Chapter XX

Real-Time Software Course for Undergraduate Computing Programs: Experience Report

ANDREW J. KORNECKI and FARAHAZAD BEHI
Embry Riddle Aeronautical University, Daytona Beach, FL 32114, USA. E-mail:< kornecka,behif>@erau.edu

The paper reviews the authors' experience resulting from offering instruction on development of real-time software over the course of 16 years in undergraduate computing programs. It underlines the role of real-time computing in modern industry and society, the place of real-time systems in undergraduate curriculum, and gives account of the course transition from a preliminary offering with limited laboratory support to the current setting utilizing a host-target development environment with a modern real-time operating system. An assessment methodology conforming to ABET accreditation standards is also described.

INTRODUCTION

Software-intensive systems are often used where time-criticality is the issue and the margin for errors is narrow. Examples include aircraft avionics, air traffic control, space shuttle control, medical equipment, and nuclear power stations. It is imperative that software developers would understand such basic real-time concepts like timing, concurrency, inter-process communication, resource sharing, interrupts handling, and external devices interface.

Graduates of *Computer Science* (CS) and *Software Engineering* (SE) programs are typically employed to develop industry-strength software. *Computer Engineering* (CE) programs are primarily focusing on designing computing systems, often with significant software components. The emphases of these three programs differ: development of new algorithms vs. development of large and complex software systems vs. small embedded software and device drivers. All three cases require good software engineering practices. Regardless which of the three flavors of computing program is considered, the basic tenet of a real-time systems course is development of interactive and time-critical software.

The industry needs graduates with knowledge of dependable time-critical reactive systems and those who understand how the software will interact with the operating system and the environment.

This paper describes rather unique real-time undergraduate course based on the discipline of software engineering, addressing the critical elements of the software development life-cycle, while concentrating on elements critical in real-time applications. The course evolved over the last sixteen years with strong influence from aviation and aerospace stakeholders of the university.

The majority of computing programs, due to constraints imposed by the available laboratory resources, often focus on theoretical foundations and programming skills. Frequently, the background and the interest of computer science faculty, preferring theoretical topics of scheduling and concurrency, inhibits them from teaching about the practical aspects of hardware-software interactions, which are so much in demand by the industry. Conventional computing laboratories limit the potential for students' experiments.

Incorporating the practices of software engineering into undergraduate computing programs, as opposed to teaching conventional computer science courses, is important for the software industry [1, 2]. Introduction of process scripts, requirements, design and code reviews, and metrics familiarizes the students with the software engineering discipline. A rigorous repeatable process helps to create an environment where the software products are developed more efficiently and with fewer defects.

The following sections describe the place and role of real-time in the computing curricula, short history of the course offering, the course assessment, and observations.

PLACE OF REAL-TIME SYSTEMS IN THE UNDERGRADUATE CURRICULUM

ACM/IEEE-CS taxonomy categorizes the body of knowledge of computer discipline into subject areas, knowledge units, and lecture topics [3]. Real-time is treated marginally as an elective topic within the operating systems subject area. The "Computing Curricula 2005 – The Overview Report" [4] uses the term "real-time" only once in the glossary defining computer system engineering as integrating aspects of CE, SE, and CS. The Software Engineering 2004 Curriculum Guidelines [5] identify real-time as one of the topics of Software Modeling and Analysis unit and as a separate specialty (Embedded and Real-Time Systems) – a suggested topic in several application specialties. The Computer Engineering Body of Knowledge 2004 [6] lists Real-Time Operating Systems as one of the topics of the Embedded Systems Knowledge area.

In view of the above, in the practical implementation of the curriculum, computer science programs implement real-time systems as an equivalent of operating systems. In electronics, control, and computer engineering programs, a real-time course is an equivalent of an interfacing course.

Comprehensive real-time system knowledge includes elements of four layers [7]. The specification and design layer describes the software development life cycle phases commencing with the requirements collection and progressing through the early design. The implementation layer includes language constructs, detailed design, and testing (including system prototyping and cross-compiling). The kernel layer introduces specific features of real-time kernels and practical implementation of the system within the hard real-time constraints. The hardware architecture layer addresses the hardware, downloading to the target systems, debugging, and analyzing timing effectiveness of the system.

It is difficult to find an undergraduate computer science/engineering program with a course (or a sequence of courses) covering all four layers. Adequately equipped laboratory and appropriate staffing are required to suitably cover the course material. The faculty needs to have a multi-disciplinary expertise covering elements of software engineering, computer science, computer engineering and often electrical and control engineering. Typically, one with knowledge and experience in one area often lacks background in the other, producing a one-sided curriculum biased toward either computer science (operating systems) or electrical engineering (interfacing and control issues). While the points of emphasis were shifting over the years, the course described in this paper was developed attempting to address all four layers.

Due to rather broad definition of what real-time computing is, a number of undergraduate programs offer a real-time systems course. Sometimes it is a conventional lecture, discussion and analysis of the existing systems (covering everything from avionics instrumentation software to a multi-user airline reservation system). Sometimes it is about implementation of microcontrollers, sensor-based computing and process control. Computer science programs may focus entirely on a scheduling analysis and design of operating system elements. Computer engineering programs would focus on low-level programming and time critical device handlers. The papers by Shaw, Halang, Sanden, Meja and Ferro, Gomaa, and Jacker at the IEEE Workshop on Real-Time Systems Education [8] and by Zalewski [9, 10] show that there is no single accepted standard for undergraduate real-time course offering.

A Short History of Real-Time Course Implementation

This section describes evolution of real-time course from its first offering in 1989 to current implementation [11, 12]. The key objectives of this junior/senior level course require the students to:

- understand the concepts of real-time process and control
- understand the role of the computer as a real-time device
- use established software engineering methodologies for real-time systems
- understand the concepts of multitasking and inter-task communication
- understand issues of time-critical computing
- understand impact of real-time operating system (RTOS) on application software
- have a hands-on experience with development of time-critical real-time systems
- appreciate the role of real-time systems in aviation/aerospace applications

With the progress of the discipline and the availability of technology, four major phases of the course offering can be identified.

Phase 1: Conventional Native Environment – Miscellaneous Projects

The first phase (1989-1990) was conventional lecture course combined with a small team project to produce a real-time software artifact. The course explored the structure, programming, and properties of real-time software, review of software engineering notation and discussion of practical aspects of concurrent processing, timing, scheduling, synchronization, and inter-task communication.

Using domain-specific resources of the Aeronautical Science Simulation Laboratory, student teams developed e.g. air traffic control simulations in Pascal, data acquisition and processing from the Kavouras weather station in BASIC, Silicon Graphics flight simulator with F15 instrument panel, a visual display controlled from a modified Boeing 707 flight procedure trainer in C [13].

Phase 2: Conventional Native Environment – Organized Projects

In the second phase (1991-1993), the course served as the de-facto capstone project for the computer science majors. Students used identical platform and the teams were required to apply more rigorous software process.

At this time the department was using Ada as the primary programming language. For a project, a team of 3-5 students used IBM RS6000 AIX workstations with multitasking/real-time support, creating software life cycle artifacts for a real-time system. The resulting prototypes typically acquired external data and interrupts, reacted to user input, and generated output conforming to pre-defined timing specifications. This setting allowed the students to explore such topics as inter-task communication, resource contention, scheduling, timing, and interfaces with the operating system software. Supporting software was simulating external sources of data and interrupts. The implementation language was Ada95. Examples of developed projects included a dog-fight aviation video game, space shuttle launch control, gas-distributor controller, security station, elevator system, robotic arm control (Figure 1), AWACS radar control panel, a reconfigurable aircraft instrument panel, monitoring airport ground traffic from a GPS feed, etc. [13, 14].

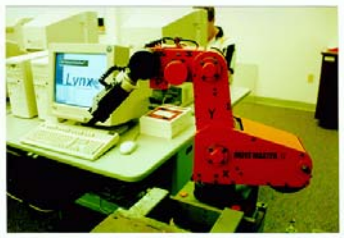

FIGURE 1
ROBOTIC ARM

Each of the three project phases had pre-defined tasks and deliverables. The students were requested to keep logs of their project activities and records of their meeting and co-operative work - applying selected elements of the Personal Software Process [1, 2]. Each phase results were reviewed and discussed, and necessary revisions were requested. The integral part of the project was a prototype demonstration and a formal presentation.

Phase 3: Conventional Native Environment: Ada/C Experiments

The introduction of a senior-level capstone project to the computing curriculum necessitated moving to the third phase of the real-time course (1994-1997). The course was re-designed to a new format using a series of individual (or two-student) lab experiments covering the major real-time concepts using conventional laboratory workstations.

In the early offerings of this phase, several individual assignments were designed to build skills using Ada. Attention was paid to exception handling, modularization, information hiding, and reliability, concurrency, inter-task communication, resource contention, and timing.

Title	Lab Content	Version
Timing	Introduction of the experiment format, real-time lab familiarization, measuring time in application	1,2,3,4
Multi-Tasking	The basic concepts of multi-tasking including calls necessary to manage multiple processes	1,2,3,4
Synchronization /Semaphores	Introduction of shared memory and use of semaphores for synchronization and control access to shared data structures	1,2,3,4
Signals	Basic signal concepts, use of signals to support asynchronous code execution	1,2,3
Scheduling and Priority	Scheduling and prioritizing of the system tasks, round-robin and priority based scheduling, priority inversion and its solutions	1,3
Interfacing with an External Device	Interface with an external device: serial port initialization, read and write between two workstations via null modem	1
Real-Time Data Acquisition	A capstone experiment: real-time data acquisition (reading the data, processing, and storing)	1
Message Passing /Rendezvous	Inter-process communications calls in C (inter-task rendezvous in Ada), supporting data passing, synchronization, and communication	2
Interrupt Service Routines	Implementation of interrupt service routines and responding to external hardware interrupts.	3
Communication/ Message Queues	Inter-task communication designed to convey data between tasks via pipes and message queues	3,4
Timer /Exception Handling	Watchdog timers, exception handling and application-level interrupt processing	4
Capstone Project	Design and implement simple intersection traffic light controller	4

TABLE 1
LABORATORY EXPERIMENTS; THE NUMBERS IN THE LAST COLUMN INDICATE:
1: NATIVE ENVIRONMENT EXPERIMENTS (C ON UNIX/SUN)
2: NATIVE ENVIRONMENT EXPERIMENTS (C AND ADA ON AIX/IBM R6000)
3: HOST-TARGET ENVIRONMENT EXPERIMENTS (C ON VXWORKS/VME MC68K)
4: HOST-TARGET ENVIRONMENT EXPERIMENTS (C ON VXWORKS/ARCOM GX-1)

As a special topic project, an early phase of real-time laboratory experiments was developed in C on a UNIX system (Table 1, entries with #1 in the last column). The capstone exercise was a small data acquisition system [15]. However, since Ada was the language of instruction for the undergraduate student population, these experiments were tested as a special topic, but not in the classroom settings.

Subsequently, the original C/UNIX experiments were adapted to create a series of laboratory experiments designed to teach through examples and exposing students to the laboratory resources with both Ada and C as the implementation languages, on IBM R6000 AIX. The experiments focused on interaction between the operating system and the application program that was either modified or written by the student. A “dual” experiments were designed: each lab provided a sample programs (both in C and Ada) introducing the specific concept (Table 1, entries with #2 in the last column).

Phase 4: Host-Target Environment: Experiments with RTOS

The earlier version of the course incorporated only instruction on two upper layers of RT (specification/design and implementation) introducing a soft real-time system with simulated external data and events. The fourth phase of the course originated with the acquisition of host-target environment: VME Motorola SBC MC68040 target boards running VxWorks and UNIX hosts (1998-2002), supported by National Science Foundation instrumentation grant. Subsequently, the equipment was upgraded to Intel Pentium Arcom SBC GX-1 targets and Window hosts (2002-2006). This course lab experiments were similar to the earlier phase, while extensively using RTOS features in the host-target development environment (Figure 3). The laboratory infrastructure supported thus the "hard" aspects of real-time software development defined by the two remaining real-time layers (kernels and hardware).

FIGURE 2
VME ENCLOSURES

Over the years the experiments were updated and modified. In the initial course offerings, a set of experiments (Table 1, entries with #3 in the last column) for MC68040 targets were designed and posted on a website to be completed by a student during a single semester while learning the appropriate theory component in the classroom [16]. The department received a number of inquiries from industry and a few academic organizations, and the Real-Time Laboratory webpage was reviewed in ACM SIGSOFT

[17] as a “great primer for real-time programming …” and “… series of experiments that demonstrate a wide variety of real-time concepts.”

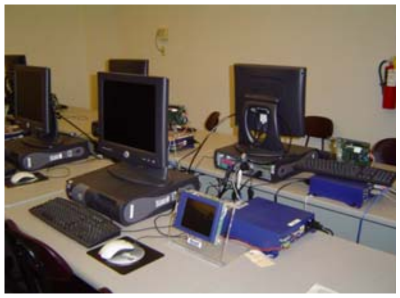

FIGURE 3
ARCOM SBC GX-1

Upgrading the lab hardware to Arcom SBC-GX-1 target board and installing new versions of Tornado IDE and VxWorks required modifications to the existing labs (Table 1, entries with #4 in the last column) including an addition of a capstone Traffic Signal Controller project. The new labs are linked from the University Blackboard course website and accessible to the registered students. The Real-Time website continues to go through upgrades and modifications.

Until transitioning to Phase 4, the course did not require any external funding. Creation of the Real-Time Laboratory was supported by the NSF Instrumentation grant. The subsequent equipment upgrade was supported by the university capital equipment funding request.

COURSE ASSESSMENT

“An understanding of real-time, safety-critical, embedded computer systems” is one of ABET outcomes for both the Computer and Software Engineering programs. The real-time course, designed to contribute to this outcome, is one of indicator courses assessed annually as part of the department’s ABET assessment procedures. Certainly, performance in this course also reflects performance in prerequisite courses: Computer Science II (background in software construction), and Microprocessor Systems (background in hardware operation).

The assessment form filed by the instructor indicates how well the students achieved the associated program outcomes (on a scale 1 to 5). These ratings are discussed at the end-of-term faculty assessment meeting. The assessment provides relatively reliable information about the achievement of the following ABET outcomes:

- the ability to apply knowledge of math, science, and engineering
- design and implementation of software systems or computational processes
- the ability to identify, formulate, and solve engineering problems
- knowledge of contemporary issues in computing
- the use of modern engineering tools, and of course
- real-time systems concepts

The university’s Institutional Research organization conducts a survey of alumni one year after graduation and a survey of employers of those alumni one and a half years after graduation. The survey instruments address general and program-related skills and attributes of alumni on Program Outcomes.

The results of the 2000-2003 surveys were extracted (a sample of 21). The aggregate evaluation in terms of the level of alumni preparation for real-time related tasks is 82% very good/good (with 7% evaluating the preparation as poor). The response to a question regarding usefulness on the job shows 81% respondents answering the course was very or somewhat useful (with 19% not useful).

The department conducts an exit survey of graduating seniors. The students rate the effectiveness of the courses from “poor” to “excellent”. The assessment results for the last three years indicate that the course strongly contributes to the program outcomes and the department’s mission. Some of the alumni feedback after graduation included: “I am glad that I had to take the real-time course and learn the concepts of real-time systems, I used it at work to write programs for Bluetooth devices” and “I am working on a simulator software at work, having done the labs and projects in real-time helped me a lot, specially working with real-time operating system”.

The department’s Industry Advisory Board (IAB), which consists of senior engineering and management personnel of several high-tech companies employing our graduates, has been instrumental in determining initial set of the program outcomes. Each year, the IAB is surveyed about their opinions and provide feedback on the knowledge of program graduates with regard to real-time, safety-critical systems.

OBSERVATIONS AND CONCLUSIONS

Over the years, irrespective of the phase of the course offering, the primary objective of the course was to support undergraduate and graduate students in learning core real-time concepts. The real-time topics fit well with the current curricula for both Software Engineering and Computer Engineering majors. The course is a required component of both these undergraduate programs. Students are introduced to real-time concepts and the laboratory sequence provides hands-on practical experience.

The laboratory experiments motivate students to think not only about the code but also about its implications on the computing environment, timing, performance and interactions with the operating system. The sample programs, in laboratory handouts, give the student solid programming examples to experiment and modify. The course Web site support with handouts and discussion board gives students easy access to the experiments and help to understand the material.

The nature of the real-time course, as presented, has made it hard to find an appropriate single textbook. Most textbooks address a variety of topics, but do not fit with the available tools and the development environment.

The course needs more hands-on and embedded-system type projects with additional hardware interfaces connected to the target system. Adding more safety-critical concepts to the content would also be a good improvement. Other courses in software and computer engineering curricula address the concept and practice of the software engineering process. It would be possible to replace part of the same contents in this course with rudimentary interfacing concepts.

Judging from the feedback from alumni and the employers of our graduates, the real-time laboratory with its hands-on experiments adequately prepares them for similar tasks in the real world. Modern software development requires an environment to teach about reactive and distributed computing, and engage in a teamwork-oriented software engineering process. Many universities can not provide such an environment because they lack either a dedicated faculty or the laboratory infrastructure (or both). Potential solutions to these limitations are:

- Cross interdisciplinary boundaries, utilizing engineering faculty to teach real-time courses (with close cooperation from computer science faculty)
- Use industry personnel to support teaching selected courses
- Use Personal Software Process early in the curriculum
- Use soft real-time with simulated supporting software
- Use an available low cost PC-platform based real-time operating system
- Engage industry to provide funding to develop the laboratory infrastructure

Considering the progress of industry, dedicated, dependable, real-time computing is an integral factor of majority of applications. The experience with the real-time course provides students with an appropriate mix of concepts and skills essential to become a successful real-time software developer. We believe the presented approach to teaching real-time computing meets industry's need for graduates with knowledge and experience with the fours layers of real-time computing, coupled with modern software engineering practices. It is hoped that the approach can serve as an example for other computing programs.

REFERENCES

1. T. Hilburn, I. Hirmanpour, A. Kornecki, "The Integration of Software Engineering into a Computer Science Curriculum," *Proceedings of 8th SEI Conf. on Software Engineering Education*, R.L. Ibrahim, Ed., Springer-Verlag, Berlin, 1995, pp. 87-95.

2. W. Humphrey, "*A Discipline for Software Engineering*," Addison Wesley Publishing Co. Reading, MA, 1995.

3. Computing Curricula 2001, "Computer Science, Final Report, Joint Task Force for Computing Curricula 2005," ACM/IEEE-CS, December 2001

4. Computing Curricula 2005, "The Overview Report, Joint Task Force for Computing Curricula 2005," ACM/AIS/IEEE-CS, September 2005

5. Software Engineering 2004, "Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, The Joint Task Force on Computing Curricula," ACM/IEEE-CS, August 2004. Available: http://sites.computer.org/ccse/SE2004Volume.pdf (accessed May 5, 2007).

6. Computer Engineering 2004, "Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering, The Joint Task Force on Computing Curricula," ACM/IEEE-CS, December 2004.

7. J. Zalewski, "Cohesive Use of Commercial Tools in a Classroom," *Proceedings of 7th SEI Conference on Software Engineering Education*, J.L. Diaz-Herrera, Ed., Springer-Verlag, Berlin, 1994, pp. 65-75.

8. J. Zalewski (Ed.), *Real-Time Systems Education*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

9. J. Zalewski, "A Real-Time Systems Course Based on Ada," *Proceedings of ASEET 7th Ann. Ada Software Engineering Education and Training Symposium*, Monterey, CA, January 12-14, 1993, pp. 25-49.

10. J. Zalewski, "Real-Time Systems Education: An Introduction, *Real-Time Systems Education* (J.Zalewski, Ed.), *IEEE Computer Society Press*, Los Alamitos, CA, 1996, pp. vii-xiv.

11. A. Kornecki, "Real-Time Systems Course in Undergraduate CS/CE Program, CD-ROM Supplement," *IEEE Transactions on Education*, Vol. 40, No. 4, November 1997, pp. 295-296.

12. A. Kornecki, "Real-Time Computing in Software Engineering Education," *Proceedings of 13th SEE&T Conference*, Austin, TX, March 2000, pp. 197-198

13. A. Kornecki, J. Zalewski, "Projects for Real-Time Systems Classes *Real-Time Systems Education* (J.Zalewski, Ed.), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 80-89

14. A. Kornecki, "Global Positioning System as a Real-Time Software Engineering Project in an Undergraduate CS Curriculum," *Proceedings of Conference on Software Engineering Research in Florida*, SERF'93, R. Guha, Ed., Orlando, FL, 1993, pp. 126-129.

15. E. Sorton, A. Kornecki, "Hands-on Software Design," *IEEE Potentials*, Vol. 17 (2), April/May 1998, pp. 42-44.

16. A. Kornecki, J. Zalewski, D. Eyassu, "Learning Real-Time Programming Concepts through VxWorks Lab Experiments," *Proceedings of 13th SEE&T Conference*, Austin, TX, March 2000, pp. 294-301.

17. M. Doernhofer, "Surfing the Net for Software Engineering Notes," *ACM SIGSOFT*, Vol. 30, No. 1, Jan 2005

Andrew J. Kornecki received MSEE'70 and PhD'75 degrees from the University of Mining and Metallurgy in Krakow, Poland. He is a professor in the Computer and Software Engineering Dept. at Embry Riddle Aeronautical University. In 1995, he was funded by a NSF ILI grant to design and implement the Real-Time Laboratory. He contributed to research on intelligent simulation training systems, safety-critical software systems, and served as a visiting researcher with the Federal Aviation Administration (FAA). He has been conducting industrial training on real-time safety-critical software in medical and aviation industries. Recently he has been engaged in work on certification issues and assessment of development tools for real-time safety-critical systems.

Farahzad Behi received the BS'79 in Civil Engineering from North Carolina State University and MSE'85 in Computer Engineering from University of Central Florida. He was a test engineer for Continental Testing Laboratory and a senior software engineer for Siemens Telecom Networks. He teaches Microprocessors, Real-Time Systems, and computer languages. In 2002 he received the Faculty Appreciation Award and, in 1993 and 1994, he received the Best Teacher Award from Seminole Community College's Computer Institute. His interests are in real-time systems and their application for unmanned autonomous vehicles.