security integrated semester-long project to its real-world exercises, this course encompasses a framework that should be employed by all those responsible for educating the next generation of software developers.

2.2 Methodology

The methodology behind this course is for students to gain more than a high-level view of software security. Students will focus not just on tools and techniques, but also on secure methodologies, design principles, and risk analysis. This comprehensive approach fosters critical thinking skills while engaging students with relevant, exciting topics. Moreover, the professor should encourage an interactive learning environment with involved discussions, building both technical and analytical skills. These skills combined with a proficiency in team work through peer review effectively prepare students for the professional development environment.

Content of most computer science courses can vary significantly as new problems emerge and new concepts are put into practice. This course is no exception. Contemporary principles are what drive the software industry, and avoiding the adoption of these principles only suppresses student creativity. Therefore instead of hard coding its content, this software engineering course will integrate concepts from the widely accepted development life cycle with applicable software security tools and techniques used in academia, industry, and the hacker community. And by concentrating the students' efforts primarily on security at the design phase, instructors can maximize the investment of their time and money.

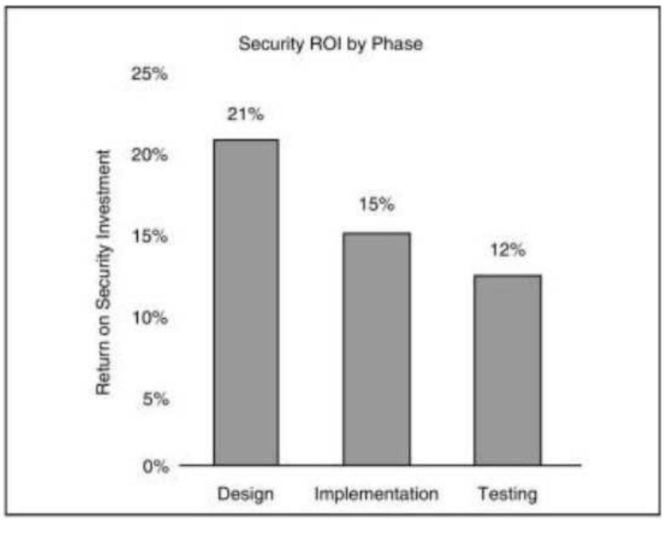


Figure 2: Return on Investment (ROI) measured by @stake after 23 engagements. [8]

The course methodology can be modeled into five characteristics:

2.2.1 *Comprehensive*

One of the primary motivations for this course format is maximum exposure to the realm of software security. Applicable labs, guest lectures, presentations, and a team project all contribute to a well-rounded syllabus. This in turn prevents graduates from starting in industry without the necessary tools to perform their job effectively.

2.2.2 *Innovative*

Detailed course content drives student creativity and encourages research outside of the classroom. One of the core competencies of today's leading software developers is their ability to innovate easy solutions to hard problems. Without this skill, software engineers are left unprepared to face many financial and technological challenges in the real world.

2.2.3 *Relevant*

As with any computer science course, its content must be renewed with each semester. New ideas and methods surrounding software engineering are introduced every day. Students must stay current of the latest in development practices to successfully integrate into the trade following graduation.

2.2.4 *Interactive*

The course also requires a lively and active classroom environment, vibrant with student discussion and feedback. Clear communication between students and the professor is paramount as many of the tasks assigned in this course are quite challenging.

3. CHALLENGES

Like any course, its implementation will not be without its challenges. Instructors and students will both have to examine new territory and work together to achieve the aforementioned objectives.

3.1 Role of Instructor

Software security encompasses a wide range of principles, tools, and practices. This puts an increased responsibility on the professor to have widespread knowledge of the course material. As such, the professor must also possess several years of experience in industry to properly address applicable ”real-world” issues. Experience can include being a member of a software development team or a security consultant.

3.2 Role of Student

The success or failure of the program largely depends on the student. During the course there will be opportunities for discussions and feedback. As such it is the student's responsibility to participate and demonstrate comprehension to their peers. Students should also be encouraged to work in groups and collaborate during the exercises although all submissions must be their own. This is not so much a rule, but a benefit for the instructor. Team work ensures students facilitate a positive mentality to help their classmates and exposes them to a professional atmosphere. Additionally students must synthesize all they've learned by drawing upon the concepts taught in class and creating innovative solutions to real-world problems.

3.3 Assumptions

This course is designed for senior level undergraduate computer science or computer engineering students with a strong proficiency in systems programming and software engineering fundamentals. Although the class seeks to educate students from all backgrounds, the assignments are technical in depth and breadth. Therefore, the professor should expect students to possess extensive technical competence in at least one engineering discipline. All students must also have the ability to program in an object oriented language at an intermediate or advanced (preferred) level.

The course will incorporate a large knowledge base with both technical and non-technical aspects. It should be made clear in the syllabus that this course requires students to communicate effectively in front of a class, perform under pressure, and be willing to collaborate as part of a project team. These three elements emphasize professionalism and diversity, characteristics that will stimulate creativity and talent.

4. COURSE COMPONENTS

4.1 Lecture

The main purpose of lecture is to outline the steps required to complete the semester long project and investigate modern concepts of software engineering. Emphasis during lecture is to facilitate discussion among students and the professor should attempt to include relevant articles from IEEE and ACM publications. The goal is to encourage students to further investigate the extensive subject matter presented in class.

To accompany the lecture, students will be reading from Craig Larman's *Applying UML and Patterns* or an equivalent software design textbook. Readings will parallel the teaching material and provide a useful reference for the project. Mr. Larman includes educator resources on his website [10]. At the discretion of the professor, lectures may also include guest speakers from academia or industry. These distinguished visitors can expose students to advanced research efforts in software security and some of the practical tools implemented in real organizations.

4.2 Homework

Students should also be assigned weekly readings from *The Mythical Man-Month* by Fred Brooks, a collection of essays on the management of computer programming projects [4]. This book contains reflections on the many obstacles related to software engineering and highlights some of the design principles that might relieve development headaches down the road.

Students can benefit from this book's insight while working on their team project. Instructors should assign two chapters a week. Each reading assignment should require a one-page reaction paper from each student overviewing its content and providing a response. The paper should be due the following class and the beginning of lecture allocated for discussion and feedback.

Table 1: Example Course Schedule

Week	Lecture Topics	Readings (UML)
1	Introduction, review course contents	Ch. 1-3
2	Requirements Analysis, Use Cases, Unified Process	Ch. 4-7
3	Software Project Management, Collaboration Diagrams	Ch. 8-11
4	Software Quality Assurance, Software Testing	12-16
5	Guest lecture, Software Architecture	Ch. 17-20
6	Maintenance, Coupling & Cohesion	Ch. 21-24
7	Implementation, Exam Review	-
8	MIDTERM EXAM	Ch. 25-29
9	Review Exam, System Architecture, Design Patterns	Ch. 30 - 33
10	Guest lecture, Deployment Diagrams, State Machines	Ch. 34 - 37
11	Software Architecture, OO Design Principles	Ch. 38 - 40
12	Guest lecture, Formal Methods, Refactoring	-
13	Group Presentations	-
14	Final Exam	-

4.3 Exercises

The exercises are designed to both parallel stages of the semester-long project and apply critical thinking skills to software security. It is this component of the course that students can receive hands-on experience with skilled guidance from the instructor. The exercises also provide students with the software security tools and techniques needed to effectively complete the first stages of their team project.

Each exercise should be completed weekly during a two-hour lab time. The first hour should be a presentation from the professor, identifying the particular security topic and providing examples of its associated vulnerabilities. Following the lecture, students will be assigned a related exercise with a lab report due at the next class meeting. Each student must include the following in their report:

- Problem statement
- Background
- Tools & Techniques
- Risk Analysis
- Problem Solution
- Assumptions
- References

The instructor is expected to remain in the lab for the second hour to answer any questions students may have about either the project or the exercise. Each exercise is based on one of the *Seven Pernicious Kingdoms* as described by Gary McGraw [16]. These "kingdoms" outline the most common coding mistakes that affect the security of modern software products. Students are free to collaborate, but are encouraged to work independently.

The following are example exercises and each requires a moderate level of configuration. It is suggested the professor implement each exercise on a series of virtual machines. This provides increased flexibility with backups and installation. NOTE: Additional material for the labs can be found in the 19 Deadly Sins of Software Security [9].

Exercise 1 - Input Validation) Software developers put a lot of trust in their inputs. Test the students' ability to spot code "sins" by setting up a system equipped with a service that supports remote logins. Protect the system by configuring the service to require username and password credentials for proper authentication. Modify the service's source code to instigate a vulnerability that would allow an attacker to exploit the remote authentication and gain access. Possible methods include buffer overflow, command injection, or race conditions. Implementation is at the professor's discretion. Students will be provided with the vulnerable source code and must gain access to the system using a computer attached to the same network as the target machine. Once they've circumvented the remote authentication, students must fix the given source code to prevent exploitation. The report must include the corrected code and the result.

Exercise 2 - API abuse) Functions that cannot be used safely should never be used. This exercise features a target machine hosting a modified FTP server. The server's source code includes the *chroot()* function to change the perception of the root directory of the file system. Implement the FTP GET function as to improperly invoke *chroot()*. Students must gain access to the server and retrieve the system's password file. Since the users logged into the remote system are in a *chroot()* jail they will need to discover the vulnerability in the API call and perform a directory traversal attack. After they've retrieved the password file, students must locate the FTP's vulnerable source code, correct the error, and recompile. The report must include contents of the password file and the corrected source code.

Exercise 3 - Security Features) Security through obscurity can be fool-proof, but never hacker-proof. Students must obtain administrator privileges to a database running on a target system. The instructor will provide SSH credentials (username and password) to gain a remote terminal with restricted privileges. The website should be readable, but not writeable and in the standard web server directory. Its source code will reveal the username and password used to connect to the internal database. Students must document how they obtained the password and why it's not a good security practice to store password information in a configuration file.

Exercise 4 - Time and State) Inside a computer, shared state interactions occur all the time and can be exploited. For this exercise, a respectable user is known to have recently logged into a secure website. The student has access to the same machine used by the user and must recreate their session. Students must figure out that the website stores session information in a temporary directory and does not properly delete this data at the end of a session. Each lab report must include how the temporary file was found and how the session was recreated.

Exercise 5 - Code Quality) Poor code quality results in unpredictable behavior. This exercise demonstrates the liability of improper error handling. The instructor provides a computer program that requests a filename to read from the user. If the filename specified does not exist, the program creates one. The program improperly checks to see if the file is readable. The students must determine this by observing its source code and will document how they exploited this capability. Implementation of the attack is left up to the creativity of the student.

Exercise 6 - Encapsulation) The encapsulation of code prevents the unauthorized use of methods private to an application. Students in this exercise will learn the importance of encapsulation by exploiting a program's public functions. Students should be given the source code and binary of a banking program. Upon execution the program allows users to create an account and deposit money. The student must find a way to abuse the system and hijack another user's account. The banking source code will reveal that it contains a public *get* function that retrieves an account given an index. The lab report must illustrate how the hack was performed, the fixed source code, and a demonstration of the secure result.

Exercise 7 - Environment) An unknown environment of an application can lead to dangerous assumptions and mistaken trust. The professor should build an improperly configured program that contacts a server periodically for updates and provide the client/server source code to students. They are to misuse this trust relationship between the client application and server and cause the client to crash. This can be done by poisoning the network's ARP tables and forcing the client to download an update from an attack server. The source code revealed that the buffer that read in the file expected proprietary file format input. Students must record their steps in the lab report, fix both the server and client code with a solution of their choosing, and demonstrate the result of the secure transaction.

Exercise 8 - Capstone (Tentative) As the final exercise for this course, students will demonstrate their knowledge of all the security vulnerabilities reviewed in lab. Content is at the discretion of the professor and should be tailored to the student's comprehension of the material.

4.4 Article Presentations

Since each exercise is held weekly, the last exercise will be on week eight or nine. For the lab sessions thereafter, each student will give an article presentation. These presentations will give students the opportunity to investigate a topic they are personally interested in and encourage the work of their peers. When students take an active role in the class it gives them a sense of responsibility for their education. While motivating the class, students will also improve their communication skills and prepare them for their final project presentation.

Presentations must be at least 15 minutes in length and related to secure software design and development. Articles should also be from a respectable source such as an IEEE or ACM journal. Instructors should require their students to submit their article topic for approval at least two weeks prior to the presentations. The focus of the presentation is its technical content. Students should strive to brief its specific details and educate the class much like the instructor. No two presentations should be similar in topic and students are free to reference any additional material as needed.

Professors should grade their students based on content, style, and skill. They should also ensure they've clearly outlined the problem statement, included any necessary background, and provided sufficient technical depth. Since it is a class presentation students should also make good use of graphics, maintain eye contact with the audience, and field any related questions. Professors should not hesitate to ask tough questions regarding the subject matter and offer critiques following the presentation.

Below are a few examples of appropriate article topics:

Code Obfuscation	Protocol Vulnerabilities
Novel Web-based attacks	Usability vs. Security
Reverse Engineering	Pseudo Random Generators
Software Tamper-proofing	Static Analysis Tools

The second half of lab time can be dedicated to the team project. Teams can hold meetings, continue software development, add to their documentation, or work on their final presentation. It is in the instructor's best interest to fully investigate each team's progress and ensure they are meeting proposed deadlines. Once all the students have presented, the regularly scheduled lab time should no longer require attendance. However, the professor should be accessible during this time for the remainder of the semester to advise any of the teams.

4.5 Team Project

Secure software development is the core competency students hope to achieve at the conclusion of this course. The exercises help to teach secure code, but it's the development project that applies secure code. Students are tasked with completing a semester-long project that will incorporate all the phases of the software development life cycle including project planning, requirement analysis, software design, coding, testing, configuration management, quality assurance, documentation, and delivery. The lecture material parallels each stage of the Unified Process and applies software security principles during all phases of development.

5. PROJECT ELEMENTS

5.1 Member Roles

For this project, students are instructed to join into teams of 5-6 people. Caution students when selecting team members for the roles of each member will be different. Technical and non-technical diversity is encouraged. Each team will have weekly deliverables in addition to individual assignments given during class. Member accountability forms

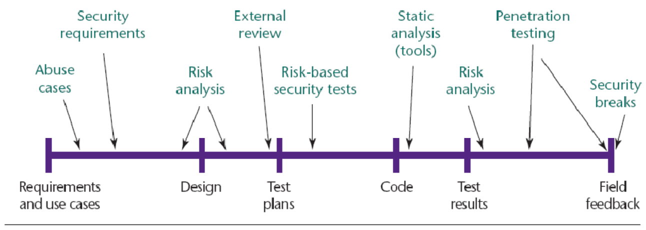


Figure 3: Software security best practices applied to development cycle. [13]

should also be provided to ensure no member is unsatisfactory in their duties. Responsibility is shared amongst all team members.

Each team should divide its members into differing roles. Since these roles depend mainly on the project topic, each team is welcome to create their own positions. However, each team must have at least a team leader. Typical roles include:

- *Team leader:* organizes meetings, manages design and implementation efforts
- *Test engineer:* designs, codes, and runs automated testing algorithms to explore the results of unexpected input, environment variables, privilege escalation, and other validation assessments
- *Support manager:* documents the development process, ensures project deliverables are coherent, updated, and accurate
- *Senior developer:* develops the core construct of the software, coordinates the efforts of subordinate developers
- *Coder:* the default role of all team members

The project itself requires each team to analyze, design, and build a substantial software application using the Unified Software Development Process. This process divides the project into four phases [1].

1) Inception - define the scope of the project and develop the business case

2) Elaboration - plan project, specify features, and baseline the architecture

3) Construction - build the product

4) Transition - transition the product to its users

These phases will serve as the framework for the project. Within this framework students will complete a series of stages that expands on each phase and derives additional approaches to the SDLC. For each stage, teams must submit related deliverables. Submissions must be revised of any coding/design changes and updated in the documentation. Stages include several principles of secure software as expressed in Gary McGraw's *7 Touchpoints of Secure Software* such as code review, abuse cases, and security requirements [12].

5.2 Process Stages

Stage 1 - Project Planning and Requirements Analysis

Determine project goals, estimated work schedule, tool selection, and management styles. Generate use and abuse cases to determine the system's behavior both when operated legitimately and under attack. In what ways could the project go wrong? What if the user provides unexpected input? Also discuss the security requirements such as privileged operations or, if needed, the type of encryption. How will the team meet these requirements? What methods will ensure the team completes these requirements and produces deliverables on time?

Submission 1 - The team submits a document outlining the project scope, organization, requirements, management process, technical process, and task schedule.

Stage 2 - Risk Analysis and Software Design

Define the project's operations and possible user interaction. Construct the program architecture and clearly document all vulnerabilities or possible attacks. What assumptions does the software make about its environment? List methods to track risks and monitor the program's activities. Does the application include any protection mechanisms to prevent architecture flaws?

Submission 2 - The team submits a document outlining the project's operations, class hierarchy, unsafe assumptions, attack mitigation, and techniques to monitor/reduce risk in subsequent development stages.

Stage 3 - Coding

Develop standardized, consistent, and well-commented source code. If done in Java, Javadoc comments are required. Employ a shared development solution and version management system such as Subversion. Append additional misuse and abuse cases when discovered. Investigate the security of any third-party components integrated into the application.

Submission 3 - The team submits a document outlining the software's core construct, content management system, defensive programming techniques, and any additional assumptions.

Stage 4 - Testing

Check if the implementation satisfies the design requirements. Formulate a testing plan that encompasses techniques for standard functionality testing and also risk-based security testing based on attack patterns. Investigate methods of black-box and white-box penetration. Focus on possible implementation flaws through external review and static analysis tools to scan the source code for common vulnerabilities.

Submission 4 - The team submits a document outlining the security testing plan, static/dynamic analysis tools used, discovered attack patterns, and the results of each member's risk analysis.

Stage 5 - Quality Assurance and Documentation

Discuss how users will know how to use and maintain the product. Develop an action plan for supporting fielded systems and cycling back any discovered system weaknesses. Will the project receive software updates? Is there a mechanism for users to submit feedback? What guarantees do the users have in quality assurance? Create adequate documentation to describe all aspects of the application.

Submission 5 - The team submits a document detailing the contents of a technical user manual and plans for a quality assurance program to monitor security flaws in the software post-deployment.

Stage 6 - Product Release

Review all documentation, source code, and functionality of the program at runtime. Compile related materials and prepare a package for final submission.

5.3 Final Submission

5.3.1 *Website*

Each team might create a Wiki that simulates a project notebook. It must include literature reviews, reference information, design notes, meeting minutes, report drafts, and any additional documentation the team feels valuable. Since the primary purpose of the website is to help the team, implementing this project component is in the team's best interest. Students should also use it to keep precise records of each member's responsibilities to enforce accountability and ethical responsibility. Encourage the teams to update their Wiki daily. This will enable them to develop good documentation habits.

At the professor's discretion, students may also license their source code under the Creative Commons and make it available for download. The site should be accessible from the internet and contain at least a few appealing visuals. Once the project is complete, students should also post their final presentation and report.

5.3.2 *Final Report*

The final report is a comprehensive document containing all the project related contributions from each team member. The report should begin with a table of contents and arranged with sequential page numbers. The sections thereafter will be from the Unified Process as described in class. Each section should include its primary author's name, along with a revision history.

5.3.3 Group Presentations

The final presentation should take each team 30 minutes to complete and should be subject to a peer review by classmates in the audience. The team should possess a dynamic presentation style with each member contributing to at least one aspect of the talk. Students should also set aside time at the end for questions and feedback. The below presentation format is suggested for students and has been adapted from a software engineering course taught at the University of Delaware.

- Overview: Describe what the system does and why it's a good idea.
- Architecture: How is the system organized? What processes run, where do they run, and how do they communicate? Where is data stored, and how is it accessed?
- Infrastructure: Disclose source code, build scripts, Wiki, and version control system.
- Functionality: Demonstrate all of the main Use Cases.
- Error handling: Describe how the software handles error conditions.
- Automated testing: Explain the team's testing methodology.
- Lessons learned: What went wrong during the project and how could it be prevented?

6. CONCLUSIONS

The state of computer science and engineering is always changing. New technology and new concepts have driven this dynamic technical field to the industry marvel it is today. Its evolution has contributed to triumphs and failures. From personal computing to mobile computing, the hot topics of years past have paralleled community interests and driven innovative novelties to societal integration. However, this final shift to security has been out of technological necessity and is a discipline that is here to stay.

Acknowledged by many professional developers as one of today's most critical issues, software security challenges the experience and expertise of its workforce [2]. The same technology and concepts mentioned earlier are transforming this domain, but the aggregation of knowledge remains unseen. Academia has tried to fill the gap. However, the complexity of the material and almost daily introduction of new security practices defies the anticipation of a stable and coherent undergraduate course.

Nevertheless, the difficulty of software security must not deter educators from teaching critical subject matter. The threat to the software industry is real and only continues to expand its influence with the deployment of insecure applications. Although the knowledge constructs for software security are in their infancy, implementing an educational strategy similar to the one described in this paper can help build a refined community focused on secure design and development, and ultimately resolve industry's challenge to produce secure code.

7. REFERENCES

[1] S. W. Ambler. *The Unified Process Inception Phase : Best Practices for Implementing the UP.* CMP Books, 2000.

[2] S. Barnum and G. McGraw. Knowledge for software security. *IEEE Security & Privacy*, 3(2):74–78, 2005.

[3] M. Bishop. What is computer security? *Security & Privacy, IEEE*, 1(1):67–69, Jan.-Feb. 2003.

[4] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1975.

[5] Cert coordination center cert/cc statistics 1998 – 2008. http://www.cert.org/stats/cert_stats.html (Accessed on 20 Nov 2008).

[6] E. Fernandez. A methodology for secure software design. In *Conference on Software Engineering Research and Practice (SERP'04)*, Las Vegas, Nevada, 2004.

[7] D. Fisher. Programming progress? *Information Security*, 10:p15 – 16, 2006.

[8] J. Geer, D., K. Hoo, and A. Jaquith. Information security: why the future belongs to the quants. *Security & Privacy, IEEE*, 1(4):24–32, July-Aug. 2003.

[9] M. Howard, D. LeBlanc, and J. Viega. *19 Deadly Sins of Software Security (Security One-off).* McGraw-Hill Osborne Media, 2005.

[10] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* Prentice Hall PTR, 2004.

[11] C. Y. Lester and F. Jamerson. Designing an undergraduate software security course. In *SECURWARE '08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, pages 257–262, Washington, DC, USA, 2008. IEEE Computer Society.

[12] G. McGraw. The 7 touchpoints of secure software. http://www.ddj.com/security/184415391(Accessed on 22 Nov 2008).

[13] G. McGraw. Software security. *Security & Privacy, IEEE*, 2(2):80–83, Mar-Apr 2004.

[14] G. McGraw. *Software Security: Building Security In (Addison-Wesley Software Security Series).* Addison-Wesley Professional, 2006.

[15] M. Newman. Software errors cost u.s. economy $59.5 billion annually. Technical report, National Institute of Standards and Technology, 2002. http://www.nist.gov/public_affairs/releases/n02-10.htm (Accessed on 20 Nov 2008).

[16] K. Tsipenyuk, B. Chess, and G. McGraw. Seven pernicious kingdoms: a taxonomy of software security errors. *Security & Privacy, IEEE*, 3(6):81–84, Nov.-Dec. 2005.

[17] R. Walling. Computer science enrollment is going down, and taking software jobs with it. *Software by Rob*, June 2007. http://www.softwarebyrob.com/2007/06/27/computer-science-enrollment-going-down-taking-software-jobs/ (Accessed on 20 Nov 2008).

Exhibitor's Session

MyitLab*

Pearson Canada

Abstract

Do your students struggle with Office 2007? Are you interested in improving student evaluations of your course? If so, we welcome you to join us as we hear a fellow faculty member share his experience with myitlab and how it has helped him to be successful in teaching Office 2007 to his students. The presentation will explore the features of myitlab and the benefits it provides to students. The discussion will also include teaching with myitlab: what is the benefit for instructors and how can it be incorporated into the classroom.

Myitlab is an online study and assessment tool that simulates the Office 2007 environment. It provides students with skill-based and project-based exercises that they can complete to learn the office skills necessary for their course. Within myitlab, instructors can provide students with simulations containing hints, to guide them to the correct action when they are stuck. All work that students do in myitlab is recorded in a gradebook, which provides instructors with powerful, at a glance diagnostics so that they can see how their students are doing.

Categories & Subject Descriptors: K.3.1 **[Computers and Education]**: Computer Uses in Education – Computer-assisted instruction.

General Terms: Management.

Bio

The speaker, Randy Jenne, has been teaching college computer courses for over 25 years in areas such as business computing, networking, web development, and Cisco certification. He recently earned his M.Ed. with a focus on educational technology and currently works in the Grant MacEwan College School of Business developing and teaching online, hybrid, and conventional IT courses.

* Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
ACM 978-1-60558-415-7

Special Session

eScience Inspired CS Education

Yan Xu
Microsoft Research
One Microsoft Way
Redmond, WA 98075
yanxu@microsoft.com

Abstract

Rapid advances in computing technology has fundamentally changed the way scientists conduct research. Meanwhile, real-world computational challenges have become one of the major driving forces of computer science innovation. This interdisciplinary paradigm in research is making a significant impact to CS education and curriculum development. Microsoft Research has been deeply involved in computational science research (eScience) and education by collaborating with academia in over 50 disciplines. I will talk about how we are making an effort to facilitate pedagogy research to infuse computational thinking into science education, share with you what we have learned and achieved from our academic partners in CS and natural sciences, and discuss how we can work together to create a generation of interdisciplinary computational thinker.

Categories & Subject Descriptors: K.3.2 **[Computers and Education]**: Computer and Information Science Education – Computer science education.

General Terms: Experimentation

Bio

Yan joined Microsoft Research in March 2006. Her research has been focused on exploring technologies and pedagogical strategies that facilitate and enhance interdisciplinary computational research and education. She is responsible for the WorldWide Telescope Academic Program, which enables collaborations with academic researchers and educators in computer science and astronomy; the Transform Science–Computational Education for Scientists initiative, which enables collaborations with academia for infusing computational thinking into science education to create tomorrow's scientists; and the Phoenix Academic Program for applying Microsoft Phoenix technology to computer science research and education.

Prior to working at Microsoft Research, Yan was a Sr. Software Architect and worked for several startup software companies for over ten years. Yan received her Ph.D. in Physics from McGill University, Canada.

Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada ACM 978-1-60558-415-7

Computational Biology Unplugged!

Sarah Carruthers
University of Victoria
Victoria, BC
1-250-240-6624
scarruth@uvic.ca

Kat Gunion
University of Victoria
Victoria, BC
1-250-472-5715
kgunion@uvic.ca

Ulrike Stege
University of Victoria
Victoria, BC
1-250-472-5786
stege@cs.uvic.ca

ABSTRACT
In this workshop, we present an "unplugged" computational biology (or bioinformatics) activity. The activity, which may range from beginner to advanced in difficulty based on the interest of participants, will present a topic of interest in computational biology or bioinformatics such as: gene sequencing, phylogenetics, or parsimony. Bioinformatics is a rich and diverse area of study, in which new discoveries (mathematical, computational and biological) are occurring. Based on the pioneering work Computer Science Unplugged (by Michael Fellows, Tim Bell and Ian Witten), these activities aim to present topics of interest in computational biology in a hands-on way to encourage active participation and learning.

General Terms
Algorithms.

Keywords
Computer Science Education, Elementary Grades, Middle School.

1.INTRODUCTION
As demonstrated in the book "Computer Science Unplugged", by Tim Bell, Michael Fellows and Ian Witten, many advanced topics in computer science can be presented to young students, without the use of computers. Computer science is not a required component of the curriculum at an elementary grade level in most areas of the world, which means that most elementary school teachers are not required to have studied any computer science in order to gain accreditation. Overly technical lesson plans could discourage many elementary teachers from offering computer science activities in their classrooms. The activities in "Computer Science Unplugged", however, are presented in a manner which is accessible to teachers with no background in computer science. Elementary school teachers have also expressed that they are reluctant to use technology-dependent lessons in the classroom. While sometimes this may be because the teachers are themselves uncomfortable with computers, it is often due to issues with technical support with school computers and computer labs. By designing computer science activities which are not dependent on computers, the authors of "Computer Science Unplugged" ensure that issues with IT do not prevent teachers from being able to offer computer science enrichment activities to their students. The activities in this book have been deployed successfully to students in elementary and secondary schools around the globe. Based on this success, the Solving Problems with Algorithms, Robotics and Computer Science (SPARCS) group at the University of Victoria has begun developing new activities, some of which are based not only on traditional computer science problems and algorithmic techniques, but also on problems and approaches in computational biology, or bioinformatics.

Computational Biology (or Bioinformatics) is a rich and diverse area of study, providing many new and interesting computational problems for computer scientists and biologists alike. Due to its interdisciplinary nature, computational biology may also be able to spark the interest of more female students in computer science, as biology typically has a larger proportion of female students than computer science. With these ideas in mind, the SPARCS group has begun developing and deploying unplugged computational biology activities for young students. These activities have been successfully piloted in public school classrooms, outdoor workshops and after school clubs, with students ranging in age from 5 to 18.

In this session, we present an "unplugged" computational biology activity. The activity, which could range from beginner to advanced in difficulty (based on the interest of the participants), will present a topic in computational biology or bioinformatics such as: gene sequencing, phylogenetics, and parsimony. This session aims to teach a lesson in computational biology, in a hands-on, experiential fashion, to encourage active participation and learning.

2.REFERENCES
[1] Bell, T., Witten, I. and Fellows, M. 2002. Computer Science Unplugged. http://www.lulu.com/content/166249.

Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
ACM 978-1-60558-415-7

Workshop Session

Using Subversion in Your Class*

Gregory Baker
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
ggbaker@cs.sfu.ca

Abstract
In this interactive workshop, we will explore the use of the Subversion version control system in a class, particularly one that involves a group project. Topics will include the basics of version control, using Subversion, creation of shared repositories given various technical restrictions, resources for students, and discussion of how instructors can enhance their teaching using a version control system. Participants with laptops will be invited to install a Subversion client and explore a shared repository as part of the workshop.

Categories & Subject Descriptors: K.3.1 **[Computers and Education]**: Computer Uses in Education – *Computer-assisted instruction*; K.3.2 **[Computers and Education]**: Computer and Information Science Education; D.2.7 **[Software Engineering]**: Distribution, Maintenance and Enhancement – *Version control*.

General Terms: Management.

Keywords: Group project, Subversion.

Bio
Gregory is a senior lecturer at Simon Fraser University, where he received his M.Sc. in 2000.

* Copyright held by the author/owner(s)
WCCCE'09 May 1-2, 2009 Burnaby, BC, Canada
ACM 978-1-60558-415-7

Author/Panelist Index