Design Methodology Suitable for Team-based Embedded Systems Education

J.W. Bruce
Department of Electrical and Computer Engineering
Mississippi State University, Mississippi State, MS 39762-9571
jwbruce@ece.msstate.edu

Abstract

This paper describes a design methodology useful for team-based (cooperative) and problem-based embedded systems education. The design methodology includes a detailed design convention and formalized hardware and code design reviews where the quantity and nature of each design errors are documented. Reviews are held before design implementation and dramatically reduce development time by aborting the far too common cycle of develop, test, change, and test again. The design methodology presented here yields a high-quality product within a short design cycle, while mimicking design methodologies found in industry. Furthermore, data obtained in design reviews can be used to improve the instruction quality and track the maturity of the student design skills. An added benefit of the methodology is development and exercise of the students' teaming and communication skills often neglected by traditional engineering curricula. The proposed methodology has been used in a senior-level embedded systems course at Mississippi State University. In this course, student teams design, build, and troubleshoot a microcontroller-based project composed of common embedded systems peripherals, including I/O and electromechanical devices, industry standard communication networks, and complex digital integrated circuits. The target design is progressive requiring each successive subsystem to be incorporated without disturbing previously completed subsystems. Details of the methodology as it relates to this course offering, sample design review forms, collected data and discussion are presented. Course evaluations were obtained from students and external reviewers, and the results show that offering was well received and achieved its educational objectives.

1. Introduction

Embedded computer systems are quietly changing our world — the way we eat, play, work, and live. Embedded systems are used in a diverse range of products including home appliances, automobiles, toys, and medical equipment. Embedded systems are located at the “front line” where technology interacts with the physical world. These systems measure temperature, motion, human response, and other inputs. They also control motors and other devices, and deliver information for human consumption. The movement of the last two decades toward more ubiquitous computing systems will continue and embedded systems will become even more prominent in every aspect of technology and life [6]. Engineers comfortable with common embedded systems components, embedded systems design, and embedded system functions, such as data acquisition, processing, and delivery, will be well equipped for the future.

Approaches to embedded systems education are as varied as the programs that contain them. Most approaches have student teams specifying, designing, and implementing their own designs. While this situation simulates professional engineering practice, the experience very likely is the student's first exposure to the design process [4]. Not surprisingly, careful and resource intensive supervision is required if designs are to be successful. Structured laboratory experiences (each student or group of students perform rigid and contrived experiments) require less faculty resources but do little to develop student design and project management skills [10][12]. A compromise between the two approaches has been created by the author [2].

The embedded systems design experience described in [2] strives to develop professional skills that will serve students well in their careers in addition to the "traditional" technical skills the student expect. Overarching goals of the experience are to expose the student to a realistic embedded systems design environment and to develop the student's teamwork and lifelong learning skills. The design experience strives to emulate situations found in industry. Students work in cooperative design teams composed of students with diverse technical backgrounds and skill sets. Students must develop teamwork and communication skills. Lifelong learning skills are promoted through the design experience's problem-based learning approach.

In an initial offering of the experience described in [2], student teams commonly reported "pulling all-nighters" in the lab. Rarely were the marathon sessions spent in hardware design or troubleshooting. A questioning of the students revealed that much of the effort was in software programming, testing, debugging, and further testing. Designs made excruciatingly slow progress toward meeting the system requirements. The author (the course instructor) was aghast. This cohort was exceptional well qualified by prior experience (obtained via cooperative education rotations in industry) and high marks in prerequisite courses. Further investigation revealed the cohort was utilizing a disturbing development cycle, best described as "trial-and-error". The development cycle employed by the students was ad hoc and chaotic. Using the Software Engineering Institute's Capability Maturity Model (CMM), the students' process was textbook CMM Level 1 [14]. (See Fig. 1.)

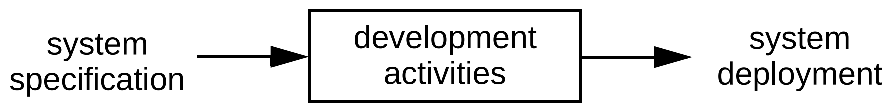

Figure 1. "Processless" development (CMM Level 1)

In the following sections, the design experience is reviewed briefly. Observations from the initial offering are presented. Then, improvements to the design experience are suggested. Finally, faculty and student assessment results from a second offering of the course are presented and discussed.

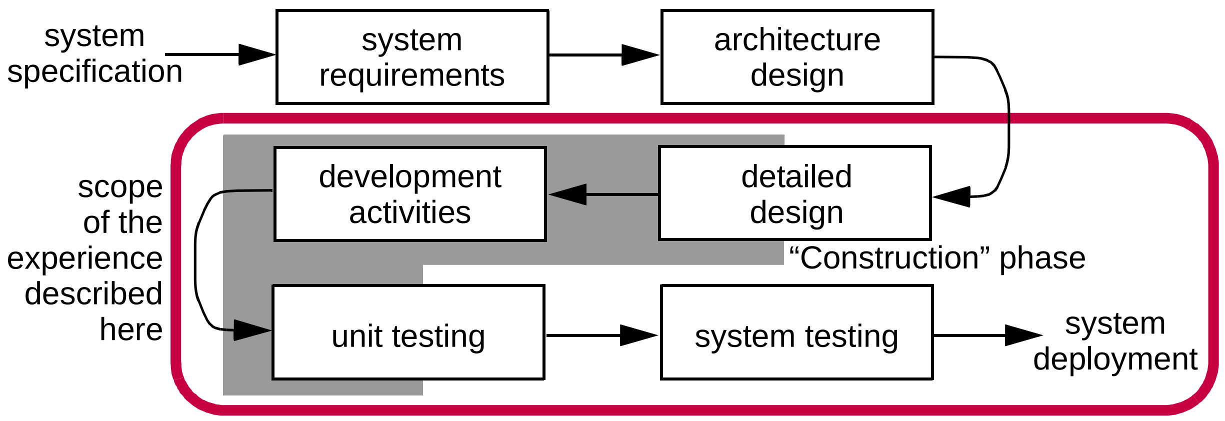

Figure 2. Waterfall development process

2. Design Experience

Before creating the design experience, input from faculty, alumni, and employers was solicited. It was agreed that it was important that students, in addition to learning about embedded systems, (i) experience an environment as close to industry as possible, (ii) practice interpersonal and communication skills, and (iii) develop lifelong learning skills. These goals are what Waitz and Barrett term "implicit curriculum", those educational topics students are expected to know (often requested by employers and alumnae/i), but not taught in any course [16].

The design experience is offered at the senior and introductory graduate students as a sixteen-week semester three credit-hour course. The course was organized into two hours lecture and three hours laboratory each week. Since the later design tasks require much time, lecture periods are scheduled for every meeting early in the semester to cover subject material required in the laboratory portion of the course. Therefore, three hours of lecture and three hours of laboratory were given in the first portion of the semester. Later, lectures were culled back to one hour per week with some informal meetings occurring in the design lab. The author's university requires students to own their own laptop computer. Freeware tools and round-the-clock access to the design lab gives student teams flexibility to set working hours to accommodate busy schedules.

2.1. Design environment

Simulating an industrial-like working environment increases the realism of the project and student perception of the relevance of their education [4]. Design tools and practice should mimic their industrial counterparts as closely as university constraints allow. Design software tools should include "industrial-strength" features such as real-time in-circuit debugging and field programmability, while being low cost or free and easily to install and operate.

Constituent employers noted that new engineering graduates are not accustomed to or not willing to follow design specification and conform to corporate design practices such as design and coding conventions. The experience in [2] was created, in part, to give students exposure to strict design specifications and conventions and the need to follow them faithfully. The teams are required to design according to a progressive design specification. The design specification is "progressive" as the complete design is divided into distinct tasks of subsystem design. Each week, the team creates and incorporates the week's design into the system. All previous tasks must continue to function as described in their specifications. Some have suggested that long retention learning and skill adoption does not take place until skills are internalized and used as a part of a "professional persona" [13]. The progressive specification used in the experience leads to this internalization.

Teams are required to design according to specified hardware and software programming conventions. Hardware design conventions include specific rules on integrated circuit placement on breadboards, wiring color conventions, and limitations on wiring resistances and loading capacitances. Programming conventions require students to follow specific variable, subroutine, macro, and library naming rules. Memory usage is strictly specified, and all code written is to be as hardware, memory, peripheral, and frequency independent as possible. Hardware design and software programming conventions facilitate cooperative designs where subsystems created by individual students can be combined together. Furthermore, design conventions facilitate and ease intra-team, inter-team, and team-instructor communication during design reviews. This cooperative, team-based design flow approximates the cooperative learning technique called "jigsaw" [11].

Finally, the importance of specification and design conventions adherence is further reinforced as the lab progresses. After the first few tasks are complete and students have confidence in their hardware and software design skills, existing intellectual property (IP), such as circuit and software libraries, is incorporated into the design. Component manufacturers provide a portion of the IP with the remainder created by students in previous course assignments.

2.2. Cooperative design skills

In data taken from employers hiring graduates from the author's institution, a common request is that engineering education should work to improve teamwork and communication skills. Other educators have heard similar comments from various industries [8]. ABET Criterion 3(d) and 3(g) have similar requirements [1]. Students must practice those workplace social skills like conflict resolution and polite communication of potentially painful critical design review comments. Interpersonal skills are exercised further with team-building active learning exercises during lectures and with group homework assignments [11]. Furthermore, students learn interdependence and group accountability.

2.3. Lifelong learning skills

ABET's Criteria 3(i) requires that engineering programs demonstrate that graduates have a commitment and ability for lifelong learning [1]. Programs must strive to make students aware that they are responsible for their own learning, not the teacher. In effect, university education is the start of a professional education, not the culmination. When confronted with the realization that they are responsible for their own learning, students often feel self-doubt and lack self-confidence. Students can progress through seven steps similar to those associated with trauma and grief [17]. Often, these steps are manifested as resistance to any instructional pedagogy that deviates from passive learning. Student resistance fades as their abilities and confidence grow [3].

Student are required to critically read and understand component datasheets. The student must comprehend the design from a detailed specification. Reconciling datasheet information with the design specification requires all six levels in Bloom's Taxonomy: identification of design requirements (knowledge), interpretation of same (comprehension), derivation of possible real-time and resource constraints (analysis), generation of solutions (application), implementation of designs (synthesis), and selection of a suitable design (evaluation).

All homework and lab designs are performed as student teams with a single deliverable per team. Students must function effectively in a group working together toward a common goal. Cooperative learning and design teams are a proven technique to facilitate these skills [11][13]. Much research shows that each experience in a sound cooperative learning environment benefits the students in many ways. Students learn to rely on themselves and one another. The instructor becomes one resource, not the one-and-only resource [11][15]. Therefore, lifelong learning skills are honed.

2.4. Observations from Initial Offering

In keeping with the tenet of lifelong learning that students are responsible for their own learning, students must be held responsible for assessing instructional effectiveness. Student assessment aids in maturation of team citizenship [3] and promotes reflective evaluation [10]. Student-based assessments were taken three ways in the initial course offering. First, students provided qualitative mid-semester course evaluations and suggestions. Specifically, they provided feedback on what instructional methods are and are not "working" for them, which course topics have been clearly understood, and which topics are still confused. Students were encouraged to provide suggestions to improve the remainder of the course. I adopted my teaching style as much as possible to further engage the students in the assessment process. Second, students provided weekly quantitative team citizenship and participation assessments of their teammates and themselves. This assessment provides the individual accountability required in cooperative learning and prevents academic "hitchhiking" [11]. "Hitchhikers" and "overachievers" will exist within any cooperative effort. That fact, in itself, is a valuable lesson for students to learn. I have used a simple "autorating" scheme, much like the one in [9], for years with much success. Each team's design task is evaluated as a whole. Each team member receives an individual score computed from the team's score adjusted by a "citizenship" factor computed from the team's evaluation forms [9]. Finally, students were asked to assess the course as a whole at the end of the semester. They evaluated the skills they obtained and the course's perceived relevance. To the statement "I have become more competent in the (embedded systems) subject area because of this course", the students agreed with a mean score of 4.56 on a Likert scale, where 1 is "strongly disagree" and 5 is "strongly agree". In the final course evaluations, a significant number of students cited "integration and application of material learned in prior courses" as the most enjoyable part of the experience. One student reported that the course made them realize that they had "actually learned something" during their years in school.

Faculty reviewers, usually the student's senior design project advisor, were asked to evaluate the design experience effect upon the student. Reviewers observed that the cohort exhibited average or better abilities in the areas described by objectives given in [2]. Many influences are involved in a student's education, and the design experience described here is only a part of that education.

However, I take some pride in learning that several external faculty reviewers commented that many students from this cohort executed some of the most successful capstone projects in recent years. Finally, students in this initial offering were seen in the student common areas brainstorming capstone design project ideas, collaborating, and studying for many different courses. This continued in the semesters following the course offering. I interpret this as evidence that students have realized some value-added benefit to cooperative learning and have started to internalize the interpersonal and communication skills.

The cohort taking the initial offering was exceptionally well prepared. Most students had GPAs placing them in the top of the class. Every team was successful in meeting the design specification fully. Many teams implemented extra features. However, their effort was not efficient. Many students spent unbelievably long hours in the lab. Working through the night until the wee hours was common. My TA and I were perplexed. Our prototype effort during the previous semester was arduous work, but we did not spend the hundreds of person-hours that these “all-star” teams were exerting. Questioning the students exposed that the long hours were not spent in hardware design or troubleshooting. Most of their time was in software programming, testing, debugging, and further testing. In general, the team would write code for the task, download onto the target hardware, and run it. As expected, it would often crash. The latest code changes would be examined, and a rapid decision formed as to the cause. A single line of code is changed, and the process begins a new. Most teams were developing the code in an evolutionary, trial-and-error process. They seldom gave any thought to the ramifications brought about by changing one line of code. When that “fix” didn’t work, they changed another line of code. After several hours, their software modifications created a multitude of defects that they hoped to find in testing. My discovery of their methods came late in the semester. I vowed to take corrective action in the next offering.

3. Take Two — Design Experience Improvements

In order to impose a more thoughtful, systematic, and repeatable design process in the second offering of the embedded systems design course, I turned to proven IEEE Software Engineering Standards. Most introductory software engineering texts, like [14], provide an excellent introduction.

3.1. Adding “Process” to Software Development

In the university environment, the “time-to-market” is approximately 16 weeks. This short duration makes it difficult to motivate, train, and deploy a full-featured industrial software development process with students possessing a limited software engineering background. The foremost learning objective of the offering described here is embedded systems, to include hardware, software, and interfacing [2]. However, a main objective of the experience is to create a realistic team-based design environment. Toward this end, I added requirements based very loosely on two IEEE software engineering standards to the course’s second offering. Specifically, IEEE Standard 1028 (Software Reviews), and IEEE Standard 982.1 (Software Development Metrics) were selected to transform the first offering’s development process in Fig. 1 into the more mature and reproducible development process in Fig. 2. While the academic calendar restrains us from deploying a full feature process, the additions described give our development process at least some elements in CMM Levels 2 and 3 [14].

3.2. Adding Formal Design Reviews to the Development Process

IEEE Standard 1028 defines five different types of review for software development. In the interest of time and to facilitate teaming and communication skills, I chose to implement only one review, the “Inspection”. In the course offering described here, these inspections are called “Code Design Reviews”. These reviews create a peer evaluation of code before it is deployed. During the review, “impartial” reviewers (team members who did not author the code) try to identify software defects, deviations from standards and requirements, and determine the code’s behavior as written, not its intended behavior. The goal of this design review process is to identify as many defects as possible. (Fixes are made by the code author at a later time.)

In the design review process, there are four well-defined design review roles: coordinator, author, reader, and recorder. Ideally, each role is played by a different team member. The role of the coordinator is to facilitate communication, schedule meetings, and ensure the process is successful. The author is the person who wrote the code and, ulimately, corrects the defects identified by the other team members. The reader and recorder act as impartial code reviewers (in addition to the coordinator) and have major roles during the design review meeting.

For the software development portion of the project, the “development activities” block in Fig. 2 involves five steps:

- Planning (coordinator and author)
  Code design review planning involves the coordinator and the author. After the author creates a cleanly compiled code module, the author requests a code design review meeting be scheduled and sends the code module to the coordinator. The coordinator determines a time and location for the meeting that is suitable for all team members, and sends each team member the code to be reviewed. For each week’s design task described in [2], there might be several significant code modules being written. Therefore, the coordinator might be dealing with multiple authors, or the team may select to rotate the coordinator role for each code author. In either case, most teams elect to hold a single code design reviews meeting where the design review roles rotate depending on the code being inspected.

- Preparation (entire team)
  Each team member prepares for the meeting by reading and inspecting the candidate code before the meeting. The author is encouraged to do so as well. In fact, one study has shown that upwards of 90% of code defects are identified during the preparation step before the actual code design review meeting [5].

- Design Review Meeting (entire team)
  The coordinator’s role is to keep the review meeting on task and to ensure attendance by all team members.

  The reader is to do just that: read the written code’s behavior aloud. The reader must be vigilant to paraphrase the code’s behavior as written, not the code’s intent. Therefore, the reader should never be the author. The reading should be comprehensive; the review critiques everything about the code: instructions, headers, comments, formatting, coding conventions, and data interfaces.

The recorder maintains written record of the code review meeting findings. Sample design review meeting forms are given in the Appendix.

The code's author is present only to clarify questions that cannot be ascertained from the code itself. (Requests for the author to clarify the code is usually a sign of poorly design work.) Under ideal conditions, the author will not be required to say anything. Another important task of the coordinator is to ensure that the design review is impersonal. The author and his/her ability is not be critiqued, only the written code is under review. The author should never be placed into a situation where he/she feels forced to defend themselves.

- Rework (author)
  At the end of the design review meeting, the team determines whether the severity and number of defects warrants additional design review inspections. If so, the author uses the defect summary form to correct all defects identified and submits the corrected code to the coordinator who schedules another design review meeting as specified in the previous steps. If defects are few and minor in nature, the team may opt for the author to correct the defect independently and proceed to the follow-up step.

- Follow-up (coordinator)
  After the author has corrected the defects identified in the meeting. The revised code is sent to the coordinator again. Using the defect forms previously completed, the coordinator determines if the author has corrected the defects satisfactorily. If corrections do not appear satisfactory to the coordinator, he/she can request additional rework by the author. When the coordinator is satisfied with the code, the coordinator approves the code to proceed to testing and deployment in the lab.

I imposed one additional constraint on the design teams. The code design review process is to be done after compilation, but before testing. After the author has completed the code module in question and obtained an error-free and warning-free compilation, then, and only then, can he/she request the coordinator arrange a design review meeting. The students are instructed that the code in question should not be run on the simulator, or programmed onto the hardware before the design review. The compiler is used to check syntax. The author verifies code functionality mentally. A quick "test-drive" of the code in simulation or in reality is not allowed until after the follow-up step in the code design process. This requirement is imposed to force the student to mentally check and cross-reference every action written in the code. Too often, students (and mature programmers, as well) write code thinking if that some esoteric, unexpected behavior is created that it will be caught in testing. In short, programmers often put off thinking about details until after the code has taken shape. Of course, data has shown, time and again, that correcting defects at later stages of development is much more expensive than if corrected earlier [14]. Since simulation and testing are not allowed before peer review, the code's author will spend the time to think through the code and its effects fully so as not to be embarrassed.

3.3. Measuring the Process

Software development metrics can increase productivity, identify development process shortcomings, increase software quality, and aid in development planning [14]. Furthermore, students' confidence in abilities can be increased by comparing their own metrics to published values in the literature. As most educators will attest, students do not always believe or follow advise gleaned from the instructor's experience. In the first offering, students listened attentively to stories from my industrial experience in embedded systems design. However, few were willing to commit the energy to implement many of the guiding principles created from these examples. Most students were content to use their own development process even though my experiences show that they are defective or inefficient. Software metrics provide "hard" and indisputable evidence.

Many development metrics in IEEE Std. 982.1 are used and obtained very naturally with the design reviews discussed in the previous section. During the design reviews, students were asked to record

- Defects
  Recorded by author, task, type (see forms in Appendix), and severity

- Person-hours
  Recorded by team member, task, and activity (design, development, review, and testing)

- Output
  Recorded as Lines of Code (LoC) by team member as author, task, and routine

Each team member maintains their own records. Students are required to compute several metrics in a variety of ways. Coding efficiency is measured in LoC/workday, where a workday is defined as eight person-hours of effort. Assuming a contract hourly wage of $75/hour, each student determins their development cost in $/LoC. Code quality is measured as number of defects per thousand LoC.

At several points during the semester, I collect data from each team and compare the productivity of each team at the next lecture. This allows me to (*i*) give feedback to each student to ensure quality data collection, (*ii*) identify teams with a poor team dynamic, (*iii*) promote a friendly competition between teams to operate with maximum efficiency and lowest coding defect rate, and (*iv*) motivate engaging discussion on ethical, economic, and design methodology issues.

4. Observations and Discussion

An inventory of the students' abilities was taken during the initial course meeting. Each student completed the survey form in the Appendix. Students provided information on their performance in the prerequisite classes, an introductory microprocessors course and an introductory electronics course. The students gave a self-assessment of their ability to perform seven skills needed during the design experience. The students were asked to provide additional information detailing any prior experience with embedded systems design. Students were to describe their pre-course impressions of the course content and objectives. Finally, student expectations were measured to determine their motivation for taking the course and what they expected to learn.

The cohort consisted of eighteen seniors majoring in electrical engineering (EE) and computer engineering (CPE). All eighteen were men[1]. Eight students were pursuing EE degrees and ten were pursuing CPE degrees. Table 1 shows the results from the skills portion of the abilities survey. Students ranked their own abilities on a scale from 1 (never heard/done it) to 5 (expert/guru). Each table entry is the average score with standard deviation in parentheses.

Table 1: Abilities Survey Results

	Ability	Before (N=18)	After (N=15)
A1	programming in HLL (C, C++, Java, etc.)	3.61 (0.608)	3.67 (0.724)
A2	programming in assembly	3.17 (0.514)	3.93 (0.458)
A3	programming user or human-machine interfaces	2.11 (1.079)	2.53 (1.187)
A4	knowledge of a networking protocol (serial, I2C, CAN, etc.)	2.78 (1.114)	2.86 (0.770)
A5	figuring out datasheets	3.39 (0.608)	3.93 (0.594)
A6	hardware design	2.94 (0.873)	3.73 (0.961)
A7	soldering	3.06 (1.060)	3.67 (1.1047)

The substantial number of required computer science courses in the CPE degree program undoubtedly led to the strong high-level language (HLL) programming skills among the cohort (A1). Students felt less comfortable with their assembly language programming abilities (A2) gained in the microprocessors course. These students' observations agree with the observations leading to this course's learning objectives [2]. Most students reported a "casual knowledge" or less in the "hands-on" skills (A5-A7). Again, student self-assessment agrees with the faculty observation leading to the creation of this course [2]. Several students reported some substantial previous hardware and/or embedded system design experiences, mostly from cooperative education rotations in industry. The students were nearly unanimous in desiring additional hardware design experience, specifically interfacing digital computers to "real-world" devices. The phrases "build a device that does 'something'" and "get my hands dirty" appeared in numerous student responses. Several students cited the gaining the ability to "build their own robots" as an ultimate objective for the course.

At the course conclusion, students were asked to take an optional online survey. The online survey was built, in large part, using the assessment infrastructure developed for EC2000 accreditation at Mississippi State University [7]. The survey questioned the students concerning frequency of selected actions, their perceived progress in abilities, and change in confidence levels. About 83% (15 out of 18) of the class participated in the survey. The ability survey questions in Table 1 were asked again. The results are given in the rightmost column of Table 1. Student report a negligible increase for ability A1. This is expected as HLLs were used only as pseudocode to express ideas, program flow, or documentation. The design experience concentrated on assembly language,

1. Women earn a little more than 10% of EE and CPE bachelor degrees at MSU. The two offerings of the elective class described here have enrolled exactly one woman out of 36 students (2.8%). Why women are not enrolling is an open question.

interfacing, and hardware design skills [2]. The students report a substantial increase in these abilities (A2, A5-A7).

In the online survey, students were asked to rate the frequency of several actions. Frequencies are rated from 0 (never) to 3 (very often/almost always). Table 2 shows the results ordered by average score with standard deviation given in parentheses. Students felt the design environment was "often" like what they thought industry would use. A significant portion of the cohort felt that they were "very often" called upon to work and communicate as a team and given assessment of each team member's contribution.

Table 2: Task Frequency Survey Results

In THIS COURSE, ... (N=15)	Score (0 to 3)
I had to make critical assessments of my work, other's work, and our relative contributions	2.67 (0.617)
I had to learn to work and communicate with others to accomplish my tasks	2.60 (0.632)
I use knowledge or skills learned in previous/other classes	2.53 (0.640)
I had to supplement material learned in lecture with self-study	2.20 (0.941)
the design environment is very similar to what I know/envision industry to be like	2.07 (0.829)

Students rated their personal progress in each of the course objectives formulated in [2]. Responses are rated from 0 (none) to 3 (a great deal). Table 3 shows the survey results ordered by average score. Students unanimously felt their understanding of embedded systems increased "a great deal". This learning objective constitutes the "information transfer" expected by students. However, it is interesting to note a large increase in appreciation for the value of design documentation. My past experience has shown that students often dismiss these documents initially and express these feelings aloud. It seems that by the semester end, the students have bought into using these documents in the design process. As hoped, students responded that their abilities to learn on their own, work in teams, and troubleshoot were improved by the experience. Time constraints prohibited us from doing many of the team-building exercises I had planned. However, students note "some" improvement in their ability to motivate others.

Table 3: Learning Objective Progress Survey Results

Progress made, BECAUSE OF THIS COURSE, in your ... (N=15)	Score (0 to 3)
understanding of embedded systems and components	3.00 (0.000)
your appreciation of the usefulness of design documents like specifications and coding conventions	2.80 (0.414)
ability to troubleshoot systems involving hardware	2.73 (0.458)
ability to create a system to meet a detailed design specification	2.67 (0.488)
ability to read, understand, and apply datasheets	2.60 (0.910)
skill to work in an engineering design team	2.60 (0.632)
ability to troubleshoot systems involving software	2.53 (0.640)

Table 3: Learning Objective Progress Survey Results

Progress made, BECAUSE OF THIS COURSE, in your ... (N=15)	Score (0 to 3)
ability to apply and effectively use existing intellectual property, like software libraries or circuits	2.47 (0.640)
ability to evaluate and justify competing designs against a specification	2.40 (0.737)
skill in assessing the performance of yourself and others	2.00 (1.000)
ability to motivate others to achieve their maximum potential	1.67 (1.175)

Students were also asked to rate their change in confidence, due to the taking the course, in several areas. Students rated the change from -2 (decreased greatly) to +2 (increased greatly). A student rating of zero denotes no change in confidence. Table 4 shows the results ordered by average score. In Table 4, students report the greatest ability increases in teaming and leadership skills. Design ability increases lagged slightly. While in Table 3, students rated their progress in teaming behind some design abilities. I would propose that students rated their absolute ability levels in Table 3, while Table 4 represents their perceived ability improvement. This interpretation leads me to think that students felt quite confident in their abilities with the mechanics and subject matter in the design experience. However, they were not as confident in their teaming skills, although they thought they were very much improved.

Table 4: Confidence Change Survey Results

Changes, as a result of TAKING THIS COURSE, in your .... (N=15)	Score (-2 to +2)
confidence in your ability to work in a design team has ...	1.27 (0.884)
confidence in your ability to lead a design team has ...	1.20 (1.014)
confidence in your ability to design, build, and troubleshoot hardware has ...	1.20 (0.676)
confidence in your ability to design, build, and troubleshoot software has ...	1.13 (0.640)
self-reliance in learning new concepts or skills has....	1.13 (0.516)

I wish to convey a couple anecdotes about the design review process and the documentation required to support it. As expected, I encountered some initial resistance from the students. I was prepared and countered their resistance with many examples from the literature to support (quantitatively) the effectiveness of reviews. Of course, some students continued to be skeptical of this process that they argued was a waste of time. They, as students are oft to do, wanted to dive in and design — a design process I termed "hacking". I insisted the students humor me and follow the prescribed procedure for a few weeks. I promised we would discuss, evaluate, and incorporate any improvements they could suggest. (By the way, this is an excellent way to give students ownership and responsibility of their own learning.) After the first lab design task, one team was elated to give a testimonial. It went something like this: "We did not simulate, emulate, or download our code before the review process. Our review found several bugs in the code and hardware wiring errors. Each error was fixed and certified by the team. Our very first attempt at downloading our code onto the hardware was a success, and we were done with the lab in record time." For the remainder of the semester, this team insisted on following the design procedures exactly. This team consistently finished assignments first and with high quality. Clearly, this team had internalized the design review process and made it a part of their "professional persona".

The end of the semester brings a much more complex design task to the team: integration of several different subsystem. This complex task helps to "sell" the methodical and documented design process to most of the remaining skeptics. But, I had one last trick up my sleeve to convince them. Money. That always gets the students' attention. After all designs were completed and design metrics were accumulated, I had each team use their coding efficiency, defect-production rate, and time expended to determine a Manufacture's Suggested Retail Price (MSRP) for the design, assuming several different potential sales volumes. A sample analysis is given in the Appendix. This exercise was more successful than I could have ever imagined. Students immediately recognized that the age-old adage "time is money" is very much true. They also appreciated that the design metrics could help them to monitor an ongoing design process and to make more accurate schedules in future projects. I am looking forward to incorporating similar exercises into my other courses.

Further analysis of and reflection on the survey data is required, but initial inspection indicates the design experience was successful in teaching the "implicit curriculum" of teaming, communication, and lifelong learning skills. Also, students gained appreciation for a more formal and realistic design flow. The students felt the experience was realistic, and this heightens their perception of the relevance of their education.

5. Conclusions

In order to meet the current and future needs for embedded systems engineers, the author has created an embedded systems design experience that emulates industrial situations as much as possible. The student teams build a progressively more complex design using formal and documented design reviews and collect data to monitor their design performance. The student's response is an increased engagement and higher perceived relevance of the educational experience. The team-based approach also develops the student's teamwork and lifelong learning skills

Bibliography

1. Accreditation Board for Engineering Education (ABET), "Engineering Criteria 2000". Available at http://www.abet.org/
2. J.W. Bruce, J.C. Harden, and R.B. Reese, "Cooperative and progressive design experience for embedded systems," *IEEE Trans. Educ.* (To appear)
3. R.S. Culver, D. Woods, and P. Fitch, "Gaining professional expertise through design activities," *Engineering Education*, vol. 80, pp. 533-536, 1990.
4. A.J. Dutson, R.H. Todd, S.P. Magleby, C.D. Sorensen, "A review of literature on teaching engineering design through project-oriented capstone courses," *J. Engineering Educ.*, pp. 17-28, 1997.
5. P.J. Fowler, "In-process inspections of work products at AT&T," *AT&T Tech. Journal*, pp. 102-112, March 1986.
6. J. Ganssle, *The art of designing embedded systems*, Boston: Newnes, 2000.
7. J.C. Harden and M.G. Lane, "Web-based tools for assessment automation," *Proc. 2002 ASEE Annual Conference and Exposition*, Sesssion 1532, 2002.

8. M. Hedley and S. Barrie, “An undergraduate microcontroller systems laboratory”, *IEEE Trans. Educ.*, vol. 41, no. 4, pp. 345, 1998.
9. D.B. Kaufman, R.M. Felder, and H. Fuller, “Accounting for individual learning effort in cooperative learning teams,” *J. Engineering Educ.*, pp. 133-140, 2000.
10. D.L. Maskell , “Student-based assessment in a multi-disciplinary problem based learning environment,” *J. Engineering Educ.*, pp. 237-241, 1999.
11. W.J. McKeachie and G. Gibbs, *Teaching Tips: Strategies, Research, and Theory for College and University Teachers, 10/e*, Boston: Houghton Mifflin, 1998.
12. W.R. Murray and J.L. Garbini, “Embedded computing in the mechanical engineering curriculum: a course featuring structured laboratory exercises”, *J. Engineering Educ.*, pp. 285-290, 1997.
13. R. Pimmel, “Cooperative learning instructional activities in a capstone design course,” *J. Engineering Educ.*, vol. 90, no. 3, pp. 413-421, 2001.
14. R.S. Pressman, *Software engineering: A practitioner’s approach, 5/e*, New York: McGraw-Hill, 2001.
15. R.H. Todd, C.D. Sorensen, and S.P. Magleby, “Designing a senior capstone course to satisfy industrial customers,” *J. Engineering Educ.*, vol. 82, no. 2, pp. 92-100, 1993.
16. I.A. Waitz and E.C. Barrett, “Integrated teaching of experimental and communication skills to undergraduate aerospace engineering students,” *J. Engineering Educ.*, vol. 86, no. 3, pp. 255-262, 1997.
17. D.R. Woods, *Problem-based learning: How to gain the most from PBL*. Waterdown, Ontario: Donald R. Woods. 1994.

J.W. BRUCE

Dr. Bruce is an Assistant Professor of Electrical and Computer Engineering at Mississippi State University. He received the B.S. degree from the Univ. of Alabama-Huntsville in 1991, the M.S. degree from Georgia Institute of Technology in 1993, and the Ph.D. from the Univ. of Nevada Las Vegas in 2000. Dr. Bruce’s research and teaching interests are in embedded systems design and VLSI.

Mississippi State UNIVERSITY

ELECTRICAL AND COMPUTER ENGINEERING

Code Design Review Error Listing Form

Project ____ **Author** __________________________________

Function Name(s)____ **Date** ______________________

Location	Error description	Major	Minor

Recorder ____

ELECTRICAL AND COMPUTER ENGINEERING

Code Design Review Summary Form

Project ____ **Author** ________________________________

Function Name ____**Date**______________________

Num. of Errors		**Error Type**
Major	**Minor**	
		Code does not follow coding conventions
		Function size and complexity are unreasonable
		ISR size, complexity, execution time are unreasonable
		Unclear expression of ideas in code
		Poor encapsulation
		Function prototype not correctly used
		Data types do not match
		Uninitialized variable at function start
		Uninitialized variable going into loop
		Poor logic - will not function as needed
		Poor or missing comments
		Error condition not caught or ignored
		Switch statement without a default case
		Incorrect syntax
		Non-reentrant code in dangerous places
		Slow code in speed-critical area
		Interrupts are not masked during possible critical code
		Other:
		Other:

Major bugs are ones that will result in a problem that the customer will see. Minor bugs are those that include spelling errors, non-compliance with coding conventions, and poor workmanship that does not lead to a major error.

Time Code Design Review Started: ____________________________ **Ended:** ____________________

Recorder ____

Project MSRP Analysis

Units sold (for amortization)	100		10000	1000000
Manuf costs % of BoM costs	100%		Manufacturing costs as percentage of BoM	
Manufacturer margin	35%		Manufacturer's profit (as percentage of TMC)	
Distributor margin	30%		Distributor's profit (as percentage of distributor's cost)	
Retailer margin	40%		Retailer's profit (as percentage of retailer's cost)	
NRE costs	$ 18,750.00		obtain from Team semester metrics	
Manufacturer				
Development cost/unit		$ 187.50	$ 1.88	$ 0.02
BoM cost (Given by instructor)		$ 48.30	$ 48.30	$ 48.30
Estimated manufacturing costs		$ 48.30	$ 48.30	$ 48.30
(labor, F&A on manuf., packaging, etc.)				
Total manufactured cost (TMC)		$ 284.10	$ 98.48	$ 96.62
Margin		$ 99.44	$ 34.47	$ 33.82
Manufacturer's sales price		$ 383.54	$ 132.94	$ 130.44
Distributor				
Distributer's cost		$ 383.54	$ 132.94	$ 130.44
Margin		$ 115.06	$ 39.88	$ 39.13
Distrbutor's sales price (wholesale)		$ 498.60	$ 172.82	$ 169.57
Retailer				
Retailer's cost		$ 498.60	$ 172.82	$ 169.57
Margin		$ 199.44	$ 69.13	$ 67.83
Retailer's sales price (MSRP)		$ 698.03	$ 241.95	$ 237.39