Distance Learning and Cognitive Load Theory to Enhance Computer Programming for Mechanical Engineers: Qualitative Assessment

Thomas J. Impelluso, Ph.D.
Associate Professor
Mechanical Engineering
San Diego State University

ABSTRACT

A computer programming class for students of mechanical engineering was re-designed with regard to both content and delivery. The goal was to improve student learning attitudes. Cognitive Load Theory (CLT) was used to re-design the content; on-line technologies were used to re-design the delivery. Since the targeted students were not computer scientists, the course was re-designed to focus on computer programming examples used in mechanical engineering. Scaffolding was used to integrate syntax elements with each other, algorithms with each other, and, the algorithm to the syntax. The effort was assessed using student attitudinal data. The effort confirmed the utility of CLT in course design, and it demonstrated that hybrid/distance learning is not merely a tool of convenience, but one, which, used purposefully, inspires students to learn.

Introduction

Cognitive Load Theory (CLT) provides guidelines to present information in a manner that encourages learning and optimizes intellectual performance [1]. 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 itself. Interestingly, this is resonant with the challenge of learning to program a computer (learners must master operating systems and the syntax) for students not in the computer science major. CLT can mitigate challenges in such cases when learning loads are high. CLT was used to re-design a computer programming class for mechanical engineers at San Diego State University.

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 (which, in the case of programming, encompass text editing, operating systems and compilers). CLT posits that there are three basic types of cognitive loads placed on a learner:

- “Intrinsic cognitive load” was first described in 1991 [2] 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 [3] 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 [4]. 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 [4] that when courses are redesigned with due respect paid to the interaction of cognitive loads, learning is improved. 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.

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

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 presents this author's view of the various learning loads experienced in computer programming. The intrinsic learning load is high in computer programming. Thus, if one also employs methods that add an extraneous load (such as complex compiler interfaces)), it is very likely that there will be little capacity left for germane load that might be used to motivate students of mechanical engineering to learn programming; and the ensuing overload would then hinder their learning.

Furthermore, it has been shown that complete, thorough, and fully commented programming examples provide greater motivation for novices than simply working out problems from scratch [5][6]. Although this may seem counterintuitive, tests have demonstrated that studying complete examples facilitates learning more than actually solving the equivalent problems [7]. Additionally, in many cases, a variation of worked examples balanced with assignments was used [8]. 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 providing completely worked examples, this serves to decrease extraneous cognitive load [9].

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 [10].

2. Course prior to re-design

2.1 Course content

The course under discussion is SDSU: ME203 – Programming. In this class the C programming language is taught in a UNIX environment. The course presents 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.

The focus of the class was on the syntax of the C language. Advanced syntax techniques such as 'data structures' were taught. Secondary attention was paid to the Gauss Reduction method and various algorithms to multiply matrices. Mechanical engineering coding examples were not integrated into the course; they were presented without instructional design forethought.

2.2 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 weaknesses. 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. Third, 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. Thus, course re-design was initiated.

3. Course after re-design

3.1 Course content

Two levels of course material were scaffolded 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). The purpose of the scaffolding was to avoid previous student criticisms of seeing no purpose to learning programming. The scaffolding more tightly connected the syntax to the algorithm and gave purpose to the class for mechanical engineers.

3.1.1 Vertical Scaffolding

The left column of Table 2 indicates the *syntax structures* that were discussed in the class. Complete and commented code syntax examples were scaffolded on the skeleton of preceding ones: loops were discussed in the context of logical structures, and arrays were discussed in the context of loops. The right column indicates the *mathematical algorithms* that were discussed in class: Complete and commented algorithms were scaffolded on previous ones. 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.

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

Algorithm	**Syntax**
Temperature Conversion	Data types and logic
Bisection Method	Logic & loop formality
Newton's Method	Logic & loop formality
Numerical Integration	Logic & loop formality
Repeat of all previous algorithms	Input/Output
Matrix-Vector Multiplication	Arrays
Gauss Reduction	Arrays, files
Gauss reduction with functions	Arrays, files functions
Newton Raphson Method	Arrays, files, functions, memory

3.1.2 Horizontal Scaffolding

The driving focus of the content in this class was the algorithm rather than the syntax. Thus, this inverts the way programming has traditionally been used in which syntax rules are presented and are the focus – as often happens when programming classes are farmed out to computer science departments. Furthermore, there were no coding examples of sorting, alphabetizing, or interest rate problems that plague introductory computer programming courses for mechanical engineers.

Table 2 indicated 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 also 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 instructional goal was to challenge the students to read codes as if they were a new language. It was 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.

Next, the course delved into a series of code examples involving matrix manipulations. And finally it moved on to the two core concepts of the course. In fact, students were often reminded that these two algorithms were the core algorithms in mechanics. 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.

3.1.3 Course Delivery

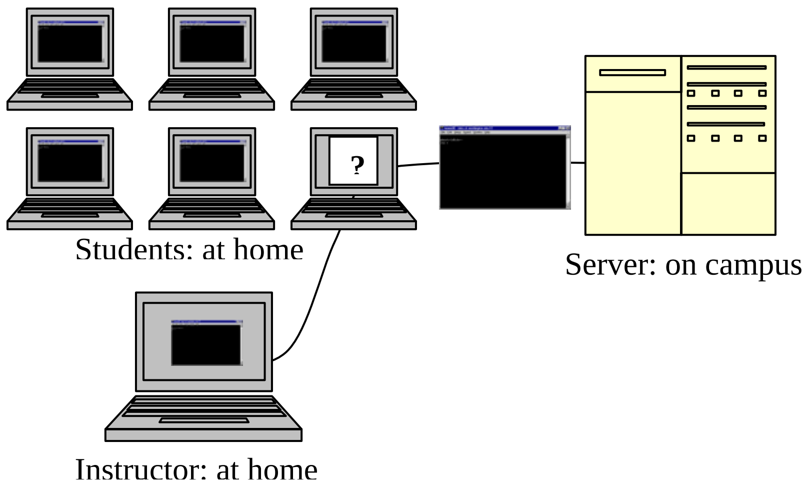

Figure 1. Student-instructor connections for online class sessions.

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 focused on the interplay of intrinsic and germane learning loads.

The extraneous learning load was obviated by use of on-line instructional technologies. However, this was not a passive use in which students simply observed lectures. Application sharing technology was used – the instructor took control of student laptops as if working with the student, side by side, while also demonstrating the effort to the rest of the class. 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.

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 would 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 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.

Special note is made of the fact that all the students in the class used the same operating system on which to compile and run their code examples. This minimized a great deal of confusion for the students: they used an SSH tool to connect to the common server on which they all studied and learned. This summary warrants 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 computational 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.

4. Attitudinal Assessment

This course was subjected to qualitative student reviews. 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. Two assessment periods are provided. Fall, 2006 assessments were for the class before there was any re-design. Spring, 2008 represents the first semester in which the instructor gained facility with the instructional technology and the modified curriculum.

Students were asked to respond to this question: *What are your comments/suggestions for the course?* Naturally, this paper cannot provide all responses, so random responses were selected from the fall, 2006 semester and are presented in Table 3. Responses from the spring, 2008 semester are presented in Table 4

Table 3
Fall, 2006 Responses to Qualitative Questions Concerning the Course

hard to understand
For a course that is comparable to learning a foreign language in a matter of 15 weeks, not only was it hard to adapt to but the professor made it even more difficult by making it hard to ask questions.
Try to make subject more interesting
\it seems to be an unnecessary class for mechanical engineers

The previous responses were mostly negative and the few positive responses were apathetic. After the re-design, the negative responses became more active in suggesting improvements while the percentage of positive responses increased and became more vigorous.

Table 4
Spring, 2008 Responses to Qualitative Questions Concerning the Course

More examples of real world application, shorter classes, keep this direction going!
i felt that this course didn't teach me how to write programs only a few key algorithms. I feel that the course should be more focused on the coding and not the algorithms
NO MORE ONLINE CLASSES, I PAY MY TUITION TO LEARN FROM A HUMAN NOT A COMPUTER!!!!!!!!!!!!!
This is an excellent introductory course in programming for any non computer engineering student. I think the structure of the exams should be changed; I think the exams should be more like project-oriented.
This is a great course overall, but there has to be a class before this at least to show us how programming works; to teach us the basics of programming and getting familiar with it. This course just jumps ahead and you can fall back very easily.
go a little slower in the beginning, since that's the most important part

Clearly, this re-design has excited the students. Enrollment has increased. Students took a more active part in their learning. They offer advice on how to improve the class, and, this, in turn, makes it easier to continually improve the course through consistent re-design.

Students were then asked to respond to this question: *What are your comments, positive or negative, on the instructor's teaching?* Again, random responses were selected from the fall 2006 semester and are presented in Table 5. Responses from the spring, 2008 semester are presented in Table 6.

Table 5
Fall, 2006 Responses to Qualitative Questions Concerning the Instructor

Dr. I came off as the least approachable professor i ever had causing communication problems which led to a tear in his teacher/student relationship.
Moves much to fast through the course material. Maybe it's because there is too much material for the course, but it needs to be slowed down.
I wish he would explain what he is doing a little more clearly. He can breeze right through an area expecting us to know what he is talking about.
Answer peoples questions with out making them think that they are stupid. Usually people who are in engineering are not stupid so treat them with respect.
Teacher was not able to stimulate interest. Unable to fully/clearly explain the material.
negitive, everyting i learned was from the internet

Table 6
Spring, 2008 Responses to Qualitative Questions Concerning the Instructor

He moves a little fast but if you ask him to slow down he will. You can ask any question no matte how dumb and he will answer it over and over again until you get it. Very helpful
He goes out of his way to keep students active in the class which is rare in a teacher.
Very helpful teacher and clearly taught the material
Instructor goes above and beyond the call of duty.
The instructor was always available for the students and was very flexible about helping out students.
Dr. Impelluso 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 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.
Dr. Impelluso extends himself above and beyond to help his students. However, his lectures seem to change focus abruptly which leaves the student lost.
The vocabulary for the course makes it difficult to ask questions. It is new so I didn't know how to formulate questions and Impelluso is not aware or not sensitive enough to this. I really like the fast response time in email. I really appreciated the extra time and effort and caring about students that he put into this class. Impelluso stressed too much on how students were doing and made himself too avaiable to students, resulting in him stressing and getting upset. I suggest setting up office hours but holding them online or by individual appointment onlyand make help more the students responsibility. good teacher. I love it when teahcers care, it helps and motivates, so thank you, overall I give you an above average for teaching, most don't care, especially those with tenure,is that how you spell it? Thanks!!

5. Discussion

5.1 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 ninety 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 other classrooms.

5.2 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.

5.3 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.

6. Conclusions

Distance technologies have changed since they first emerged – they are no longer passive technologies which simply enable download of content; they now enable interactive learning. This course re-design made have use of these latest technologies, specifically including desktop sharing. The redesign was guided by CLT. CLT has been demonstrated to be a useful schema for re-designing a course. Three types of scaffolding have been deployed: vertical and horizontal (scaffolding intrinsic learning loads) and temporal (scaffolding germane learning loads). Distance technologies were used to minimize extraneous learning loads.

The re-design resulted in a tremendous cost savings to the department. Student perceptions and motivations have dramatically increased. It is noted that that author chose to use increasingly more difficult exams to ensure equity in grading and to prevent grade inflation. In future work, the author will shortly revert back to exams from years past and report on the results. It is expected, however, that the majority of the students will excel.

Can this design and implementation impact other classes in mechanical engineering; specifically, in courses which teach programming interfaces such Computer Aided Design software? 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 spring (2009), 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.

REFERENCES

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

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

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

BIOGRAPHICAL SKETCH

Dr. 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.

FUNDING AND SUPPORT

(This work was supported by two grants.)

Multi-Phase Mechanics
Agency: NSF
Amount: $124,550
Duration: 11/05—11/06 (plus one year no-cost extension)
Summary: Extend a platform for solution of multi-phase problems by incorporating non-linear interaction and contact with extension year duties of curricular deployment.

Dissemination of a New Mechanical Engineering Curriculum
Agency: FIPSE
Amount: $370,000
Duration: 9/02—9/05 (plus two year no-cost extension)
Summary: This project funded the evolution and dissemination of a method to teach mechanical engineering at its intersection with computer science.

ACKNOWLEGEMENTS

The author thanks Suzanne Aurilio and SDSU's pICT for providing valuable assistance with this course re-design.