Advances in Engineering Education

SPRING 2009

Leveraging Cognitive Load Theory, Scaffolding, and Distance Technologies to Enhance Computer Programming for Non-Majors

THOMAS J. IMPELLUSO, PH.D.
San Diego State University
San Diego, California 92182

ABSTRACT

Cognitive Load Theory (CLT) was used as a foundation to re-design a computer programming class for mechanical engineers, in which content was delivered with hybrid/distance technology. The effort confirmed the utility of CLT in course design. And it demonstrates that hybrid/distance learning is not merely a tool of convenience, but one, which, used purposefully, enhances the learning experience.

To ensure scaffolding of domain knowledge, the content was both the syntax of a structured language, and also the algorithms encountered by students of mechanical engineering: Gauss Reduction and Newton-Raphson. To maximize germane (motivational) learning load, course material scaffolding occurred vertically, connecting language syntax constructs to each other, and also application algorithm constructs to each other. Scaffolding also occurred horizontally, connecting language syntax to application algorithm. Finally temporal scaffolding occurred in which application algorithms were connected to examples of simulation science and engineering, including, for example, finite element, multi-body dynamics and computational fluid mechanics.

Comparative evaluations demonstrate improved student learning outcomes, streamlined and enhanced course delivery, improved instructor evaluations, a great cost savings to the department, and a transition from low to high enrollment. The department need no longer hire lecturers or provide and upgrade workstations.

Keywords: Cognitive load theory, hybrid learning, computer programming

I. INTRODUCTION

The discipline of Mechanical Engineering is in flux. Today's machines "think," and they also "communicate" with each other. Thus, it is incumbent on mechanical engineers to understand the language that enables machine-to-machine communication. But there are more profound challenges, as well as opportunities, confronting this discipline.

In January of 2003, the U.S. National Science Foundation identified cyber-infrastructure (CI) as a suite of tools to enhance research in "Simulation Science" [1]. Simulation Science is essential to the study of engineering systems. It includes traditional sub-disciplines such as finite element methods, multi-body dynamics methods, and computational fluid mechanics; but it also includes non-traditional disciplines such as physics-based animation for films and games. This latter portends a stronger incentive for mechanical engineers to embrace computer science beyond its traditional role of implementing algorithms to solve field equations in mechanics. Engineers should move more decisively into the intersection of their discipline with computer science, but this necessitates a more vigorous deployment of computer and information technology in the mechanical engineering curriculum and this, in turn, is fraught with pedagogical obstacles.

For example, it has been demonstrated that computer programming poses great difficulty for beginning students in general [2, 3] and specifically for mechanical engineering students not well versed in computer science. In a computer programming course, students must master essential skills concerning the syntax of the language. However, there are two extraneous content areas that must also be mastered: 1) the editor to write the code, and 2) the operating system on which to run the code. This interdependent complexity can overload cognitive abilities, posing challenges to course designers and, more importantly, learners. With these challenges in mind, a computer programming course was offered to beginning students of mechanical engineering at San Diego State University. To obviate probable cognitive overload, recourse was taken to pedagogical methods (Cognitive Load Theory) and instructional technologies (Hybrid Learning).

"Cognitive Load Theory has been designed to provide guidelines intended to assist in the presentation of information in a manner that encourages learner activities that optimize intellectual performance" [4]. As an example, consider the obstacles in learning new material in a non-native language: clearly, there is an overload (learners must master the new material and the language) that, in this case, is resonant with the challenge of learning to program a computer (learners must master operating systems and the syntax). CLT can mitigate challenges in such cases when learning loads are high.

According to CLT, information can only be stored in long term memory after first being properly integrated, by working memory, into a mental structure that represents the schema of the material. However, the faculty of working memory has limits and this, unfortunately, can hinder learning, especially when many extraneous facts compete to challenge the cognitive learning loads. CLT posits that there are three basic types of cognitive loads placed on a learner:

- "Intrinsic cognitive load" was first described in 1991 [5] as the essential material to be learned. Accordingly, all instruction has an inherent difficulty associated with it and this intrinsic material may not be altered by an instructor. In learning a foreign language, this includes the vocabulary and syntax.

- "Extraneous cognitive load" is generated by the manner in which information is presented to learners [6] and, in the case of a programming language, the ancillary information such as text editors, compilers and operating systems. Or, in the case of a spoken language, the technologies such as language labs or voice recordings.
- "Germane cognitive load" was first described by Sweller, van Merrienboer, and Paas in 1998. It is that load devoted to the processing, construction and automation of schemata necessary to integrate knowledge into consciousness. This includes motivations to learn and how the knowledge is situated in the rest of the curriculum such as reading novels, or programming mathematical algorithms.

These three loads are additive in the learning process and research suggests [7] that when courses are redesigned without reference to the interaction of cognitive loads, learning is undermined. For example, while intrinsic load is generally thought to be immutable, instructional designers can exercise the option to manipulate extraneous and germane load. With complex material, it is best to strive to minimize the extraneous cognitive load and maximize the germane load.

CLT has had success in other disciplines [8–10]. Others have recommended it specifically for the process of learning to program [11].

Table 1 presents this author's view of the various learning loads experienced in computer programming. The intrinsic learning load, concerning syntax, is high in computer programming. Thus, if one also employs methods that add an extraneous load, it is very likely that there will be very little capacity left for germane load and this will obstruct learning. These issues were all addressed in this new course in computer programming discussed in this paper.

Four other issues were also considered in this course re-design. Various media play a role in learning; and, more importantly, proper integration of these various learning modalities could have rewarding consequences. In addition, many methods have proven to be effective at facilitating learning [12–14]. Thus, the author applied several advanced instructional technologies in this re-design. This deployment was designed to enable student choice and 'on-demand' learning: students should be able to select from a series of modalities provided by the instructor, including

Load	Examples
Intrinsic	Syntax: data types, loops, logical tests, arrays, functions
Extraneous	The complex interplay between syntax, text editor and operating system
Germane	Numerical Algorithms in Computational Mechanics

Table 1. The Three Types of Cognitive Loads Placed on Learners of Computer Programming.

archived documents, pre-recordings of the lecture, interactive on-line sessions and, recently, distributed DVD's of the recorded on line sessions. In this way, those students who do choose to physically attend class are those most in need of one-on-one guidance and more attention can be afforded them.

Second, it is known that anything that is necessary, but not directly part of the learning objectives, should be presented before the students start working on the instructional goals [15]. This often means that it may be advisable to separate and categorize elements of the presented material and bring attention to them at the start of the class. This was also taken into account in the re-design to mitigate extraneous load.

Third, it has been shown that complete, thorough, and fully commented programming examples provide greater motivation for novices than simply working out problems from scratch [16, 17]. Although this may seem counterintuitive, tests have demonstrated that studying complete examples facilitates learning more than actually solving the equivalent problems [18]. Additionally, in many cases, a variation of worked examples balanced with assignments was used [19]. Students can be urged to *complete* the solution, which is only possible by the careful study of the partial example provided in the completion task. And like completely worked examples, this serves to decrease extraneous cognitive load [20].

Fourth and finally, the necessary ingredient to enhance the germane mode is through scaffolding. Scaffolding became an essential ingredient in this course re-design. And this approach is supported by existing research that has been successfully applied to the domain of computer programming [21].

II. DEFINITIONS AND TOOLS

This section provides a brief description of each technology exploited in the re-design of ME203—Computer Programming Applications.

A. SERVER

Since the term server will be used in this text, it is appropriate to first define it, then justify its use and finally describe the one used. First, computers communicate via the fundamental inter-process communication facilities bundled in the BSD Berkeley Socket interface. Any computer, regardless of its power or speed, becomes a server merely by running a computer program that issues the six fundamental calls of inter-process communication:

socket(), bind(), listen(), connect(), read(), write(),

thereby giving other computers access to its computer programs. (For that matter, even the laptop on which this paper is being written can be a server, once a simple client/server code is written and instantiated).

Second, it may seem ostensibly convenient to expect students to acquire the necessary compilers and develop familiarity with them. However, the high schools today are not teaching computers; they are teaching Microsoft Windows or Macintosh operating systems. Faced with this daunting obstacle, it was decided to pursue a homogenous computing environment. A UNIX server was selected because such a platform (after an SSH connection), results in a computational environment unencumbered by the paraphernalia, complexities, and nuances of windows-based operating systems. After all, the goal here is to write, edit, and execute codes which implement numerical algorithms, not learn advanced compilers, code-generators, or window environments.

Finally, the author presents the server that was used. In this course, the name of the server machine was attila.sdsu.edu. Attila.sdsu.edu is a Sun Enterprise 4500 server that contains 4,400 MHz UltraSPARC II CPUs, each with a primary 16-KB instruction and 16-KB data on chip cache. The system interconnect is a 3.2 GB/sec (at 100 MHz) gigaplane. Attila has seven CPU/Memory boards and one I/O board. The total physical memory size is 6144MB. Attila has one Fast Ethernet (10BASE-T/100BASE-T) twisted-pair standard connector (RJ-45). Attached to Attila are three Sun A1000 storage arrays in a RAID5 configuration for student accounts. Thus, over one hundred students can simultaneously connect to Attila and code their homework and assignments; and each student experiences the same computational environment. Students connected to Attila.sdsu.edu and wrote, compiled and ran their codes on it.

B. Secure Shell (SSH)

Secure Shell (SSH) is a network protocol that allows exchange between two networked computers using a secured connection. It is used primarily on Linux and UNIX based systems to access shell accounts. Fundamentally, SSH allows users to open a command window, connect to a remote server, and execute shell commands (actually type them) on that system, unencumbered by intrusive and complex point-and-click environments. An SSH server code ran on Attila.sdsu.edu.

SSH clients are readily downloadable from many sites on the internet. With an SSH client, students are able to use their computer at home, whether Windows or Mac, and connect to the UNIX server on which they compile, debug, and execute both the sample programs in the class or the codes they write themselves in response to assignments or homework.

C. Blackboard

Blackboard (http://www.blackboard.com) is a course management software system widely used by many universities, and was the system that was deployed in this course re-design. With

Blackboard, an instructor can upload documents for student review, create discussion boards for the entire class (and seed them), facilitate email contact with students, create student led working groups, gather assignments, and assign grades. All these pedagogical tools of Blackboard were used in the re-design of the class.

At this time, however, the author is experimenting with an open source version called Moodle (http://moodle.com).

D. Wimba

Wimba (http://www.wimba.com) is a leading provider of collaborative on-line learning software applications and services to the education industry. Its collaborative software applications for online and blended education enable institutions to bridge technology and pedagogy by supplementing course management systems such as Blackboard. Wimba enables educators and students to quickly and easily teach and learn online, engage in live chat and instant message exchanges, and benefit from verbal, spoken content being added to text-based course content. The most critical aspect of Wimba, and one used heavily in this course, is the application-sharing tool, which enables an instructor to control the desktop of remote students and allows third party students to observe the interaction.

Another very attractive aspect of Wimba is that it is also rolled into Blackboard. Thus, the course management system and the distance delivery technology are closely bound and this proved to be immensely beneficial to the teaching process, as well as for students in the learning process.

At this time, however, the author is experimenting with an open source version called WiZiQ (http://www.wiziq.com).

III. COURSE PRIOR TO RE-DESIGN

A. Course Material

The course under discussion is ME203—Programming. In this class, the C programming language is taught in a UNIX environment. The course is designed to teach a *procedure oriented* language (as opposed to *object oriented* language such as Java or C++), because mechanical engineers are more concerned with the *process* of applied mathematical algorithms (solids, fluids, thermal studies) than with *objects* to be manipulated (computer graphics, bioinformatics). Of the procedure oriented languages (e.g., C vs. FORTRAN), C was selected because it is the language in which most operating systems are written. If a long term goal is to encourage simulation science studies and machine communication, then C is an ideal choice.

B. Course Delivery

Prior to Fall 2006, the class met physically, and the exclusive method of content delivery was through face to face lecture. Instruction was provided in a workstation laboratory. This laboratory was a dedicated computational resource cluster of 30 UltraSPARC models 170 and 170E workstations using the Sun Grid Engine software from Sun Microsystems. Each station in the cluster had 128MB of physical memory, and contained one 167MHz US-I CPU. The workstations were interconnected using high-speed network infrastructure from Myricom.

The instructor taught at one workstation and displayed his monitor on an overhead projector. Students were able to watch the instructor discuss the code line-by-line, compile it, and run it. Then, students would work on their own code in a separate lab session.

This model of instruction had limits. First, the size of the class was limited to the number of workstations. Furthermore, the workstations had to be upgraded every few years at considerable expense. Second, the students often expressed frustration as to why they were learning the material. Student reviews consistently mentioned that there was no reason for mechanical engineers to learn programming. In this light, a course re-design was initiated.

IV. COURSE AFTER RE-DESIGN

There are three essential aspects to this on-going course re-design: 1) the material, 2) the delivery, and 3) the motivation. Scaffolding played a role in each.

A. Course Material

Two levels of course material were interlaced by themselves and with each other: 1) the syntax of the language (data types, loops, logical tests, arrays and functions), and 2) the applied *mathematical algorithms* (vector, matrix manipulation, Gauss Reduction and Newton-Raphson methods). They were scaffolded as follows.

1. Two Phase Vertical Scaffolding

The left column of Table 2 indicates the *syntax structures* that were discussed in the class. Every complete and commented syntax was scaffolded on the skeleton of the preceding code. Thus, loops were explained in the context of logical structures, and arrays were presented in the context of loops.

The right column of Table 2 indicates the *mathematical algorithms* that were discussed in the class. Every complete and commented algorithm was scaffolded on the previous algorithms. Thus, Newton-Raphson relied on the Gauss Reduction code; Gauss-Reduction relied on matrix manipulation and matrix manipulation relied on vectors. The same code “grew” and “evolved” in each example.

2. Horizontal Scaffolding

The driving focus of instruction was the algorithm rather than the syntax. This inverts the way programming has traditionally been used in which syntax rules are presented and are the focus. Furthermore, there were no coding examples of sorting, alphabetizing, or interest rate problems that plague introductory computer programming courses for mechanical engineers.

Table 2 indicates the algorithms in the first column, followed by the programming syntax in the second. Vertically, the table demonstrates the order in which both topics were addressed. It is important to note that numerical algorithmic convergence and stability issues were ignored, in lieu of the most simple algorithm implementation. This table demonstrates the horizontal scaffolding that occurred in the class. By connecting algorithm to construct, the re-design enhanced the germane load of the student learners.

The number guessing program (wherein users guess a number and the code responds if the guess is too high or low) is provided and discussed in the first week of class. The purpose of this forceful thrust of new material is to introduce a real gaming example which also reveals (but does not dissect) fundamental programming syntax: loops and logic. The instructional goal is to challenge the students to read the code as if it were a new language. It is not expected that the students master the code's nuances and reproduce them at this stage. Rather, the goal is to immerse the students in a new language and expect them to follow the general idea of how the language implements the logic of a simple game.

The Gauss Reduction is the algorithm to solve a system of linear equations, while Newton-Raphson is for a system of non-linear equations; both are critical components in mechanics-based Simulation Science. However, there is serendipitously something more profound which was exploited here: the Newton-Raphson method builds upon the Gauss Reduction method. This creates an overarching structure to the class as it drives toward the study of very simple non-linear systems.

Algorithm	Syntax
High-low number guessing game	logic loop
Temperature Conversion	data types
Bisection Method	logic loop formality
Newton's Method	logic loop formality
Numerical Integration	logic loop formality
Repeat of all previous algorithms	files
Matrix-Vector Multiplication	Arrays
Gauss Reduction	Arrays, files
Gauss reduction with functions	Arrays, files functions
Newton Raphson Method	Arrays, files, functions, memory

Table 2. The Scaffolding of Algorithm and Syntax After Redesign of the Course.

B. Course Delivery

With regard to delivery, two modes were used in equal parts: 1) face to face, and 2) interactive, on-line application sharing. Half the classes were face-to-face, and this is where algorithm and syntax were taught. Extensive PowerPoint slides were developed and they were tied to each item in Table 2. Face-to-face lectures concerned the interplay of intrinsic and germane learning loads.

The extraneous learning load was obviated, however, by use of on-line instructional technologies. However, this was not a passive use in which students simply observed lectures. Full application sharing was used, in which the instructor took control of student laptops.

The schematic for an on-line session is indicated Figure 1. Students were able to work on their assignments from home (during or after class lab-time), regardless of their operating system, by first establishing a terminal SSH session to the server on campus using a monitor prompt. Once connected to the server machine, students are able to write, compile, and debug their codes.

Meanwhile, the instructor also maintained an SSH connection to the same server and exploited the application sharing interface of Wimba. The instructor shared his desktop (which contains an SSH connection to the server) with the class and demonstrated the process of writing, compiling, debugging, and running example codes.

Occasionally, whether during a class session or during "office hours," a particular student may request assistance with an assignment (indicated by the laptop with the "?" mark and his SSH connection by the largest black monitor window external to his laptop). At those times, the instructor activated the Wimba application sharing interface and asked the student to share his/her desktop

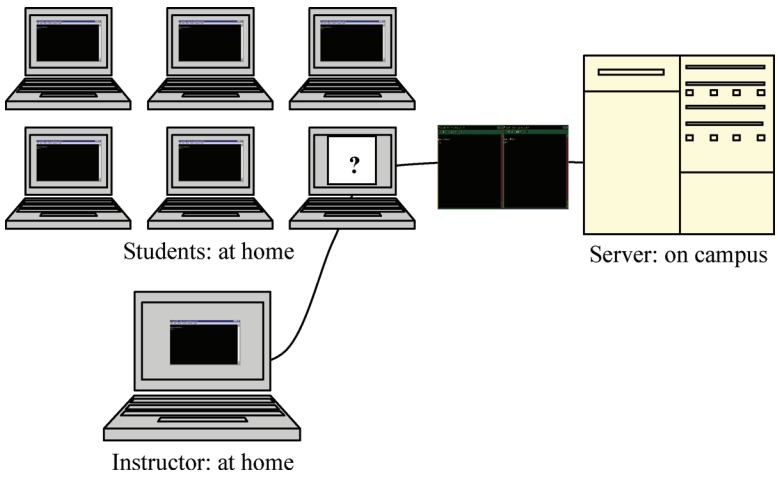

Figure 1. Student-instructor connections for online class sessions.

with him and the rest of the class. Then, the instructor addressed the student's questions, while also sharing the information with all of the students.

This summary warrants a focused reiteration. Extraneous learning loads were lessened by on-line, desktop sharing; these learning experiences demonstrated code writing and compiling, interactively. The instructor, at home, used Wimba to pass through the workstation of the student (also at home) and continued on to the student's account on the server (on campus), fixing the code and sharing the lesson with sixty other students (also at home). The instructor then archived the session, which enabled other students to play it back at their leisure. In this way, students were able to observe codes being written, edited and recompiled. Also, all the writing and compiling occurred in a common workstation environment, unencumbered by the nuances of diverse compilers. This consistency—this common environment—reduced the extraneous load of learning operating systems, compilers and text editors; students were able to focus attention on the syntax of the language and the mathematical algorithms.

C. Additional Factors: Motivation

To summarize so far, the extraneous learning load was softened by the use of on-line learning laboratories, brokered by a web conferencing learning tool. It now remains to enhance the germane learning load. This was done through a third type—temporal scaffolding.

Temporally, each applied mathematical algorithm was then scaffolded back to examples of 3D simulations and animations found in engineering and entertainment; and this is how the germane load was intensified. This will be called temporal scaffolding, since it returns student attention to the first day, as well as the final day of the class.

The instructor secured approximately a series of MPG videos of mechanics software: finite element, computational fluid dynamics, heat transfer, multi-body dynamics, collision detection, and various animation sequences created by entertainment arena tools such as Maya, 3DStudioMax, and Blender. Examples include physics based animations of blade loss, piston motion, truss deformation, artificial heart pump, flight of paper planes, bouncing balls, auto collision, femur deformation, a Slinky® bouncing down the stairs, machining, guitar string vibration, bridge vibration. In all cases, the instructor revealed how such simulations would not be possible without resorting to code writing and, more specifically, adapting the two fundamental algorithms—Gauss Reduction and Newton Raphson—of mechanics simulations. A list of such videos is provided herein, although a search of youtube.com will provide many more in which instructors can rely on their own expertise.

Bridge Vibration: http://www.youtube.com/watch?v=dvGkTz_DXx0&feature=related

Rubber Cube: http://www.youtube.com/watch?v=WtE4qhuS5-s&feature=related

Truck Crash:	http://www.youtube.com/watch?v=6vWbfKKJUD8&feature=related
Femur Rotation:	http://www.youtube.com/watch?v=ZMKG85EFxxE&feature=related
Machining a part:	http://www.youtube.com/watch?v=BEzwpBwf9I0&feature=related
Guitar Deformation:	http://www.youtube.com/watch?v=zmb_i-zTASw&feature=related
Artificial Heart:	http://www.youtube.com/watch?v=ppQbJyji32I&feature=related
Truss Deformation:	http://youtube.com/watch?v=Qw8bm3eBcVU&feature=related
Fluid Mechanics:	http://www.youtube.com/watch?v=V1A4AZGwLcM&feature=related
Piston Animation:	http://youtube.com/watch?v=Tn4Ltgb9uZo&feature=related
Blade Loss:	http://www.youtube.com/watch?v=uYnoODMc3Xs

V. ASSESSMENT

This course was subjected to both qualitative and quantitative faculty and course assessments. The same instructor has taught this class for ten years; thus, any improvement in assessment that resulted from the re-design cannot be attributed to recent mastery of the material. Three assessment periods are provided. Fall, 2006 assessments were for the class before there was any re-design. Fall, 2007 assessments were for the first offering of the course in which many instructional methods were first explored. Spring, 2008 represents the first semester in which the instructor gained facility with the instructional technology and the modified curriculum.

A. Quantitative Assessment

The class was reviewed on-line by students, anonymously, at the end of the semester, and before student grades were posted. Students were asked to comment on the statements in the Table 3. The students rated their response on a Likert Scale with 5.0 representing complete agreement and 1.0 representing disagreement. Results from Fall, 2006 (before the re-design) were compared with results from Spring, 2008 (after three iterations of the re-design).

The Table reveals increases in all student attitudes, and with significance. This cannot be attributed to grade inflation, as the average grade assigned remained consistent throughout the re-design In fact, despite the consistent grade that was offered, the difficulty of the exams increased as did the amount of homework; the latter progressing from four assignments per semester to fourteen. And the table also underscores little significant change in grading strategy: 4th row.

The last five items represent student attitudes regarding the instructor: feedback, accessibility, communication, effectiveness. There were significant increases in all five areas with data that is significant ($p < .01$).

Question Prompt	Mean		Standard Dev.		t-value	p-value
	F'06	S'08	F'06	S'08		
Objectives of the class were clearly defined	2.84	3.85	1.03	1.13	4.11	<.001
Organization of the material	2.92	3.63	1.04	1.05	2.91	<.005
Ability to stimulate your interest in the subject	2.48	3.61	1.26	1.16	4.27	<.001
Testing/grading was fair	3.72	3.83	1.06	1.13	0.44	>0.10
Feedback (on assigned work/exams) was timely	3.60	4.12	1.00	0.83	2.24	<.01
Generally accessible out side the classroom	3.88	4.62	0.88	0.83	3.32	<.01
Clearly communicated the subject matter	2.52	3,78	1.19	1.12	4.87	<.01
Overall teaching effectiveness	2.68	3.76	1.19	1.01	4.46	<.01
AVERAGE OF ALL	3.08	3.90	1.17	1.08	3.21	<.01

Table 3. Course Statistics Before and After Implementation of Course Redesign: (F'06 had 25 N = 25 students; S'08 had N = 59 students).

Comments	Negative		Indifferent		Positive	
	F'06	S'08	F'06	S'08	F'06	S'08
On the course	78%	5%	11%	48%	11%	47%
On the instructor	45%	6%	36%	9%	19%	85%

Table 4. Responses to Qualitative Questions Before and After Redesign.

However, the first there items represent how the re-design impacted the organization of the material. Here there were also great increases, and the data was extremely significant ($p < .001$)

B. Qualitative Assessment

Students were asked to respond to two questions: 1) What are your comments/suggestions for the course?; and 2) What are your comments, positive or negative, on the instructor's teaching? These were written response, and not scaled. These written responses were far more detailed in Spring 2008 than the Fall 04 assessments—students took the time to give extensive comments. Naturally, space limitations limit the listing of comments, so they are summarized into three

categories: negative, neutral and positive. Responses from the Fall, 2006 semester and Spring 2008 semester are presented in Table 4.

This re-design has clearly excited the students. Further data indicate enrollment has increased (from 30 to 80 students). Students take a more active part in their learning (as demonstrated by the timbre of their responses). They are more active in offering advice on how to improve the class, and, this, in turn, makes it easier to continually improve the course through consistent re-design. One student comment is excerpted below:

> "[The instructor] is the most accessible, straightforward, fair teacher that I have ever had in my life. He fully employs every method possible to teach us, including blackboard, a separate class website, wimba 'liveclassrom' and archives, powerpoint and word outlines, emails on a regular basis, in-classroom instruction, and was even very active on the discussion board. I do not doubt that he spent more time on this class than I did on all of my classes combined. He made every effort (and I mean every) to keep his grading scheme fair. If a single person expressed their confusion with a topic, he made it a point to comprehensively review the topic during the following class. I guess I could say it got a little redundant hearing the same material multiple times after I already understood it, but the repitition really made it stick in my mind. I feel like I learned a lot about what the next steps in programming and engineering look like, and am very interested to continue to pursue these fields as a career.

In this one typical comment the student took the time to review the entire pedagogy and comment upon it. In fact, student reviews have become much more detailed.

VI. DISCUSSION

It appears from this work that CLT can be used to guide the deployment of advanced instructional technologies. Learning Technologies can enable an instructor to revisit many aspects of class, mitigating some obstacles and enhancing learning in other areas. CLT may provide an architecture on which to do this purposefully, but not just deploying technologies simply because they are available.

A. Cost Savings

As a result of this new foray into distance learning, the workstation cluster once used to instruct the class in exclusive face-to-face lectures is no longer needed. This has already amounted to

nearly $40,000 in savings to the department. Furthermore, the distance delivery has enabled the class to overcome the enrollment limitation dictated by the available workstations. Enrollment has progressed from under 30 students to now over sixty in one session, obviating the need to hire a lecturer to support additional sections. As of this writing, the original laboratory has been removed and the space is now utilized for office space.

B. First Week Intensive Training

In time, perhaps students will be more accustomed to on-line technologies. Until then, the author reports that many short on-line sessions were given during the first two weeks of the class. Each and every evening, for one week, the instructor met on-line with as many students as wanted to attend. The purpose of these sessions was to acclimate the students to the technology they would be using.

C. Homework, Assignments, Groups, Web-site and Additional Assistance

The items here were not exploited fully in the Spring, 2008 semester; it was stumbled through and will be more fully activated in Fall, 2008, with some improvements. First the instructor will distinguish between homework (which is for private practice and study) and assignments (which are in-class laboratory efforts to encourage performance). Second, the instructor experimented with groups late in the Spring semester, but will assign students to groups at the start of the next (Fall) semester. A supplemental course web-site exists: http://attila.sdsu.edu/me203 and will continue to be utilized. Finally, the course syllabus has been supplemented with step-by-step tutorials on how to download SSH clients, and how to use Wimba and Blackboard.

D. Plateau Instruction

The author of this paper engaged students in several personal discussions about their learning experiences and these are summarized herein. This section is entirely anecdotal, very personal, and without reference to the literature. It reports on feelings and thoughts expressed by students.

It seems that students are able to approach their high school studies in a disintegrated way. Consider Biology: the material and the textbooks are cleanly segregated; information is sequentially presented on the digestive system, the cardiac system, the muscular system, the nervous system, and the skeletal system. Students are quite capable, they tell the author, of mastering one system, failing another, and still securing high grades in the end. The same can happen in high school literature classes (authors are disintegrated), and physics (kinematics, thermal studies, optics, and nuclear studies).

Many of the students verbally shared their feelings that they never expected the course to stack the way it did and they attributed this to their poor performance. And this justified the introduction of the two mid-semester review periods conducted in the class. The instructor stopped all progress

in the class and reviewed all material. Rather than consider this a simple review period, placing an onerous burden upon the instructor, the instructor now considered this to be a "plateau" experience, wherein no new material was presented. This enabled students who underestimated the course to catch up and not suffer penalty.

E. On-Line for a purpose

SDSU has a policy that a course is "hybrid" if 45% of instruction occurs on-line. This compelled the course designer to initially hold physical sessions (55%) and on-line sessions (45%) indiscriminately and one after the other. In the second roll-out, however, the instructor began using the on-line sessions for laboratories to diminish extraneous load. The author advises adopters of distance learning to seek out which aspects of a course are suitable for distance learning and deploy those first.

F. Utility of Programming

Finally, some students had expressed a frustration on learning to program. The instructor has continued to mitigate this criticism several ways. First, there is the repeated use of videos which demonstrate the need to program for simulations. Second, demonstrations are given of Simulated Worlds (SIMS) such secondlife.com, realxtend.com, tribalnet.com and opensim.com. These are all SIMS in need of more accurate physics coding, underscoring the need for some mechanics students to learn programming. Third, the author demonstrates the physics coding in various on-line games such as "crayonphysics.com." Finally, the author demonstrates how one can quickly write socket codes that enable computers to communicate with each other. As a result of these examples, the author now receives regular emails from students such as the following:

> Hello Dr. Impelluso,
> As it says in the subject line, I was in your class this past semester. I'd just like to thank you for such an awesome time, this class was definitely one of the more fun ones in my schedule. Also, I wanted to ask about a program you mentioned during one of the lectures. It was a freeware program based on the C language where you could do some 3d imaging or graphic design of some sort. I forget the name of the program or the website and I'd greatly appreciate it if you remembered it. Either way thanks again, and I hope you don't mind if I occasionally stop by your office to pick your brain every once in a while, I never knew that programming would interest me so much! Take care of yourself and have a happy new year.
> Student name

VII. CONCLUDING REMARKS

There was a brief experiment midway through the most recent semester using Captivate to record the classes as a back-up to Wimba. And as result of assessment comments, the entire class has now been fully captured using Captivate (although the recordings in this first pass were experimental and not the best: http://attila.sdsu.edu/~impellus/courses/me203) In fact, students now prefer the captivate sessions to all other methods and the class delivery is currently under re-design once again.

A. Toward the Future: Impact on other classes

But can this design and implementation impact other classes in mechanical engineering? Yes, provided that an aspect of SSH is addressed.

The difficulty with SSH is that it is text-based: a *graphics* application running on a remote server cannot be displayed on the local "student" client. X-Win, however, enables graphics codes to be displayed remotely. X-Win32 is StarNet's latest X terminal application for Windows and Mac clients. X-Win32 allows Windows users to connect to Linux/Unix servers on a network and run the applications from those servers on their Windows desktop. This Fall (2008), Xwin-32 will be deployed in the class so that students can instantiate a simple graphics code.

This has further implications for distance learning in engineering, for it will enable universities to exit the computer infrastructure support business. Universities can load visualization software on a server, enable the students to download Xwin, and run the codes on the server, yet display them at home. Advanced visualization tools can now be fully deployed in all aspects of a curriculum, while still using distance learning tools. For once, universities can exit the hardware infrastructure support business and go so far as to invert pedagogies by exploiting 3D simulations with distance learning in all aspects of a curriculum.

B. New Lines of Research

While the next round of emerging data demonstrates that student learning is improved (paper pending), the author has not yet categorized how this re-design mitigates each cognitive load *individually* (but the author welcomes collaboration). There are issues that can be considered, however in such future research. Paas and van Merriënboer [19] suggested that 'worked' examples inspire learning; and this has proven to be a successful line of approach used in this re-design. Thus the author suggests the need for research into how the demonstration of expertise can ease the various cognitive loads on learning (see Felder [21]). Wimba enabled the instructor to write, debug and run codes on the fly, and during e-office hours. And there were many positive responses from students in how it helped them to watch—from their own home—the instructor do this.

In this class, the focus was on using the language to present algorithms; not teaching rigorous coding styles—that level of coding expertise development can be left for upper division courses, or for students who wish to continue working at the intersection of coding and mechanical engineering. Currently many programming courses deploy a pedagogy that require students write programs in this order: 1) the problem; 2) the algorithm; 3) the flow-chart; 4) the code; and, finally 5) debug. Yet, this author's experience—when he had been using this approach—is that some students simply start writing the code—they hack; and these students tend to do better in the class. In this class, the focus was placed on coding for the sake of algorithm; and pending results indicate better performance by non-traditional students of mechanical engineering when students are encouraged to experiment with coding (hack). Perhaps CLT can be a vehicle to assess how students learn, on their terms, without the baggage of, say, flowcharts, that students rarely use, represent an additional load, and which might impede learning.

Today's students have matured in a visual world—and a 3D world at that. In addition, they process information in parallel: they search the web, download music and communicate on cell phones, simultaneously. Yet we still educate in uninterrupted fifty minute sessions on a 2D white board or chalk board. In light of this demonstrated success at using CLT to enhance learning, the author suggests two more lines of research: 1) investigating CLT while a learner is challenged by multiple streams of information; 2) investigating CLT with a specific focus on 3D information.

ACKNOWLEDGEMENTS

This work was supported, in part, by the National Science Foundation *Multi-Phase Mechanics*, and the Department of Education, FIPSE, *Dissemination of a New Mechanical Engineering Curriculum.* The author thanks Suzanne Aurilio and SDSU's pICT for providing assistance with the educational/pedagogical issues.

REFERENCES

[1] Atkins, D.E., K.K. Droegemeier, S.I. Feldman, H. Garcia-Molina, M.L. Klein, D.G. Messerschmitt, P. Messina, J.P. Ostriker, and M.H. Wright. 2003. Revolutionizing science and engineering through cyberinfrastructure: Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure. National Science Foundation Office of Cyberinfrastructure, http://www.nsf.gov/od/oci/reports/toc.jsp.

[2] Bonar, J., and E. Soloway. 1985. Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human-Computer Interaction* 1:133–161.

[3] 1993 D.W. Johnson, R.T. Johnson, and K.A. Smith, Active Learning: Cooperation in the College Classroom, 2nd edition, Interaction Book Co., Edina, MN, 1999.

[4] Sweller, J. 1994. Cognitive load theory, learning difficulty and instructional design. *Learning and Instruction* 4: 295–312.

[5] Kalyuga, S., Chandler, P., and Sweller, J. (1998). Levels of expertise and instructional design. *Human Factors* 40, 1–17.

[6] Pollock, E. Chandler, P., and Sweller, J. (2002). Assimilating complex information. *Learning and Instruction. 12*(1), 61–86.

[7] Sweller, J., J. Van Merriënboer. and F. Paas. 1998. Cognitive architecture and instructional design. *Educational Psychology Review* 10(3): 251–296.

[8] Mayer, R. 2001. *Multimedia learning*. Cambridge: Cambridge University Press.

[9] Paas, F.G.W.C., and van Merrienboer, J.J.G. (1993). The efficiency of instructional conditions: An approach to combine mental-effort and performance measures. *Human Factors, 35*(4), 737–743.

[10] Marcus, N., M. Cooper, and J. Sweller. 1996. Understand instructions. *Journal of Educational Psychology* 88: 49–63.

[11] Van Merriënboer, J., and H. Krammer. 1987. Instructional strategies and tactics for the design of introductory computer programming courses in high school. *Instructional Science* 16: 251–285.

[12] McKeachie, W.J. 1999. *Teaching tips: Strategies, research, and theory for college and university teachers*, 10th *ed.* Boston: Houghton Mifflin.

[13] Wankat, P., and F. Oreovicz. 1993. *Teaching engineering*. New York: McGraw-Hill, Inc.

[14] Felder, R.M., and L.K. Silverman. 1988. Learning and teaching styles in engineering education. *Journal of Engineering Education* (April) 78(7): 674–681.

[15] Van Merriënboer, J., P. Kirschner, and L. Kester. 2003. Taking a load off a learner's mind: Instructional design for complex learning. *Educational Psychologist* 38(1): 5–13.

[16] Renkl, A., and R. Atkinson. 2003. Structure the transition from example study to problem solving in cognitive skill acquisition: A cognitive load perspective. *Educational Psychologist* 38(1): 15–22.

[17] Renkl, A., R. Atkinson, and C. Grosse. 2004. How fading worked solution steps works—A cognitive load perspective. *Instructional Science*: 59–82

[18] Van Gerven, P., F. Paas, J. Van Merriënboer, and H. Schmidt. 2002. Cognitive load theory and aging: Effects of worked examples on training efficiency. *Learning and Instruction* 12: 87–105.

[19] Van Merriënboer, J., and F. Paas. 1989. Automation and schema acquisition in learning elementary programming: Implications for the design of practice. *Computers in Human Behavior* 6: 273–289.

[20] Van Merriënboer, J., J. Schuurman, M. de Croock, and F. Paas. 2002. Redirecting learners' attention during training: Effects on cognitive load, transfer test performance, and training efficiency. *Learning and Instruction* 12: 11–37.

[21] Van Merriënboer, J. 1990. Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of Educational Computing Research* 6(3): 265.

[22] Felden, David, F., J. 2006. The Implications of Research on Expertise for Curriculum and Pedagogy. *Educational Psychology Review* 2006.

AUTHOR

Thomas Impelluso received his BA in Liberal Arts from Columbia University. This was followed by two MS degrees in Civil Engineering and Biomechanics, also from Columbia. He received his

doctorate in Computational Mechanics from the University of California, San Diego. Following this, he worked for three years in the software industry, writing code for seismic data acquisition, visualization, and analysis. He then commenced post-doctoral studies at UCSD, wherein he secured grants in physics-based virtual reality. He is now a tenured associate professor at San Diego State University, revisiting and researching human bone remodeling algorithms and muscle models using advanced tools of the cyberinfrastructure. He has created a curriculum in which students learn mechanics not by using commercial simulation software, but by creating their own. His interests include opera, sociology, and philosophy. He is currently enjoying teaching his two young children how to ride bicycles.