Asian Journal of Information Technology 5(5) : 518-521, 2006

Real-time Systems Course in Undergraduate Computer Science/Software Engineering Programs

Imran Anwar Ujan and Lubna Mustafa
Institute of Information Technology, University of Sindh, Jamshoro, Sindh, Pakistan

Abstract: Interactions with industry hiring new software engineers from undergraduate computer science and engineering programs show, case after case, that universities do not pay enough attention to practical aspects of software development. Another well-known deficiency of the undergraduate programs is in the area of time-critical, reactive programming. The present study describes a senior course in a computer science undergraduate program designed to address some of the above problems. The real-time course provides the students not only with the basic concepts of real-time programming, but also provides a vehicle for the development of small class projects which address methods, tools and the critical aspects of a modern software development life cycle. The experience with teaching the course may serve as a model for similar offerings in other computer science, computer engineering and software engineering college programs. The paper describes lessons learned and future plans. The critical observation we submit in this paper is that the development of a real-time system is an exercise in software engineering. It is widely known that real-time software developers are very often self-taught. They represent discipline varying from electrical engineering to nuclear physics. They are frequently brilliant computer hackers, real experts in low level programming and operating system details and wizards of non-standard device interfacing. Too often, however, they do not follow the methodology and established paradigm of the software engineering discipline. Historically, the real-time, embedded systems were small enough for a single developer to work on the entire application. This is not the case in the modern industrial world, where various subsystems of a large time-critical system need to be coupled and synchronized.The paper describes the status of real-time component in the undergraduate computer science/engineering curriculum and comments on the software engineering aspects of the course offering.

Key words: Real-time system, Software Enginreeing,

PLACEOF REAL-TIME SYSTEMS IN THE UNDERGRADUATE CURRICULUM

In ACM/IEEE-CS "Computing Curricula 1991"[Tuck91], the computer discipline is divided into nine subject areas. Each subject area, in turn, is divided into several knowledge units, further subdivided into lecture topics. The topics related to real-time systems are treated marginally as a knowledge unit within the operating systems subject area. In the practical implementation most of the programs consider real-time systems as an equivalent of operating systems (and justifiably so, considering the theoretical underpinning and the basic concepts of concurrency, task scheduling, interprocess communication, etc.). In some electronics, control and computer engineering programs a real-time course is treated as equivalent to an interfacing class, discussing non-standard input/output, device drivers and simple computer based control applications.

A complete and comprehensive picture of real-time system knowledge unit must include elements of all four real-time layers [Zale94]:

- System specification and design
- Host computer implementation
- Downloading and cross-testing on target with real-time kernel
- Testing on independent target with external hardware connections.

The specification and design layer describes the initial software development life cycle phases commencing with the requirements collection and progressing through the early design. The implementation layer takes the next step through language constructs, detailed design and testing. 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 gets to the hardware level, including downloading to the target

Corresponding Author: Engr.Imran Institute of Information Technology,University of Sindh,Jamshoro,Sindh,Pakistan.Mrs.Lubna Mustafa

systems, debugging and analyzing timing effectiveness of the system.

Not too many computer science/engineering programs can provide a course (or a sequence of courses) covering all these aspects. The most important reason is lack of necessary resources. To cover and explore the material properly it is required to have a well-equipped laboratory with elements of the four above-mentioned real-time layers. Another important obstacle is lack of manpower. Real-time systems require unique faculty multi-disciplinary expertise covering elements of software engineering, computer science, computer engineering and often lacks background in the other, producing a one-sided curriculum biased toward either computer sci. (operating system) or electrical engineering (interfacing and control issues). The solution is to promote interdisciplinary co-operation and encourage faculty to cross-teach outside their native departments.

We submit here that the basic tenet of a real-time system class, in a computer science/engineering program, is the development of interactive and time-critical software. Assuming this approach the real-time class must rely on the discipline of software engineering and discuss the critical elements of the software development life-cycle, concentrating on elements critical in real-time applications. Keeping this as the starting point we developed an undergraduate course structure to be used as an example offering in the real-time knowledge unit.III

SOFTWARE ENGINEERING FOR REAL-TIME SYSTEMS

There is a consensus that the software engineering discipline is required in development of any software, regardless of the application. The issues of maintainability, expandability and reliability are of primary importance. These issues are particularly important in development of time-critical and reactive software. Such real-time software is used often in safety-critical applications where the margin of errors is really narrow. Therefore, it is critical to indoctrinate future software developers with the importance of using good software engineering: starting from a requirement and specification document, through design, implementation, testing and maintenance.

Real-time system development has its particularities and there is room for some rapid prototyping and low-level unit testing before the actual design takes place. Lots of practitioners admit that real-time software development is a place of both: an orderly top-down design as well as occasional elements of bottom-up prototyping and implementation of low-level hardware specific interface details. Such an approach is often termed "sandwich development".

The undergraduate computer science curriculum[8] places emphasis on software engineering. The students begin with a three-course sequence (an introduction to programming and computer sci. an introduction to data structures, a course in advanced data structures and algorithms) that uses the Ada programming language and emphasizes principles of software engineering (separation of specification and implementation, modular design, information hiding, reusability, design for maintenance, etc). In the freshman year, they take two computer organization courses (covering the fundamentals of digital logic, computer organization, assembly language and interfacing). In the senior Real-time class, Ada tasking allows for an easy implementation of concurrency and interprocess communication on the language level. It is critical that future industrial application system developers have hands-on experience not only with the basic Ada syntax, but also with tasking, operating systems interactions and low-level programming

REAL-TIME COURSE IMPLEMENTATION

The objective of the real-time systems senior undergraduate course, offered for the computer science majors, is to have students:

- Understand the concepts of real-time process and control
- Understand the role of the computer as a real-time machine
- Represent a real-time system using established software engineering methodologies
- Understand the concepts of multitasking and intertask communication
- Understand issues of time-critical computing
- Be familiar with a real-time operating system and application software
- Have hands-on experience with development of time-critical real-time systems
- Use Ada as the implementation language
- Appreciate the role of real-time systems in aviation/aerospace applications

As part of the class requirements students are developing significant elements of the software life cycle for real-time system. The course introduces the concepts of real-time systems from the user and the designer viewpoints. It explores the connection of external processes to a computer by means of hardware and

software interfaces. The structure programming and basic properties of real-time systems are described with an overview of the system software. Such related topics as interrupts, concurrent task scheduling and synchronization, sharing resources and reliability factors are discussed.

The early part of the class focuses on discussing theory and building necessary skills. The material includes real-time design and programming concepts, with a review of software engineering notation and discussion of the practical aspects of concurrent processing, timing, scheduling and intertask communication.

In the past, the available laboratory resources allowed us to offer the real-time class only within the constraints of "soft" real-tim [Korn93] Instructions was lkimited to the first two rea-time layers (Specification/design and language concepts). The implementation plateform was a network of UNIX workstations with scaffolding software that simulated external sources of data and interrupts. The course has been offered, on average, once each year since 1989. Initially, the student teams worked on a semester-long project divided into three to four phases, each with separate deliverables. The final project report compiled the results of the preliminary phases. An integral part of the project is always a prototype demonstration and a formal project presentation by the development team. The project was dived into three phases of about four weeks each. Each phase had pre-defined tasks and deliverables. To keep track of the process, the students were requested to submit logs of their project activities and records of their meetings and co-operative work. The results for each phase discussed and the necessary revisions were requested.

PHASE ONE: REQUIREMENTS

Task a-Narrative description of the entire system and the environment

- Task b-System context Diagram
- Task c-Description of the system functionality
- Task d-State Transition Diagram detailing the timing behavior
- Task e-Event List and a discussion of timing requirements
- Task f-Data/Control Flow Diagrams detailing information flow

Phase two: Design

- Task g-Process/Control Specifications
- Task h-Data Dictionary identifying the data shared by the subsystems
- Task i-Identification of interfacing processes and data exchange paradigmTask j-Ada Task Graph and packaging decisions

Phase three: Implementation

- Task k-Listing of Ada code, including specifications, bodies and interfaces
- Task l-Description of the hardware/software base used for project implementation
- Task m-Test plan for the prototype
- Task n-Class presentation and demonstration of working prototype.

The final week was spent on report editing, code testing and final project integration. The grading was based on the reports from each of the phases, the final report and the project demonstration. Additionally, the instructor requested, from each team member, a confidential e-mail evaluation of the project in terms of individual contribution of the team members. The student teams coordinate their work though classroom discussions, meetings and documentation exchange (also on e-mail). They used a formal software engineering approach for a real-time system based on the works of Ward/Mellor, Hatley/Pirbhai, Gooma and Nielsen/Shumate [Niel90, Shum92, and Sand94][1-4,7,10].

In the last two course offerings we introduce individual programming assignments dealing with the critical issues of real--time programming (exception handling, Concurrency, intertask communication, resource contention). The team Project was assigned in the last six weeks of classes and the teams worked from artifacts reused from the past course offerings. The project was of shorter duration, consisting of only two phases. The first covered specification and design tasks (a) through (j)-as the teams were given specific requirements , in a form of partial artifacts from tasks (a) through (f). The second phase covered implementation and testing tasks (K) through (m). Each team, however, after consulting with the client (the course instructor) had to prepare an updated version of consistent artifacts in the form of a Context Diagram, Data/Control Diagrams, State Transition Diagrams, an Event List, Ada Task Diagrams, Data Dictionaries, etc. The major part of the project focused on implementation and testing based on the individual programming assignments done in the earlier part of the semester. The students also were required to collect some personal performance data.

LESSON LEARNED AND FUTURE PLANS

From the instructor perspective the major problems in the course were related to the formal process presentation, team coordination, system interfaces and final integration.. Generally the students had a positive attitude toward the course and the project. They generously commented on the complexity, scope and realistic nature of the project. Their criticism focused on the need for maintaining consistency between design and implementation pointing out deficiencies, specifically, in the software development process definition. Also, the students who did not take the software engineering class (not required, but a recommended pre-requisite), struggled with software engineering concepts in the early project development stage.

Modern software development requires an extensive knowledge of both reactive and distributed computing, as well as a team-oriented software engineering process. There is a need to provide an environment to support such instruction. Many universities cannot 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 classes (with close cooperation from computer science faculty)
- Use industry personnel to support teaching selected classes
- Use personal software processes early in the curriculum
- Use soft real-time systems with simulated scaffolding software
- Use an available low cost PC platform based real-time operating system
- Engage industry to provide funding to develop the laboratory infrastructure

We believe that real-time computing is a critical area for the majority of industrial applications. Our experience is that the real-time classes offerings have provide and in the future will provide, the mix of concepts and skills essential for a successful real-time software developer.

REFERENCES

1. Cooling, J.E, 1991. Software Design for Real-Time Systems, International Thompson Computer Press, London, England.
2. Halang, W.A and A Curriculum, 1990. for Real-Time Computer and Control Systems Engineering, IEEE Tranon Educ., pp: 171-178.
3. Hilburn, T., I. Hirmanpour and A. Kornecki,1995. The Integration of Software Engineering into a Computer Sci. Curriculum, Proc. 8th SEI Conf. on Software Engineering Education, R.L, Ibrahim, ED. Spring-Verlag, Berlin, pp: 87-95.