Context as Support for Learning Computer Organization

ALLISON ELLIOTT TEW, BRIAN DORN, WILLIAM D. LEAHY, JR.,
and MARK GUZDIAL
Georgia Institute of Technology

The ubiquity of personal computational devices in the lives of today's students presents a meaningful context for courses in computer organization beyond the general-purpose or imaginary processors routinely used. This article presents results of a comparative study examining student performance in a conventional organization course and in one that has been contextualized using a personal gaming platform as the pedagogical architecture. We find minimal differences in student learning but significant motivation and engagement gains for those in the contextualized course.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*; *Curriculum*; C.1.0 [**Processor Architectures**]: General

General Terms: Design; Measurement

Additional Key Words and Phrases: computer organization, context, course design

ACM Reference Format:
Tew, A. E, Dorn, B., Leahy, Jr., W. D., and Guzdial M. 2008. Context as support for learning computer organization. ACM J. Educ. Resour. Comput. 8, 3, Article 8 (October 2008), 18 pages. DOI = 10.1145/1404935.1404937. http://doi.acm.org/10.1145/1404935.1404937.

1. INTRODUCTION

There are two common approaches to undergraduate introductory computer organization courses. The first of these, adopted by such authors as Patterson and Hennessy [2007], introduces topics by examining an actual, general-purpose architecture (e.g., MIPS) at progressively deeper levels. Critics of this approach are often concerned with the inherent complexity in examining a real-world processor. An alternative approach advocates using an imaginary machine (e.g., LC-3) which has been simplified for pedagogical purposes [Knuth 2005; Patt and Patel 2003]. Such processors are imaginary in the sense

Author's address: A. E. Tew, College of Computing, Georgia Institute of Technology, 85 5th Street NW, Atlanta, GA 30332-0760; email: allison@cc.gatech.edu.

that they have been crafted solely for educational uses and would not normally be seen outside of the classroom.

In either case, actual experience with the processor has been recognized as a crucial component in student learning about computer organization. This is often achieved by using a processor simulator running on a standard desktop computer. However, other educators have also argued for using physical embedded hardware to teach introductory computer organization, either using a commercial processor and system board (e.g., Bonato et al. [2004]) or implementing a custom imaginary processor using FPGA technology (e.g., Pearson et al. [2002]).

The preponderance of powerful and inexpensive personal computation devices like MP3 players, cell phones, and gaming platforms enables a new twist on teaching computer organization. Many of these technologies make use of embedded processors that are sufficiently complex to serve as examples for a first computer organization course. Though these embedded processors present some of the same issues of complexity as their general-purpose counterparts, they may also provide a tangible context for learning that is absent in the other common approaches.

By context, we mean an application area for the content being learned that is familiar to and valued by the students in the course. A contextualized course uses the application area in lecture descriptions, examples, homework assignments, and other areas where it is useful to relate the course content to any application area. While one might imagine a number of possible contexts for a computer organization course, we have chosen to use a personal computation device as the context here. In this case, students are familiar with personal computation devices and many own one or more. The ubiquity and everyday value of these devices brings a natural authenticity to course content.

Previous studies have shown value in using a familiar context for teaching introductory computing courses. At our own university we have observed that the use of teaching within a context provides a means to attract and retain students—particularly those from underrepresented populations [Forte and Guzdial 2005]—and as a motivator for students to do work beyond that which is required [Forte and Guzdial 2004; Yarosh and Guzdial 2007]. Furthermore, we have noted that these effects persist when the contextualized approach is adopted for courses targeting both majors and nonmajors, as well as for institutions other than our own [Tew et al. 2005].

While the contextualized approach used at Georgia Tech is centered around digital media and its communicative value [Guzdial 2005], other institutions are exploring the value of a variety of contexts throughout the curriculum. Some approaches maintain a traditional computer science (CS) focus but incorporate a domain-specific theme, such as science [Dodds et al. 2007] or engineering [Smith 2007], that motivates the homework and projects. Others are introducing CS concepts throughout the general education curriculum as a means to increase computing literacy while broadening perceptions of the discipline. For example, researchers at Duke University are building modules related to social network analysis to be used in high school math and statistics courses [Alt et al. 2006]. There are also “fully contextualized” approaches

that completely reenvision the traditional content of the introductory computing course. In such courses, an inherently motivating context for students is chosen, and all CS content is then driven by that context where thematic needs play a critical role in determining what CS is covered. The Institute for Personal Robots in Education is developing such a curriculum using a robotics context [Blank 2006], and Bryn Mawr College is further refining this effort to also incorporate a gaming theme [Xu et al. 2008].

The value of a context lies in its ability to make the abstract content areas of computer science more concrete for the student. As academics, we tend to value the abstract, that which generalizes over many instances and application areas. However, there are advantages to more concrete styles of thinking. The term concrete describes how a person relates to an idea or object [Wilensky 1991]. An item in the physical world is necessarily concrete because one can relate to it via several senses, through experiences with the item, and with memories of the item. Ideas become concrete as we enhance our mental connections and associations with them. Papert [1991] argues that concrete ideas are more easily learned than abstract ones.

Additional research in cognitive science supports this claim. Anderson and Bower [1973] have demonstrated the associative nature of human memory. Kolodner [1997] further argues that people more easily recall specific memories to which they have more associations. Concrete ideas are more likely to be remembered than ideas which are brittle and associated with no context other than the class in which it was learned [Bruer 1993]. Studies of how people transfer knowledge suggest those ideas learned well, with more associations to those ideas, are the ones most likely to transfer [Bransford et al. 2000]. Therefore, our theoretical foundations for contextualized computing education suggest that the context makes the concepts in computer science more concrete which should make them more easily learned and, later, more easily transferred.

The growing interest in contextualized computing education is also supported by literature which suggests that students perceive traditional computing curricula as asocial, overly abstract, and irrelevant to their daily lives [AAUW 2000; Margolis and Fisher 2002]. Contextualized courses are specifically designed to leverage concrete experiences and make connections to personally meaningful domains. Thus, we believe that a successful context must provide a sense of "thick authenticity" [Shaffer and Resnick 1999]—that is, it should pique personal interests while also having obvious ties to the real world.

Our success with context in previous courses has led us to explore its role beyond the introductory sequence. Georgia Tech offers two courses to introduce students to fundamental concepts of computer organization and C programming tied to an underlying architecture. One course is taught with a conventional approach utilizing Patt and Patel's imaginary LC-3 computer. The other employs the architecture of the Nintendo® Game Boy® Advance[1] personal

[1]Nintendo, Game Boy, and Game Boy Advance are registered trademarks of Nintendo Co., Ltd.

gaming system as the context for studying these issues. These courses provided the setting for the study described in this article.

Introducing a context in the domain of computer organization brings up a number of issues to consider. How can context be used to leverage learning opportunities with a real-world architecture? Further, in contextualization of this course content, how are student learning outcomes affected? A somewhat separate, but related issue is whether previous findings about increased student motivation in contextualized CS1/2 courses are reproducible in courses beyond the introductory sequence. We aimed to start exploring these issues by posing the following specific research questions:

RQ1. How is student performance on computer organization learning outcomes impacted by course contextualization?

RQ2. How does course context influence the way students express their knowledge of computer organization concepts?

RQ3. Does course context impact student engagement and motivation in the course?

The remainder of this article proceeds as follows: Section 2 elaborates on the two courses studied. Details of the study design are presented in Section 3. Results and discussion of the performance assessment appear in Sections 4 and 5, respectively. Section 6 explores student demographics and motivation as potential factors. Finally, we conclude with commentary on implications and limitations of this study.

2. THE COURSES

The two courses studied here are both sophomore-level courses that serve as primers to computer organization. The third author was the instructor of both of these courses during the study. One course, referred to as the "conventional course," is an introduction to basic computer hardware, machine language, assembly language, and C programming. It is a required course for computer science majors, and they make up the majority of its student population. The other course, denoted as the "contextualized course," is a class in media device architecture exploring the interface between hardware and software in media devices through machine-level programming in C. The primary audience for the contextualized course is made of those majoring in computational media, a joint degree between liberal arts and computing.

While on the surface they may appear to be disparate courses, they do have many similarities. In the subsections to follow each course is explained in more detail. Section 3.1 further elaborates on the shared learning outcomes that served as the content under investigation in this study.

2.1 Conventional

The overall objectives of the conventional course are for students to understand how a computer works at a basic level, including assembly programming, and to get a fairly comprehensive introduction to the C language in the context of the hardware upon which the program runs. The idea is to start with

transistors and show the development of logic gates, basic digital circuits, state machines, and a workable data path. Building on that structure, instructions, instruction set architectures, assembly language programming, and finally, C programming, are presented.

The digital electronics and assembly language portions of the course take the first half of the semester while the C programming portion takes the second half. More specifically, the course syllabus is as follows:

Weeks 1–3. A discussion of datatypes followed by the basics of digital logic, making use of the digital logic simulator LogicWorks™ [2].

Weeks 4–6. An introduction to the von Neumann machine and the datapath of an imaginary computer, the LC-3 Patt and Patel [2003]. Its instruction set architecture and assembly language are covered.

Weeks 7–8. Traps, subroutines, activation stacks, recursion, and interrupts with students writing assembly programs for the LC-3 emulator.

Weeks 9–14. A thorough exposition of the C language including function pointers, dynamic allocation, and dynamic data structures. Students become familiar with the string library, write a linked-list application, and complete a final project on memory allocation.

2.2 Contextualized

The contextualized course uses what might be described as just-in-time teaching. Rather than attempt to present material in a well-ordered, bottom-up approach we begin with a C program that lights a pixel on the screen of the Game Boy. We then explain hardware and language details as necessary. For example, pointers, memory, and bit-wise operations are needed to successfully light the pixel. The semester progresses in this fashion as we introduce successively more complex programs and incorporate the various hardware features of the Game Boy: bit-mapped video, double buffering, tiles and sprites, DMA, interrupts, etc. Our overall goal in this course is to illustrate the implications of hardware design on program development. The syllabus shown below details the approach used:

Weeks 1–2. An introduction to the Game Boy Advance console, by explaining how to light a single pixel. An overview of a basic machine model, including a processor, memory, and input/output, is provided.

Weeks 3–8. General low-level C programming, applied in a Game Boy context of color, video, and text displays. A more sophisticated machine model is introduced to explore advanced C concepts such as memory-mapped I/O and DMA.

Weeks 9–13. A discussion of data and its associated representation in memory by using standard video game programming constructs, such as tiles and sprites. Other game specific topics, such as timers, are addressed.

[2]LogicWorks is a trademark of Capilano Computing Systems, Ltd.

Week 14–15. A final creative, open-ended project requiring students to apply their low-level C knowledge to the Game Boy machine model. In addition to the game they build, students are responsible for explaining how Game Boy hardware features are utilized in their project.

3. THE STUDY

3.1 Instrument Development

Recognizing that the two courses under consideration did not aim to cover identical course material, it was necessary to determine the set of shared content elements derived from common learning objectives. We noted overlap on general computer organization topics as well as low-level C programming objectives. These elements formed the content which we used to assess student performance:

—arrays
—basic machine models
—bitwise operations
—character strings
—datatypes and their hardware representations
—memory and pointers
—number base conversions

To investigate research questions 1 and 2, we developed a short instrument based on the topics above. This six-question assessment was given to students in both the conventional and contextualized courses. Four of the questions were conceptual questions designed to address individual topics while the other questions combined topics in exercises that asked students to solve a programming question by writing C code. Verbatim prompts from the content instrument appear in Appendix A. We will present each question in more detail alongside the results in Section 4.

Face validity [Anastasi and Urbina 1997; Gay and Airasian 2000] was addressed by two primary activities. The instructor reviewed the instrument to ensure that the content was comparable to exams and quizzes given in both courses. A group of colleagues also examined the questions to confirm that they were reasonable measures of our intended topics. We did not, however, address content or criterion validity due to the preliminary nature of this study.

In addition to the content instrument, we also prepared a brief survey to allow us to explore issues of student engagement and motivation (RQ3). We gathered basic demographic data (e.g., gender, major, class standing) and attitudinal indicators (e.g., enjoyment, difficulty) using this survey distributed with the content instrument.

3.2 Data Collection

The survey and content instrument were given to students of both courses in the week immediately preceding Spring 2007 final exams. Data collection

occurred during a regularly scheduled, one-hour course period and was overseen by research personnel not associated with either course. Student participation was voluntary; however, members of both classes had been previously encouraged by their instructor to participate as a means to review course material. The participation rate for the conventional course was 54.5% (36 of 66) and 43.6% (17 of 39) for the contextualized course.

3.3 Data Coding

Following data collection, a rubric was developed in order to determine student performance. The rubric was created one question at a time in a data-driven manner. We examined responses for both their technical correctness and their qualitative nature. Degrees of correctness were derived from model solutions; in most cases this measure took the form: no response, incorrect, partially correct, correct. How a response was classified on this continuum depended on its inclusion of the essential features or substeps present in the model solution (or an obvious attempt to do so). The rubric for each question also included a means to characterize the qualitative nature of the response. The possible characterizations were generated inductively by examining a majority of the responses and noting unique properties. For clarity, we discuss the coding rubric in more detail along with student performance results for each question individually (see Section 4).

Responses were coded using the rubric by a panel of nine researchers. Working in groups of two to three, we coded individual questions. For example, one group coded all responses to question two while another group coded question six. Coding proceeded in a collaborative fashion within each small group to maintain consistency in the rubric's application. In the event that a consensus could not be reached for the coding of a particular response, the issue was escalated to the entire panel for resolution.

4. ASSESSMENT RESULTS

We present the quantitative results for each question on the assessment in this section. For each question we provide an overview of its content, an explanation of the coding rubric, and the student performance data. A discussion and our interpretation of these results follows in Section 5.

For this analysis, our research questions necessitate a nuanced view of student solutions beyond a simple binary correct/incorrect distinction. We are concerned with degrees of correctness as a measure of student knowledge development and variation in the ways similar answers are expressed. Accordingly, assessment results are analyzed using the full spectrum of correctness. Our statistical tests take into account these performance distributions for the two courses, rather than simply comparing the number of accurate responses. Statistical significance is reported using the 0.05 level, but due to the exploratory nature of the study, we also report statistics for $0.05 < \alpha < 0.10$ as marginally significant.

Table I. Q1–Number of Subparts Correct.

	3	2	1	0	No Ans
Conventional	19 52.8 %	13 36.1%	3 8.3%	0 0.0%	1 2.8%
Contextualized	4 23.5%	6 35.3%	2 11.8%	3 17.6%	2 11.8%

4.1 Question 1

This question was designed to assess students' knowledge of converting numbers into different representations, a common introductory topic in computer organization courses. The question was divided into three parts: part A asked students to convert a number from base 10 into base 16, a common short hand for binary; part B asked students to convert between base 16 and base 8, two nondecimal bases; part C asked students to convert from base 8 back to base 10. The coding rubric measured whether the students got the correct answer for each subpart of the question or whether no answer was given. We then aggregated the marks for each subpart to give a score, 0–3, to each student on this question.

Students from the conventional course were more likely to attain a perfect score (52.8% vs. 23.5%). Many (41.2%) of their contextualized counterparts struggled, failing to score more than one point. This data is presented in more detail in Table I. The Fisher exact test[3] yielded a statistically significant difference ($p = 0.025$) in the score distributions of the two courses. However, we noted no significant differences on the comparison of performance on the individual subquestions (that is, neither course outperformed the other on any particular base conversion).

4.2 Question 2

Students were asked to "Describe the steps used by a computer to execute the C language statement `c = a + 7`." The goal was to explore the level of abstraction used by students when describing code execution. A correct answer was indicated by a description, in some form, of the following three steps: retrieve `a`, add `7`, store the sum in `c`. The second part of the rubric characterized how they described the process of evaluation: Was memory referred to as addresses or registers? Was memory referred to as variables at the source code level? Did their response refer to a stack-based evaluation scheme? Did their response refer to the need for compilation prior to the execution of the C code statement?

Students in the conventional and contextualized course populations both performed well with 86.1% and 76.5% (respectively) answering the question correctly. There was no statistically significant difference between the courses and the correctness of their answers. However, there was a statistically significant difference in the nature of their answer. Students in the conventional class were more likely to refer to memory at the address/register level

[3]Fisher's exact test is used in place of χ^2 when expected cell frequencies are less than 5 [Fleiss et al. 2003]. All p-values reported are computed using Fisher's exact test.

Table II. Q3–Correctness.

	Correct	Partial	Incorrect	No Ans
Conventional	3 8.3 %	15 41.7%	11 30.6%	7 19.4%
Contextualized	0 0.0%	2 11.8%	8 47.1%	7 41.2%

($p = 0.006$), while students in the contextualized class were more likely to refer to memory as high-level language variables ($p = 0.035$).

4.3 Question 3

To investigate students' understanding of a basic organizational model for a processor (e.g., a von Neumann model), students were asked to draw a block diagram of a typical processor's components. The coding rubric measured both correctness of the model as well as a characterization of the model. Students were expected to represent five distinct entities in their model: memory, a control unit, an execution unit, input/output, and a bus connecting the components. Correctness of student models were placed into categories along a continuum. Correct models included all five entities and used appropriate terminology to describe the components. Models were categorized as partially correct when they were missing at most one component, but had to include the memory and execution unit components. All other models were deemed incorrect. Responses were also characterized by whether they decomposed the five basic components into subcomponents (e.g., representing the control unit as a finite state machine with additional components such as a program counter and/or an instruction register.) Models that included domain-specific components on the bus, such as video or sound, were also noted.

Students in the conventional course performed better on this question, although the difference was noted with marginal statistical significance (see Table II, $p = 0.054$). Looking deeper, a majority of students (68.9%) in the conventional population included a control unit component in their processor model, while only 10.0% of the students in the contextualized population did so. This was a statistically significant difference between the courses ($p = 0.002$). On the other hand, we noted that the contextualized course students were more likely to incorporate domain-specific components (e.g., a graphics processor) into their diagrams ($p = 0.013$) while often omitting the standard components.

4.4 Question 4

To explore models of memory, data, and pointers in an open ended-format, students were asked to draw a diagram representing integer and pointer variables in memory after lines of C code were executed. In addition to correctness, the responses were characterized along three different dimensions: the notation of the diagram or representation, how pointer values were represented, and whether pointer and integer data were treated uniformly. Representations noted in the data set included text, box and arrow diagrams, and activation

stack. Pointer values were represented by the address-of operator (i.e., `&pa`), arrows, an arbitrary memory location, or not included in the diagram.

A majority of the students in the conventional and contextualized courses answered the question correctly, 77.8% and 58.8% respectively. The most common notation used was a box and arrow diagram (conventional: 52.8%, contextualized: 58.8%) with pointer values being denoted with arrows (conventional: 75.0%, contextualized: 70.6%). Students in the two populations showed no statistically distinguishable differences on either their performance on this question or on the qualitative characterization of their responses.

4.5 Question 5

The final two questions were developed to explore students' familiarity with low-level programming concepts—one question was designed to be similar to problems they had seen previously in their course assignments, and the other was designed to be in a context with which they were unfamiliar. Question 5 asked students to write code that would set the color value of a specific pixel on a Game Boy screen. Color values for pixels in the Game Boy architecture model are packed two pixels per 16 bit short integer, with the low order bits specifying the color of the first pixel. In order to simplify the complexity of the prompt, the question was divided into two parts. Part A asked students to write the function `getPixelOffset` to compute the location in the video buffer where the color value is stored. Part B used this function to set the appropriate pixel to the new color value.

The coding rubric for technical correctness for part A looked for students to recognize the essential steps to the problem solution: addition of row and column variables, use of an appropriate constant from the screen resolution given in the problem statement, and division by two (as there are two pixel color values per addressable memory location). Correct responses accurately computed the offset according to the problem specifications. Partially correct responses used the row and column variables in their calculation of the offset but may have missed one of the steps in the problem solution or incorrectly computed the offset. All other responses were marked as incorrect.

Student responses were also characterized by two errors common in the data set: attempting to address the specific pixel (i.e., the correct half, rather than the addressable memory location) and attempting to return the actual value in the video buffer rather than the offset to that location.

The coding rubric for part B was focused on technical correctness and again sought to recognize the fundamental steps of the model solution. Students first had to determine which half of the short the color value was located in and then clear out the existing value, typically using a mask and bitwise operation. Color values to be stored in the higher order bits were shifted and then the color value was merged back into the original video buffer. Correct answers properly placed the new color value in the corresponding pixel location in the video buffer. Partially correct responses were required to determine which half of the short the pixel was located in and merge the color value into the original video buffer, but may have neglected to include one of the intermediate

Table III. Q5–Correctness.

		Correct	Partial	Incorrect	No Ans
Q5a	Conventional	7 19.4%	12 33.3%	10 27.8%	7 19.4%
	Contextualized	6 35.3%	6 35.3%	4 23.5%	1 5.9%
Q5a	Conventional	5 13.9%	8 22.2%	13 36.1%	10 27.8%
	Contextualized	1 5.9%	10 58.8%	3 17.6%	3 17.6%

Table IV. Q6–Correctness.

		Correct	Partial	Incorrect	No Ans
Q6	Conventional[4]	4 11.4 %	11 31.4%	15 42.9%	5 14.3%
	Contextualized	0 0.0%	3 17.6%	7 41.2%	7 41.2%

steps to correctly process the color value. Other responses were marked as incorrect.

Students in the contextualized course received higher marks for technical correctness on question 5, with over 70% of the students receiving correct or partially correct marks on part A and over 64% on part B (see Table III). The student performance differences between courses on question 5a were not statistically significant, but were marginally significant on question 5b ($p = 0.094$).

4.6 Question 6

String processing was a low-level programming topic covered in depth by the conventional course and was used as the subject for question 6. This question asked students to write a case-insensitive comparison for two null-terminated strings. The coding rubric assessed both technical correctness and a qualitative characterization of the responses. A correct solution to this problem required iterating on two non-empty strings, comparing the current character in a case-insensitive manner, dealing with strings of different lengths, and setting the appropriate return values. Partially correct responses did not account for different length strings or did not correctly set the return values. All other responses were marked as incorrect. Responses were also characterized based upon whether they resulted in side effects, most often altering the input string.

Students in the conventional course performed better on this question, with over 42% of the responses being categorized as correct or partially correct, while only 17.6% of the responses from the contextualized course were categorized this way (see Table IV). Performance differences on question 6 were not statistically significant.

[4]Q6 conventional percentages are computed with $n = 35$ due to a clerical error prior to data collection.

5. ASSESSMENT DISCUSSION

The previous section outlined the quantitative results of the performance assessment. Differences between the courses were examined on two levels: in the general nature of student answers and in a more strict analysis of response correctness. Due to the style of the question, Q2, Q3, and Q4 were the most amenable to exploring differences in expression. We noted statistically significant differences between the courses in how students approached solving Q2 and Q3. When discussing code execution in Q2, conventional students used terminology at a lower level of abstraction (e.g., registers) than their contextualized counterparts, who tended to remain at the level of the programming language (e.g., variables). In the case of abstract processor models (Q3), the contextualized group seemed to struggle to distinguish domain-specific processor components from those in generic models.

Whereas some stylistic differences between the two courses were pronounced, the interpretation of response correctness was more nuanced. There was significance (Q1) or marginal significance (Q3 and Q5b) on three of the seven performance measures[5]. However, we argue that if the data is examined more broadly, one can conclude that student performance in the two courses was quite comparable overall. First, the data did not reveal significant differences on most of the measurements. Though the conventional course students may appear to have performed better on Q2, Q4, and Q6, and the contextualized course better on Q5a, the variation in the student performance distributions for these measures was not beyond that attributable to random chance. Second, on Q1 where the performance distribution was significantly different, there were no statistically significant differences between the two courses when considering Q1's subparts individually. In other words, the observed aggregate difference could not be attributed to poor performance of one group on any particular base conversion. Lastly, even on the questions that were designed to intentionally advantage one group, we saw comparable performance. The Game Boy question (Q5) yielded no significant difference on part A and only a marginal difference on part B (with a p-value of 0.094). The conventional question (Q6) also showed no significant differences in how the two classes performed. All told, the sum of the data suggests that student performance was similar in the conventional and contextualized groups.

6. ENGAGEMENT AND MOTIVATION

In order to put the results from the previous section into perspective, it is useful to examine the nature of the study participants from the two courses. Results from the demographic survey indicated that the students in each class were comparable with respect to gender, ethnicity, and class standing. The majority of students who participated were male, Caucasian, and either in the sophomore or junior year of study. We noted no statistically significant differences in these demographic indicators.

[5]While there were six questions, question 5's subparts were analyzed individually, resulting in a total of seven performance measures.

Table V. Students with Related Previous Experience.

	C	Assembly	Architecture
Conventional	4 10.8%	6 16.2%	7 18.9%
Contextualized	2 11.8%	0 0.0%	1 5.9%

Table VI. Fun Compared to Other CS Courses.

	One of the least fun	Less fun than most	About the same	More fun than most	One of the most fun
Conventional	2 5.4%	3 8.1%	13 35.1%	13 35.1%	6 16.2%
Contextualized	0 0.0%	0 0.0%	1 5.9%	8 47.1%	8 47.1%

Beyond surface characteristics, the populations also had comparable previous knowledge of the course content. The survey asked students whether they had previous experience with C programming, assembly programming, or computer architecture. Over 80% of the participants reported no prior knowledge on any of these topics (see Table V). Again there were no statistically significant differences between the course populations on these factors.

Given the overall similarity between student populations, it is reasonable to assume that variation observed between the groups can be attributed (at least in part) to effects of the courses. Our assessment results encourage the further exploration of incorporating real-world contexts into computer organization courses. The general comparability of performance in the two courses supports the claim that students in the contextualized and conventional courses learned similar content with respect to our indicators.

These results are more striking considering the difference in the student populations. The content of a computer organization course is closely aligned with traditional notions of computing. As such, the stereotypical computer science major may be intrinsically interested in the course. In contrast, computational media majors, who are presumably attracted to the interdisciplinary combination of arts and computing, are less likely to see this course material as central to their pursuits. We recognize that these characterizations are based on our intuition; however, they are supported by discussion with faculty members of both programs. Given the likelihood of a motivation discrepancy, we conjecture that the Game Boy context played a significant role in supporting the contextualized students' ability to perform comparably on our assessment.

The affective impacts of context, specifically increased motivation and engagement, could be key indicators supporting this claim. Based on survey responses, students in the contextualized group had significantly more fun in their course than those in the conventional population ($p = 0.024$). Nearly all (94.2%) of the contextualized group rated their course as more fun than other CS courses, while only 51.3% of the conventional students did so (see Table VI). As we did not control for specific prior coursework, it is possible that one group had experienced a broader and more interesting range CS coursework, thus potentially confounding this observation. However, the demographic data on

Table VII. Work Above and Beyond Assignment Requirements.

	Never	**< 50% of the time**	**Half of the time**	**> 50% of the time**	**Always**
Conventional	11 29.7%	17 46.0%	5 13.5%	3 8.1%	1 2.7%
Contextualized	0 0.0%	9 52.9%	4 23.5%	2 11.8%	2 11.8%

class standing suggests that the majority of students in both courses had only completed a two-course introductory programming sequence.

Similarly, the contextualized course students reported doing work above and beyond assigned requirements significantly more often ($p = 0.046$). In fact, *every* student in the contextualized course reported having done extra work at some point, whereas 29.7% of the conventional course students claimed that they *never* did any additional work (see Table VII). These results are consistent with findings from our earlier studies of the positive impact of context on student engagement and motivation (c.f., Forte and Guzdial [2005]; Tew et al. [2005]). This measure is particularly relevant since increased time-on-task results in more learning [Bransford et al. 2000]. It is possible that our targeted instrument failed to detect such gains because the additional time may have been spent on topics outside of the overlapping course content investigated in this study.

7. CONCLUSION

We found two clear indications of how context influenced the way students expressed their knowledge of computer organization concepts. The terminology students used indicated that the contextualized students employed higher levels of abstraction in their responses than those in the conventional course. We also noted that the contextualized group included domain-specific components in their answers to questions that were intended to elicit generic models. These observations have clear implications for educators working in contextualized courses. Some course learning objectives may be confounded by the particular details of the context, especially where the objectives are abstract concepts. Therefore instructors should identify the crucial objectives and make their relation to the specific context explicit.

Though teaching with a contextualized approach may require more careful attention in highlighting some abstract topics, the presence of a context appears to provide considerable support for learning. A holistic analysis of the data showed that students in the two courses performed quite comparably despite statistically significant differences on a few measures. We argued that the contextualized student population in this study might have been less motivated to study computer organization at the outset of the course. Yet, these students reported greater enjoyment and an increased willingness to go above and beyond the required course work. If there was an initial motivation deficit, then the course context likely played a role in their comparable performance. Put directly, the context served to motivate the otherwise disconnected curricular material to the point that students learned things they would not

have originally considered worthwhile [Pintrich and Schunk 1996]. We speculate that context provides a link between student motivation and learning outcomes and that this connection scaffolds the learning process[6]. It is clear that affective concerns should be considered when designing new courses, and we propose further exploration of the role context plays in scaffolding learning.

These findings would be even more striking had we been able to compare two sections that differed only in the application of a familiar context. Since that was not feasible, we operationalized our research questions by identifying overlapping course material and learning objectives. Certainly we are not claiming that these students learned identical computer organization content—each course covered material most appropriate to its individual purposes. Nonetheless, this work does provide initial evidence that contextualized approaches can be used beyond the introductory computing sequence, and that they continue to offer similar advantages.

While the specific Game Boy curriculum described here may not meet the needs of some institutions, use of a personal embedded device provides an opportunity to leverage the benefits of context while also permitting an in-depth exploration of computer organization concepts that can appeal to a broad range of students. There are significant advantages to relating courses to the computing in students' everyday lives [Guzdial and Soloway 2002], and evidence is accumulating that doing so has little negative impact on learning.

APPENDIX

A. STUDY ASSESSMENT QUESTIONS

Question 1: Convert the following numbers accordingly:

Part A: 49 base 10 to base 16
Part B: 3A base 16 to base 8
Part C: 174 base 8 to base 10

Question 2: Describe the steps used by a computer to execute the C language statement `c = a + 7;`

Question 3: Draw a block diagram of a typical processors components (e.g., a Von Neumann diagram) and describe the purpose and function of each block.

Question 4: Given the C code below, draw a diagram illustrating the integer and pointer values in memory at the end of execution.

```
int a;
int b;
int *pa = &a;
int *pb = &b;
pa = 17;
pb = 23;
pa = &b;
a = 42;
```

[6]As used here, scaffolding is a metaphor for support given to a learner which enables her to accomplish a task beyond her individual capability (c.f., Vygotsky [1978]; Wood et al. [1976]).

Question 5: A certain video mode of a small computer requires that the color values for adjoining pixels be packed into 16 bit integer shorts. The display has $240 * 160$ pixels. Thus, for example, the first two pixels (Row 0, Column 0 and Row 0, Column 1) are stored in the first short in the videoBuffer. This short would be used to look up the RGB value in a color palette table. The low order 8 bits specify the color of the first pixel. The high order 8 bits specify the color of the second pixel. Hint: This means that if you think of the pixels as being numbered stating from 0 up to $240 * 160 - 1$ that the even numbered pixels correspond to the low order bits and the odd numbered pixels correspond to the high order bits. There is a global variable (static) named videoBuffer which contains the starting address of the video buffer which is, of course $(240*160)/2$ shorts in length. The color values are simply values between 0 and 255.

Part A: Write a function called getPixelOffset which will accept the row and column number of a pixel and returns the offset from the start of the buffer to the short which contains that pixel.

```
int getPixelOffset(int row, int column)
{

    //Blank space provided for student response

}
```

Part B: Complete the function setPixel below which accepts the row and column number of a pixel on the screen and a color value between 0 and 255. It should set that pixel to the supplied color value. NOTE: This function requires you to pack the color value into the appropriate half of the short without disturbing the value in the other half.

```
/* videoBuffer pointer */
unsigned short *videoBuffer = (unsigned short *)0x6000000;

void setPixel(int row, int column, unsigned char color)
{
     int offset = getPixelOffset(row, column);

      //Blank space provided for student response

}
```

Question 6: Write the function istrCmp which performs a case-insensitive comparison of two null-terminated character strings a and b. The function should return 1 if $a > b$, -1 if $a < b$, and 0 if the two strings are identical. You may assume that the input strings only contain alphabetic characters. NOTE: Do not use any string library functions in your answer (e.g., strlen, strcat, strcmp);

Helpful integer ASCII values for some characters are below:

```
'A' = 65
'Z' = 90
'a' = 97
'z' = 122

int istrCmp(char* a, char* b)
{

    //Blank space provided for student response

}
```

ACKNOWLEDGMENTS

We extend our gratitude to our study volunteers for their participation and to our colleagues from the Computer Science Education Research group at Georgia Tech for their help with data collection and analysis. Specifically, we thank Jill Dimond, W. Michael McCracken, Lijun Ni, Brian O'Neill, and Svetlana Yarosh for their assistance with this project.

REFERENCES

AAUW. 2000. *Tech-Savvy: Educating Girls in the New Computer Age*. American Association of University Women Educational Foundation, Washington, DC.

ALT, C., ASTRACHAN, O., FORBES, J., LUCIC, R., AND RODGER, S. 2006. Social networks generate interest in computer science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'06)*. 438–442.

ANASTASI, A. AND URBINA, S. 1997. *Psychological Testing*, 7th ed. Upper Saddle River, NJ: Prentice Hall.

ANDERSON, J. R. AND BOWER, G. H. 1973. *Human associative memory*. Washington, DC: Winston and Sons.

BLANK, D. 2006. Robots make computer science personal. *Communications of the ACM 49,* 12 (December), 25–27.

BONATO, V., MENOTTI, R., SIMÕES, E., FERNANDES, M. M., AND MARQUES, E. 2004. Teaching embedded systems with FPGAs throughout a computer science course. In *Proceedings of the Workshop on Computer Architecture Education (WCAE'04)*.

BRANSFORD, J. D., BROWN, A. L., AND COCKING, R. R. 2000. *How People Learn: Brain, Mind, Experience, and School*. Exp. ed. Washington, DC: National Academy Press.

BRUER, J. 1993. *Schools for thought: A science of learning in the classroom*. MIT Press, Cambridge, MA.

DODDS, Z., ALVARADO, C., KUENNING, G., AND LIBESKIND-HADAS, R. 2007. Breadth-first CS 1 for scientists. In *Proceedings of the 12th SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'07)*. 23–27.

FLEISS, J. L., LEVIN, B., AND PAIK, M. C. 2003. *Statistical methods for rates and proportions*, 3rd ed. Hoboken, NJ: John Wiley & Sons.

FORTE, A. AND GUZDIAL, M. 2004. Computers for communication, not calculation: Media as a motivation and context for learning. In *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS'04)*.

FORTE, A. AND GUZDIAL, M. 2005. Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, *48,* 2, 248–253.

GAY, L. R. AND AIRASIAN, P. W. 2000. Selection of measuring instruments. In *Educational research: Competencies for analysis and application*, 6th ed., Upper Saddle River, NJ: Merrill. 147–198.

GUZDIAL, M. 2005. *Introduction to Computing and Programming in Python: A Multimedia Approach*. Upper Saddle River, NJ: Prentice Hall.

GUZDIAL, M. AND SOLOWAY, E. 2002. Teaching the Nintendo generation to program. *Communications of the ACM*, *45,* 4, 17–21.

KNUTH, D. E. 2005. *The art of computer programming, Volume 1, Fascicle 1: MMIX—A RISC computer for the new millennium*. Reading, MA: Addison-Wesley.

KOLODNER, J. L. 1997. Educational implications of analogy. *American Psychologist*, *52,* 1, 57–66.

MARGOLIS, J. AND FISHER, A. 2002. *Unlocking the Clubhouse: Women in Computing*. Cambridge, MA: MIT Press.

PAPERT, S. 1991. Situating constructionism. In *Constructionism: Research reports and essays, 1985-1990*, I. Harel and S. Papert, eds. Norwood, NJ: Ablex. 1–11.

PATT, Y. AND PATEL, S. J. 2003. *Introduction to computing systems: From bits and gates to C and beyond*, 2nd ed. Boston: McGraw-Hill.

PATTERSON, D. A. AND HENNESSY, J. L. 2007. *Computer organization and design: The hardware/software interface*. San Francisco, CA: Morgan Kaufmann.

PEARSON, M., ARMSTRONG, D., AND MCGREGOR, T. 2002. Using custom hardware and simulation to support computer systems teaching. In *Proceedings of the 2002 Workshop on Computer Architecture Education (WCAE'02)*.

PINTRICH, P. R. AND SCHUNK, D. H. 1996. *Motivation in education: Theory, research, and applications*. Englewood Cliffs, NJ: Prentice Hall.

SHAFFER, D. W. AND RESNICK, M. 1999. "Thick" authenticity: New media and authentic learning. *Journal of Interactive Learning Research*, *10,* 2, 195–215.

SMITH, D. 2007. *Engineering computation with MATLAB*. Boston: Addison Wesley.

TEW, A. E., FOWLER, C., AND GUZDIAL, M. 2005. Tracking an innovation in introductory CS education from a research university to a two-year college. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'05)*. 416–420.

VYGOTSKY, L. S. 1978. *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard Univ. Press.

WILENSKY, U. 1991. Abstract meditations on the concrete and concrete implications for mathematics education. In *Constructionism*, eds. I. Harel and S. Papert. Norwood, NJ: Ablex. 193–204.

WOOD, D., BRUNER, J., AND ROSS, G. 1976. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry and Allied Disciplines*, *17*, 89–100.

XU, D., BLANK, D., AND KUMAR, D. 2008. Games, robots, and robot games: Complementary contexts for introductory computing education. In *Proceedings of the 3rd Microsoft Academic Days Conference on Game Development in Computer Science Education (GDCSE'08)*. 66–70.

YAROSH, S. AND GUZDIAL, M. 2007. Narrating data structures: The role of context in CS2. In *Proceedings of the 3rd International Workshop on Computing Education Research (ICER'07)*. 87–98.

Received April 2008; revised August 2008; accepted August 2008