Interdisciplinary Teaming as an Effective Method to Teach Real-Time and Embedded Systems Courses

James R Vallino
Department of Software Engineering
Rochester Institute of Technology
Rochester, NY 14623, USA
+1.585.475.2991
J.Vallino@se.rit.edu

Roy S Czernikowski
Department of Computer Engineering
Rochester Institute of Technology
Rochester, NY 14623, USA
+1.585.475.5292
rsceec@rit.edu

ABSTRACT

The body of knowledge for engineering real-time and embedded systems spans multiple computing disciplines. To effectively prepare students to work in these areas requires coursework that uses an interdisciplinary approach. This paper describes the approach that Rochester Institute of Technology's Departments of Computer Engineering and Software Engineering developed. This approach uses a cluster of three courses which cover a range of topics in real-time and embedded systems engineering. Students in each discipline take the courses, and teams of two, with one student from each discipline, work on all course projects. The paper describes the cluster of courses, their evolution over the last five years, and the laboratory in which the classes are taught. We present evaluation data to show the courses' effectiveness increasing student interest in real-time and embedded systems, and helping them obtain employment in the area.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**] – *real-time and embedded systems,* K.3.2 [**Computers and Education**] Computer and Information Science Education – *computer science education*.

General Terms

Design

Keywords

Real-time and embedded systems education, real-time and embedded systems courses.

1. INTRODUCTION

The standard computing curricula concentrate primarily on general-purpose desktop applications. The demands and requirements for these systems are notably different than those for real-time and embedded systems. Without coursework that specifically addresses the special requirements for these systems, students will not have the opportunity to gain the necessary skills for engineering software in real-time and embedded systems. Additionally, the current interdisciplinary nature of the real-time and embedded systems profession intertwines intimate knowledge of both the hardware and the software operating the components. Many traditional courses have worked exclusively with small microcontroller projects. This unfortunately does not reflect the breadth of the current field. We set out to develop our approach so that our course cluster and laboratory facilities encapsulated this reality and provided our students with exposure to a broader range of skills needed for entry-level engineering of real-time and embedded systems.

We presented our original work including detailed course objectives, course projects and initial evaluation of the first two courses in [4, 16]. Since that work, we have made significant changes to those two courses, and delivered the third course several times. This paper provides background on our lab, and describes the current syllabi for all the courses. We detail the lessons we learned and improvements we made to the courses as they evolved in the three years since our previous reporting. This includes the complete development of the third course. The paper concludes with our most recent evaluation data, and future directions for our work. We also have information about these courses, including password protected areas for faculty, on our real-time and embedded systems website [15].

2. REAL-TIME AND EMBEDDED SYSTEMS AT RIT

2.1 Background

In the computer engineering program at Rochester Institute of Technology, senior projects often focus on real-time and embedded systems, but there was no formal instruction in the engineering of the software for these systems. The software engineering program had an embedded systems application domain comprising three courses: two standard operating systems courses offered by computer science and a concurrent programming course from computer engineering. None of these courses directly addressed issues in developing real-time or embedded software; they had been chosen because they were the closest courses relevant to the domain.

We decided that the best way for us to address these shortcomings in the real-time and embedded domain in both the computer engineering and the software engineering curricula was to develop an interdisciplinary approach. The presence of students from both departments created a unique opportunity for synergy. The computer engineering students possess significant knowledge of electronics and control systems along with software development skills at the lower-levels. The software engineering students possess significant knowledge of how to engineer complex software systems including the design and modeling of those systems. They possess skills focused on the engineering of software that are more fully developed than for a student in the typical computer science program. Developing software for real-time and embedded systems is where the skills of computer engineering and software engineering students intersect.

It should be noted that all undergraduate engineering students at Rochester Institute of Technology are required to have a year of cooperative work experience before being awarded a baccalaureate degree in the five year engineering programs. These "coop" work periods are interspersed with academic quarters of study in the last three years of study. The typical students in these courses have had about nine months of work experience before entering these courses. To date, we have offered the first two courses in the sequence multiple times with some consistency; the third course has been something of an ongoing experiment that seems to be stabilizing.

2.2 Laboratory Hardware Facilities

The studio lab developed for these courses consists of twelve student stations and an instructor's station. The instructor's station is configured with classroom control software that enables the capture, control and display of any of the student stations on the classroom video projector. Each student station is positioned to allow a pair of students to work together. Each station has a modern personal computer for software development and a 486-based single board computer as a target system. We are using a Diamond Systems [5] pc-104 board with timers, A/D converters, D/A converters, and digital I/O as our target systems.

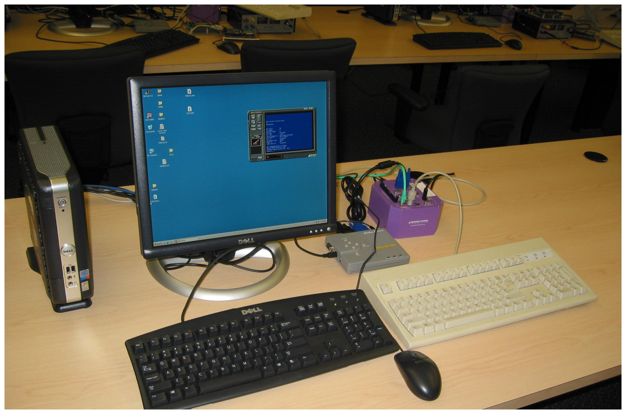

Figure 1. Basic lab station

Figure 1 above shows the basic lab work area for each student group including the development workstation and the embedded "purplebox" target system. To reduce the clutter in the student's work area we eliminated the second monitor often attached to the target system. Students can view the output from the target system in a number of ways. For text-based standard output the target system development software provides a redirected console on the development system. We also have the VGA output converted to S-video and then fed into a USB S-video digitizer. The digitizer's software provides a picture-in-picture display. With the converter's zoom and panning capabilities students see the VGA output. Finally, for projects that are generating VGA graphics output the student can view the full resolution video through the second input channel on the development station's dual-input monitor.

For the experiments involving programming a microcontroller, each station, shown in Figure 2, is provided with a Motorola 68HC12 board, a custom designed interface board on which is mounted the microcontroller board, a custom binary LED-switch board for elementary binary input and output, a signal generator and a power supply. The laboratory currently has two oscilloscopes that are moved from station to station, as needed.

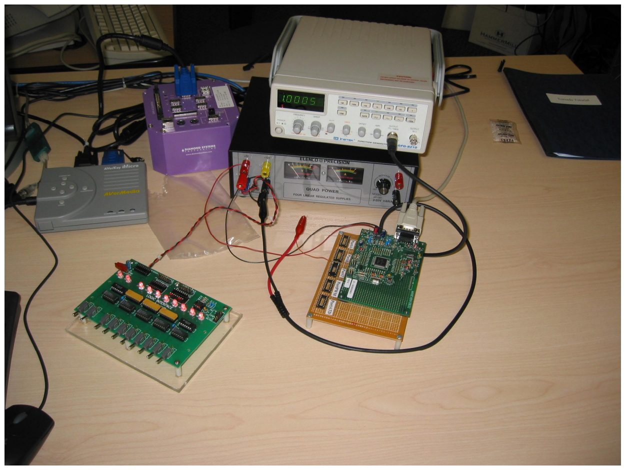

Figure 2. Microcontroller and peripherals

The last pieces of hardware to mention are primarily used in the third course in the sequence. This course covers performance engineering of real-time and embedded systems. To motivate the need for system tuning of real-time systems we use the control of physical systems. The two systems we choose for the laboratory are from Quanser Systems [11]. We selected their inverted pendulum and ball-and-beam systems. The last component of equipment in the laboratory is a Digilent Spartan 3 FPGA board [6]. Also in the third course, the students experiment with hardware/software co-design using this FPGA board. Each student station has one of these boards.

2.3 Laboratory Software Facilities

There is a set of software tools to complement the hardware in the laboratory. The development stations are running the Windows XP Professional operating system. The MGTEK MiniIDE [9] supports assembly language programming on the 68HC12 microcontroller. We received a software grant from Wind River Systems [17] allowing the use of VxWorks and the Tornado development system. We are currently considering the use of the QNX Neutrino operating system environment through a grant from QNX Systems. These are the commercial real-time operating systems that the students use in the laboratory. Matlab and Simulink from The MathWorks [14] are used for simulating and controlling the Quanser experiments. We also received software grants from IBM [8] for the Rational Rose development suite and Rational Rose Real-Time as UML modeling tools. Finally, the students use Rhapsody from Telelogic [13] as a UML modeling tool. Rhapsody's statechart modeling and code generation features are used heavily in the second course in the sequence.

3. AN INTERDISCIPLINARY COURSE CLUSTER

Three courses compose our cluster in real-time and embedded systems. In July 2003, we started work on the laboratory and the development of this course cluster. Each of these upper-division undergraduate courses is four academic quarter credit hours and meets for ten weeks of classes having a pair of two-hour studio sessions per week. We offer each course once in our three-quarter academic calendar.

Each of the courses is cross-listed in computer engineering, and software engineering. The course curricula are delivered in a studio-lab environment where we mix lecture material with hands-on exercises. Registration is initially controlled with the goal of having an even mix between students from the two programs. To the extent possible, we ensure that all project teams have a member from both computer engineering and software engineering. Typically, we have not had a difference of more than two between the student registrations in the two disciplines. Depending on the course project, this is handled by creating one or two teams of three, or having one team of students from a single discipline. For some projects, we will provide additional assistance to a non-interdisciplinary team. These courses are also available to students in the computer science and electrical engineering programs, but we have had only a small number of these students registering in the courses.

3.1 Real-Time and Embedded Systems

The first course in this elective sequence is titled Real-Time and Embedded Systems. It presents a general road map of real-time and embedded systems. It introduces a representative family of microcontrollers that exemplify unique positive features as well as limitations of microcontrollers in embedded and real-time systems. These microcontrollers are used as external, independent performance monitors of more complex real-time systems targeted on more robust platforms. The majority of this course presents material on a commercial real-time operating system (RTOS) and using it for programming projects on development systems and embedded target systems. Some fundamental material on real-time operating systems is also presented. Example topics include: scheduling algorithms, priority inversion, and configuration of a real-time operating system for a target platform and host development system. The textbook for the course is Real-Time Systems and Software by Shaw [12]. This course requires as a prerequisite either a standard Operating Systems course or the software engineering program's course Principles of Concurrent Software Systems. The project work spans the range from microcontroller assembly programming through to application development under a commercial real-time operating system. The topics covered by the Embedded and Real-Time Systems course include:

- Introduction to Real-Time and Embedded Systems
- Microcontrollers
- Software Architectures for Real-Time Operating Systems
- Requirements and Design Specifications
- Decision Tables and Finite State Machines
- Scheduling in Real-Time Systems
- Programming for a commercial real-time operating system
- Development for Embedded Target Systems
- Language Support for Real-Time
- Real-Time and Embedded Systems Taxonomy
- Safety Critical Systems.

The project assignments for this course are:

Microcontroller programming: Students program the 68HC12 microcontroller to act as an interval timer. This assembly language program measures the inter-arrival time of a series of 1000 pulses using the hardware timers available on the processor. Using these timers the students see how to measure with microsecond resolution.

Real-Time Operating System multi-tasking primitives: The main goal for this project is to have the students become familiar with programming under a commercial real-time operating system. Using VxWorks as an example of a commercial real-time operating system, students learn how to program using its concurrency and synchronization primitives. The team must implement a concurrent system such as a transit simulation or an automated factory. The programming had been done within a simulated target system running on the development station.

Real-Time Operating System performance measurements: There are two smaller projects that fall into this category and are run on the target systems. Both projects make use of the microcontroller project as a timing device. In the first project, the students learn how to schedule a periodic task under VxWorks. This task is toggling a bit on the printer port. The microcontroller timer measures the inter-arrival time and jitter of these periodic pulses. The second project measures the interrupt response time of the system by having the microcontroller measure the time between generating an interrupt signal to the target and receiving a response from the target.

Final project: There is a final programming project. This project is usually student motivated with each team thinking of a project. We have seen implementations of user-level drivers for the devices on the target system, an ultrasound distance measurement, simple video games, and a digital oscilloscope.

3.2 Modeling of Real-Time Systems

The second course is titled Modeling of Real-Time Systems. The course takes an engineering approach to the design of these systems by describing the system characteristics via UML models before beginning implementation. This course has the same operating systems course or concurrent systems course prerequisite. Students who take the first course prior to this course have a small advantage, but we have worked to provide sufficient resource materials for students who have not taken it.

The textbook for the course is Doing Hard Time by Douglass [7]. The course covers the following topics:

- Introduction to Modeling of Real-Time Systems

- Basic Concepts of Real-Time Systems
- Basic Concepts of Safety-Critical Systems
- Use case analysis for real-time systems
- Structural object analysis for real-time systems
- Behavioral Analysis using statecharts
- Design patterns for real-time and safety-critical systems
- Threading and Schedulability
- Real-Time Frameworks

This course has the strongest software engineering emphasis. Initially, the projects progressed through phases in the standard waterfall process model with emphasis on analysis and design of the software system. For the software engineering students, this is continued modeling practice using the UML, similar to what they do in all the courses in their software engineering program. The application areas chosen for the projects, i.e. embedded systems, are significantly different from the typical desktop and GUI-over-database projects that they see in their other courses. In this course, the software engineering students took the lead on most projects. Many upper-division computer engineering students have not done any modeling in the UML since their second-year software engineering course.

The strong software engineering emphasis in this course has caused some problems with maintaining computer engineering student enrollment. We do not want to have the situation where the computer engineering students feel that their expertise is not required for most of the projects in this course. With each offering of this course, we have worked to shift the content more toward the computer engineering program. The first course project was a requirements analysis and design assignment. The students received a copy of the user manual for a consumer device. The devices we used included a blood pressure monitor, pedometer, and combined binocular/digital camera. Using the manual, the students identified the actors and use case requirements for the product, and then did a class-level design for an object-oriented implementation of the system. No implementation was done in this project. Our experience is that doing software only requirements moved too far from the computer engineers' expertise. In a recent offering of the course, we dropped the class-level design from this project. We want to keep some aspect of working with requirements because it is crucial to get the requirements correct on any project [2], but this project will require further modifications to bring it more in line with the computer engineering students' interests.

In our experience, the computer science, computer engineering, and software engineering students all gravitate to static class-level modeling with ease. Most of the students feel comfortable creating a design and drawing a UML class structure diagram. Capturing the dynamic behavior of the system is much more difficult for the students. Many real-time and embedded systems have state-based behavior. A significant part of this course discusses UML statecharts as a mechanism for capturing dynamic system behavior.

The second project is a design and implementation project with a major emphasis on manual implementation of the statechart that describes the system behavior. We have used a cyclometer, and a chilled water controller for this project. With the last offering of the course, this project has moved from a standalone application to an application running on our target systems under an RTOS. The program must interface with the FPGA board which provides access to it 7-segment displays, LEDs, buttons and switches. The FPGA development board acts as the user interface controller.

The Modeling course makes extensive use of on-line discussion areas. This is a place where we use "low-stakes" grading of writing assignments that do not carry much course credit. Through multiple offerings of this course, we have also increased the number of in-class exercises that students do. These help reinforce the lecture material that is covered.

3.3 Performance Engineering of Real-Time and Embedded Systems

The third course is titled Performance Engineering of Real-Time and Embedded Systems. The objectives for this course are for students to explore aspects of real-time and embedded systems with an emphasis on measuring their performance. The Performance Engineering course has the Real-Time and Embedded Systems course as a prerequisite. Students without that course who have taken operating systems or concurrent systems have been successful in this course. Based on that experience, we are considering changing the course prerequisites to be the same as for the other two courses. Topics covered by this course include:

- Performance measurements for real-time and embedded systems
- Profiling of program execution in embedded systems
- Exploration of linear control systems
- Interpretation of linear control parameters
- Hardware system description languages
- Hardware/software co-design

The list above is an unusual combination of topics that is not covered in any single textbook. We cover the course topics with class discussions and exercises, handouts, and references to on-line resources for the students. The course splits approximately in half between real-time and embedded topics. We have offered this course three times, and with each offering we have made major modifications trying to achieve the course's objectives. At this point, most of the material from the course's first offering has been replaced with new class material and projects.

The real-time part of the course comes first in the syllabus. Real-time scheduling algorithms are discussed in detail in the first course and briefly in the Modeling course. In this third class, the discussion and exercises review real-time scheduler theory and algorithms including rate-monotonic, earliest deadline first, and least slack time. For their first project, students design and implement a testbed in which they can experiment with several scheduling algorithms. The testbed executes on our target machines running under the real-time operating system (RTOS) environment. We have a class exercise which introduces the RTOS environment and a paper and pencil exercise determining task execution timing for different scheduling algorithms. The students implement their scheduler outside of the operating system kernel because the learning curve for replacing the RTOS's scheduler would be too steep. We have an extensive class discussion about how to structure their testbed to accommodate both fixed priority and dynamic priority scheduling algorithms. This requires careful consideration of the testbed's synchronization mechanism. The students design experiments to test the schedulability limits for several scheduling algorithms comparing their results to the theoretic limits.

The next real-time systems topic is a basic discussion of control systems. The computer engineering students have seen Z-transforms, though only a small number have taken a digital controls course. The software engineering students have none of that background. Obviously, our coverage of real-time control cannot be from a deep control engineering perspective. Instead, we try to provide an intuitive perspective, and have students concentrate on the issues surrounding the implementation of linear control algorithms. A lecture introduces Z-transform notation and students work on a class exercise to implement a simple transform. This is a new experience even for the computer engineers who had a digital controls course which only dealt with the transforms as mathematical entities within Matlab. The project requires students to implement a standard proportional-integral-derivative (PID) controller on the target systems under the RTOS. The plant they are controlling is a simulation running on the development workstation using the Control System Plant Simulator (CSPS) [3]. We have extensive class discussion about how to structure their controller including issues of the timing of analog input and output conversions, and identification of control algorithm values which can be calculated prior to the next time interval. The project requires students to measure controller performance and tune the PID parameters for best performance against the project's stated control goals.

The embedded systems part of the course starts with an introductory lecture on VHDL. A class exercise gets students familiar with the FPGA development environment and walks through a simple VHDL development project. All students, including the software engineers who have done no prior VHDL work, have an individual VHDL development exercise to complete. This is a simple exercise that should take the computer engineering students, who have experience with VHDL, no more than a few hours to complete. We intend the remaining embedded systems work to concentrate on hardware-software co-design, and this is the area where we have had the most difficulty achieving our conceptual goal for this course.

In the first course offering, the students performed a set of JPEG image compressions, first using an all-software approach on the target system, and then off-loading some of the computations to an attached FPGA board. This project requires the strong VHDL experience of the computer engineering students. The image data exchange was through the parallel ports on both the target system and the FPGA development board. We intended that students would make a hardware-software co-design tradeoff by placing more device control functionality in the FPGA. At each step, the students would measure the change in system performance as the boundary between hardware and software was moved. It became clear that the largest challenge was getting reliable communication between the target system and the FPGA. Workarounds for unreliable communication overwhelmed gains made by the hardware implementation of algorithm elements.

We next tried to incorporate hardware-software co-design tools such as System C and Impulse C. We chose Impulse C and received significant support from the tool vendor during the course. The problem was again the communication between the target and FPGA, this time in the form of no Impulse C board support for the parallel port connection that we wanted to use. We asked the students to implement the necessary board support. Even with the extensive vendor support we received, partway through this exercise we realized that it was too daunting a challenge for the students. We quickly assessed the situation, and offered options to the students. One group of four students continued through the remainder of the term to work in the Impulse C environment. Another pair continued working on the project similar to the initial course offering, i.e. outside of the Impulse C environment. The remainder of the students moved onto other projects that we hurriedly created.

We learned two things from these failed attempts at achieving our course concept for hardware-software co-design. First, we needed to eliminate the parallel port's loose coupling between the software processor and the FPGA hardware implementation of algorithm elements. The industry trend, and what is supported by the hardware-software co-design tool vendors, is to embed processor cores in the FPGA. These processors will execute the algorithm elements that remain in software. Second, the students were very motivated to work on projects that were open-ended and where they had some choice in the project details and scope.

For the third offering of this course, we created an undergraduate research-style project which ran through the last four weeks of the term. We defined several topic areas all of which had open-ended project statements that would need further defining. Each student was given an opportunity to state a preference for a project area. The instructor created the final team pairs based on these preferences. Because each project was open-ended, the team's initial task was to define the exact scope, goals, and milestones for their project. While each topic area touched on material covered in class, all the projects had areas that required the students to do extensive investigation beyond that coverage. A project might entail one or more investigations, such as, learning: how to work in a different development environment, how to model a physical system, or how to work with material done by a previous project team. We describe the topic areas next, and include information about what teams accomplished.

Control of physical devices: one student pair was assigned to the Quanser inverted pendulum, and another team to the ball-and-beam system. The project goal was to control these physical devices through the Quanser-supplied interface board, but not through the high-level Simulink interface that Quanser used. The two teams decided to use different approaches. The ball and beam team started with a previous student's project work which used the RTX Real-time Extension for Control of Windows [1]. The team calibrated the sensors and motors, and developed two different control algorithms. The inverted pendulum team installed QNX Neutrino to develop an interface to the sensors and motors, and control the device. Their work with QNX Neutrino will be very useful when we move the entire lab to that RTOS.

PicoBlaze embedded processor: three team pairs chose to explore the PicoBlaze processor core. This low block-count core can be embedded in our small Spartan 3 FPGA devices. Two teams developed an interface to a magnetic card reader, and one team measured distances with two ultrasound distance sensors. The magnetic card reader teams needed to do a lot of research work to understand how to interpret data from the card strip. Neither team was able to reliably read and interpret data, even when one team's member went to the security people at his job and asked them to code several cards with known data for testing purposes. The ultrasound team built an accurate distance measuring peripheral. These projects provided a level of hardware-software co-design experience using embedded processor cores which is an important direction to achieve our objectives for this course.

Hardware-Software Co-Design: As discussed above, our previous attempts at hardware-software co-design, which used a parallel port connection for loose coupling of the software processor and the hardware elements of the algorithm, had mixed success due to problems with communications. One student team took on the challenge to fix these communication problems. They were able to increase the error-free byte transfer rate through the parallel port by a factor of almost 10, and built a framework in the FPGA for easy switching between several different hardware implementations of an algorithm. This will be an improved base to test performance enhancements for a co-design algorithm using the hardware that we currently have in the laboratory.

Real-Time System Simulation and Control: This project asked a team to select a physical system that they could model using the CSPS system [3]. The team would need to provide a graphical user interface for the model simulation which would run on the development station. The team would implement a controller for the plant on the target systems. We could then use this work as the real-time control project in future course offerings. Unfortunately, no students selected to work in this topic area.

Microsoft Robotics Studio: This project area asked a team to explore the capabilities of Microsoft's Robotics Studio [10] as a simulation engine for physical systems. The Robotics Studio has a full physics engine embedded in it. An interface to our data acquisition system could provide a bridge between the physics engine and a target system controller. One team selected this project. They ran into a number of complications just getting the Robotics Studio to run on a development workstation. Once past those problems, the team successfully modeled an inverted pendulum system with characteristics corresponding to our Quanser inverted pendulum, and built a rudimentary controller for it. The team did not have time to do a careful validation of the operation of their simulation to that of the physical device. This work shows promise for future use in the lab if we can overcome many of the installation problems that the team encountered.

We are very happy with the results the students obtained when working on their final undergraduate research projects. This was a win-win project for students and faculty. The students had some control over the choice of their project for the last four weeks of the term. The students were motivated to do the extra investigation work, and did not complain about the vague project requirements. There are two aspects of the assignment that we feel were essential to engage the students. First, the students defined their own project direction under the guidance and final approval of the course instructor. This gave them "buy-in" to the project. Second, we made it clear that as a "research" project it was not known exactly what could be accomplished in the timeframe given for the project. Grading was not going to be strictly based on achievement of specific goals. Each team did specify initial goals for their project, and could still receive a good grade if those goals were not achieved provided that the team demonstrated "due diligence" working on the project. To monitor project work on a weekly basis, each student tracked his or her time on the project along with the tasks accomplished. During class time, each team gave a weekly 5 minute project status presentation to the class describing progress made, problems encountered, and the expected accomplishments for the upcoming week. In addition, the instructor held a private 15 minute meeting with each team once a week to discuss further details of the project progress. It was in these meetings that we were able to assess the due diligence of the team and provide feedback.

The win aspect for the faculty is twofold. By purposely stating the projects as open-ended, we escape the problem, often seen with new projects, of having to guess a reasonable project scope, and provide associated deliverable expectations. The results achieved by teams that met the due diligence requirement show us reasonable expectations. We can then use that information to rework a project into a more traditional close-ended form. Secondly, with judicious selection of the topic areas, we placed some of the burden for exploring new areas on the students. With permission to use their work, we can let future teams build upon it to develop robust project frameworks that we faculty do not have the time to implement ourselves.

4. EVALUATION

We have evaluated the effect of our course cluster using student surveys. Increasing student interest in real-time and embedded systems, and aiding students in finding employment in the area were two goals for our work. The results of all our surveys to-date have consistently indicated success in meeting these goals. We presented evaluation data from the initial offerings of these courses in [4, 16].

As is the case in many organizations, crossing organizational boundaries can create unique problems. We have had our share of them running these interdisciplinary courses. RIT's departments gain credit-hours-generated credit based on the course number. Independent of whether a software engineering or computer engineering faculty member is teaching the course, credit-hours-generated are split between both departments because of the cross-listed numbers. Our department chairs have assumed that this evens out since we have also balanced the responsibility for teaching the courses between the two departments. While the aggregate number of students registered for each course is sufficient, because the courses are cross-listed under multiple numbers, individually, they often set off the course audit warnings for low enrollment in the individual courses. Our department chairs have had to provide explanations for allowing the low enrollments. Finally, it took us a bit of time to get the scheduling coordinated between the departments in separate colleges so that in each term all sections of the course were offered at the same time, in the same room, and with the correct registration limits. Similar coordination problems existed for scheduling concurrent final exams for the courses.

The latest survey has data from fifteen students who took at least one of the three courses. This data represents about a 50% response rate from the students in each course during the 2007-2008 academic year. The numbers of students taking the courses in this cluster were: all three courses – 2; two courses – 6; only one course – 7. Six computer engineering and nine software engineering students responded to the survey. The courses helped to increase student interest in real-time and embedded systems with 87% of the respondents replying Agree or Strongly Agree to the question "These courses increased my interest in real-time and embedded systems." A smaller percentage, 47%, Agree or Strongly Agree that they plan to seek employment in the real-time or embedded systems area. Six students, 40%, Agree or Strongly Agree that taking one of the courses in the cluster assisted the student in getting a co-op or full time position.

We were also interested in the students' perception of the three individual courses, and, particularly, whether the students' found value from the interdisciplinary teaming. These most recent results are shown in Table 1 below. The table rows show data for individual questions asked about each of the courses which are organized in the columns. Each data entry is the number of students who responded Agree or Strongly Agree to the question.

Table 1. Results of survey of student perceptions

	R-T&E Sys.	**Modeling**	**Perf. Eng.**
# of students	8	8	9
Amount learned was worth the time	7	5	8
Recommend to a friend	8	6	9
Adequate preparation	8	7	8
Benefit from teaming	5	5	8

This data shows that the student perception of these courses is positive particularly for the Real-Time and Embedded Systems, and Performance Engineering of Real-Time and Embedded Systems courses. The first course has been well-received by the students from its inception. As described previously, we have made significant changes to the third course to improve the ties of its content to the real-time and embedded systems area. The students particularly liked the final "research" project which allowed them to choose their individual topic area and gave a wide range of flexibility in the direction for each project.

The results for Modeling of Real-Time Systems show some weakness in our opinion. We attribute this to the heavy software engineering emphasis of the modeling aspects of the course which the computer engineering students view as too abstract. Even some software engineering students expressed dissatisfaction because too much was a repeat of modeling that they do in several other software engineering courses. We believe that this indicates the need to make modifications to the basic content, and the nature of the projects. Our move to implement the second project on the target systems running under the real-time operating system was a step in the right direction. We present other ideas for this course in the next section.

The students were almost unanimously of the opinion that the interdisciplinary teams were beneficial in the Performance Engineering course. We attribute that to the very open-ended nature of the project work, particularly the final project which had topic areas that required thinking across the hardware-software boundary. For the first introductory course and the Modeling course, a majority of students still Agree or Strongly Agree that the interdisciplinary teaming is beneficial. The negative student comments that we received center on spending too much time instructing a partner in the other discipline. The direction of the instruction depends on whether the project has a stronger software engineering or computer engineering content. Our retort is that when you assist someone else's learning of a topic it helps solidify your own learning of the material. This does not sway those who may be focused on the amount of work that must be done to get a particular grade. It remains a problem to find projects for all three courses that provide an equal challenge to students from each discipline, and benefit from our interdisciplinary teaming.

5. FUTURE DIRECTIONS

There are two changes that we plan to make in the laboratory's hardware and software infrastructure. A major change will be switching the principal real-time operating system from the Wind River Systems' VxWorks to QNX Neutrino. We made several unsuccessful attempts to create a VxWorks board support package for our new target systems with Pentium-based processor boards. Recently, QNX Software Systems released QNX and their entire development suite in open licensing. The better support provided for QNX on our Diamond Systems targets made it easier to get QNX running on our new processor boards. We now need to retarget/redevelop all our course exercises and projects to this new operating systems environment. The second change we anticipate is moving to new development boards with larger FPGA devices.

There are a number of areas where we feel these courses need improvement. We feel that we got the first course with the right content early on, and it has seen the fewest changes. For now, our own plans for modification are to introduce another physical device control project, such as, controlling the position of a model airplane servo motor with pulse-width modulation.

The Modeling of Real-Time Systems course continues to suffer from too strong an emphasis on the more abstract software engineering modeling techniques. Requirements analysis is one topic that computer engineering students have identified as too abstract. The content of this course still needs to move closer to the hardware. We plan to remove discussion of actors and use cases for requirements analysis. We think that substituting discussion and practice in the specification of specific, well-formed requirements statements for real-time systems will better serve both the software engineering and computer engineering students. Currently, the requirements project uses a consumer device. We think that this project will have more meaning for the students, if we connect the work to the design and implementation second project. Removing some discussion of requirements analysis will provide time for more discussion of real-time design patterns which are very lightly covered now. The third project, which uses the code-generation capabilities of Rhapsody, was executed as a small standalone application. A four-function calculator and a garage door opener are two applications that we have used in the past. We will consider using the same application through the entire term by having the students autocode the application that was used for the requirements, and design/implementation projects.

The Performance Engineering course has undergone the most significant changes from its initial offering. With our latest successes, we are more willing to believe that the concept of splitting the course into two parts, real-time control of physical systems and hardware-software co-design, is sound. However, the course has not yet fully realized that concept. Before adopting a widespread physical systems control project, we need to develop simulations of the inverted pendulum, and ball-and-beam systems. This will allow students to test their controllers in simulation before moving to the physical systems, which are resource constrained.

For the hardware-software co-design aspect of the Performance Engineering course, we also must take major steps to move forward. One student project from last year did solve the problems that we had in communicating via the parallel port between a target system and the Spartan 3 FPGA board. This work should allow a hardware-software co-design project enabling migration of some software components into VHDL implementations on the FPGA. Our goal is still to use a tool such as Impulse C to do high-level algorithm development in C followed by automatic VHDL generation for the components migrated to hardware. We would, however, abandon our initial approach of loosely coupling the software processor and hardware components, in favor of a processor core embedded in the FPGA alongside the components migrated to hardware. This is the trend in industry, and the approach that the design tool vendors support. Our PicoBlaze final projects gave us positive results from students working with this approach, but to move forward we need to acquire development boards with larger FPGAs that can hold more powerful embedded processor cores.

The final project in the Performance Engineering course was well-received by the students, and we will continue these open-ended undergraduate research style projects in future course offerings. We want to improve the projects' focus by reducing the number of topic areas from which students select, and placing additional constraints on some areas to ensure that the new projects build on results of previous work.

This interdisciplinary technique for course delivery has worked so well that we hope to apply it in areas other than real-time and embedded systems. The next area under consideration is an interdisciplinary course cluster in cryptography. This cluster would have three courses and involve the computer engineering, computer science, and software engineering programs. The first course would be a modified existing introductory cryptography course offered by computer science. A second course would concentrate on secure design and implementation of software systems. The third course in the cluster would study hardware-software co-design techniques. These last two courses are new, and would use cryptographic algorithms as the motivation for all course exercises and projects. As we do in our real-time and embedded systems cluster, we would control registration to balance the number of software- and hardware-oriented students.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. DUE-0311269. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] Ardence. RTX - Real-time Extension for Control of Windows. http://www.ardence.com/embedded/products.aspx?id=70. Accessed 12 July 2008.
[2] Brooks, F. P., Jr. *No Silver Bullet - Essense and Accident*. Addison-Wesley, City, 1995.
[3] Chandler, D. and Vallino, J. Control System Plant Simulator: A Framework for Hardware-In-The-Loop Simulation. In *Proceedings of the ASEE 2008 Annual Conference* (Pittsburgh, PA, June, 2008).
[4] Czernikowski, R. S. and Vallino, J. R. Embedded systems courses at RIT. In *Proceedings of the 2005 Workshop on Computer Architecture Education: held in conjunction with the 32nd International Symposium on Computer Architecture* (Madison, Wisconsin, 2005). ACM.
[5] Diamond Systems. http://www.diamondsystems.com.
[6] Digilent. http://www.digilentinc.com.
[7] Douglass, B. P. *Doing Hard Time - Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison Wesley, Reading, 1999.
[8] IBM Rational Software. http://www-306.ibm.com/software/rational/.
[9] MGTEK. http://www.mgtwk.com/miniide.
[10] Microsoft Corporation. Microsoft Robotics Developer Center. http://msdn.microsoft.com/en-us/robotics/default.aspx. Accessed 11 July 2008.
[11] Quanser Systems. http://www.quanser.com.
[12] Shaw, A. C. *Real-Time Systems and Software*. John Wiley & Sons, New York, 2001.
[13] Telelogic. Telelogic Rhapsody. http://www.telelogic.com/products/rhapsody/index.cfm.
[14] The Math Works. http://www.mathworks.com.
[15] Vallino, J. Real-Time and Embedded Systems Course Sequence. http://www.se.rit.edu/~rtembed. Accessed 11 July 2008.
[16] Vallino, J. R. and Czernikowski, R. S. Thinking inside the box: a multi-disciplinary real-time and embedded systems course sequence. In *Proceedings of the Frontiers in Education Conference* (Indianapolis, IN, October, 2005). FIE '05. T3G-12-17.
[17] Wind River Systems. http://www.windriver.com.