The Role of the Software Engineer in Real-Time Software Development: An Introductory Course

Carol L. Hoover

School of Computer Science, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Internet: clh@cs.cmu.edu

Abstract: A noteworthy feature of the computing industry is the increasing demand for application-specific software. For example, software to control a factory in real-time involves timing, fault-tolerance, and safety constraints. These requirements differ from those one would encounter in the design of a software interface to a database. Software engineers who architect real-time systems need to know how to apply basic principles about real-time computing, but there is a gap between real-time computing theory and industrial engineering practice. The subject of this report is a full-semester course, *Introduction to Real-Time Software and Systems*, which bridges this gap by focusing on the role of the software engineer in real-time software development. This course shows how real-time computing theory can be incorporated into software engineering practice. The report describes the rationale, structure, and content of the course. It also relates the author's experiences in teaching the course.

1 Overview

There is a gap between academic courses which expose students to state-of-the-art real-time computing theory and practical work in software projects [16]. Students who are interested in building real-time software systems need to learn how to incorporate computing theory into software engineering practice. To bridge this gap we have developed the first course, *Introduction to Real-Time Software and Systems*, in a track of real-time courses.

The prevalence of applications such as transportation control, factory automation, strategic defense, and communication systems shows the importance of real-time computing. Advancements in technology have enabled the construction of computerized systems which meet the constraints of real-time applications. Therefore, the incorporation of real-time computing theory into engineering practice is an essential skill needed by hardware and software engineers who architect and build real-time systems as well as by systems engineers responsible for integration, test, and measurement.

Real-time courses at Carnegie Mellon University have concentrated on theoretical aspects of real-time computing such as system resource scheduling and operating system support for real-time applications. In some courses, students construct software components for a real-time operating system or kernel; though more recently, students also build software to support emerging applications in multimedia and real-time communications. In-house training courses at companies such as British Telecom Research Laboratories [39] and Texas Instruments Incorporated [40] focus on the practice of real-time software design.

Introduction to Real-Time Software and Systems is an upper-level undergraduate and professional masters-level course which surveys issues important to real-time software development. It introduces computing theory in conjunction with existing engineering practice and, while doing this, emphasizes the role of the software engineer throughout the real-time software life cycle. The course also provides a way for computing professionals to upgrade their knowledge of state-of-the-art technology.

This report summarizes the distinguishing features of this course and represents the collaborative efforts of the people who worked with the author to create the course.

2 Background and Educational Philosophy

2.1 Rationale

Our course provides an overview of computing issues important to the development of real-time software systems. We believe that software engineers need specific application-domain knowledge (e.g. an understanding of real-time application requirements) in order to architect high quality real-time software systems. Our goal was to create a master's level introductory course that could be followed by more in-depth courses. Follow-up courses would focus on specialized topics such as real-time software design, performance analysis of real-time systems, and real-time applications such as multimedia technology.

We taught a half-semester version of the course in Fall 1992 and a full-semester version in Fall 1993. We plan to offer a full-semester version of the course in Fall 1994. The reader should see Section 5.2: Course Content for more information about the expansion of the half-semester course into a full-semester course.

Our goals in creating this course were to:

- Help students understand the role of software engineers in the development of real-time software.
- Present state-of-the-art real-time computing theory which can be applied to software engineering practice.
- Survey real-time computing issues across all phases of software development including project formulation, requirements definition, design, implementation, integration, test, and performance analysis.
- Bridge the gap between real-time computing theory and software engineering practice.
- "Turn students on to" real-time computing!

We want to motivate students to continue their study of this exciting field.

2.2 Approach

We used an approach similar to Hoover's TAP-D model for creating specialization tracks in graduate software engineering education [20]. First, we identified key knowledge bases and skills needed for real-time software development. These included:

- T - basic computer science theory (e.g. operating systems principles).
- A - application-domain knowledge (e.g. real-time applications requirements).
- P - software engineering practices (e.g. design techniques for modeling real-time software systems).
- D - development of real-time software (to be pursued in subsequent courses).

Then we mapped the knowledge bases and skills into course topics and related educational objectives. We outlined a set of lectures to cover the selected topics. The definition of behavioral objectives enabled us to structure our lectures and assignments in a way that would help students acquire the target knowledge and skills.

Typically we followed one or two lectures about a particular topic with a case study of a real-time application. We chose applications involving issues discussed in the lectures and interesting real-world systems. To acquaint students with state-of-the-art technology and actual engineering practice, we invited guest lecturers who discussed some of the more specialized topics and/or their own industrial experiences.

2.3 Prerequisites and Student Backgrounds

Prior to taking *Introduction to Real-Time Software and Systems*, students should have:

- Knowledge of undergraduate operating systems concepts (e.g. process/task model, system resource management, deadlock, and synchronization).
- Proficiency in one high-level programming language used to develop real-time software (e.g. C, C++, or Ada).

  Proficiency is defined as the ability to read and write programs in this language.
- An understanding of the following mathematical concepts:
  - Ceiling and floor functions
  - Basic proof techniques such as proof by contradiction and induction
  - Rates expressed as ratios
  - Relations
  - Set theory
  - Meaning of the summation symbol

An understanding of the following mathematical concepts is helpful but not required.

- Statistical distributions such as exponential and Poisson
- Summation techniques

Students can acquire prerequisite knowledge and skills from academic courses, through on the job training, or via independent study. Undergraduate courses in applied statistics, discrete mathematics, and logic cover the topics listed above. A handout reviewing important mathematical concepts such as the ceiling and floor functions may help students who have been out of school for awhile.

Masters of Software Engineering (MSE) students, Electrical and Computer Engineering graduate students, Masters in Information Networking students, undergraduate seniors majoring in computer science, and industrial professionals involved in real-time systems development have taken this course. These students have had undergraduate course work and degrees in computer science, electrical and computer engineering, and mathematics.

3 Lecture Topics, Readings, and Educational Objectives

Table 3-1, shown below, provides a detailed list of lecture topics and assigned readings. The abbreviation Add. Ref. precedes optional reading assignments. The subsections following Table 3-1 contain educational objectives for each set of lectures. The reader should see Section 8: References to find a detailed reference for a particular reading. For more information about homework assignments, the reader should refer to Section 4: Assignments and Student Performance Evaluation.

Table 3-1: Summary of Lecture Topics and Readings

Lecture Topics	Readings
1 Introduction	[59]
2 Real-Time Project Formulation: Software Life Cycle Models	Lecture Notes
3 Real-Time Requirements and Specification Methods	[11, 29: Chapter 5]
4 Real-Time Requirements and Specification Methods Continued	Add. Ref.: [2, 3, 13, 17, 32, 43, 65, 67]
5 Scheduling Issues	[42, 61, 63]
6 Case Study: Chimera	Add. Ref.: [7, 14, 15, 33, 58, 62, 68]
7 Rate Monotonic Scheduling (RMS) and Rate Monotonic Analysis (RMA)	[52, 53, 54]
8 RMA Continued	Add. Ref.: [36]
9 Language Issues	[26, 29: Chapter 3]
10 C++ Tutorial	
11 Ada Tutorial	[5]
12 Ada Continued	Add. Ref. [19, 31, 38]
13 Cyclic Executive	[18, 37, 50]
14 Other Software Architectures	Add. Ref.: [6, 27, 47]
15 Real-Time Kernels	[10, 23, 29: Chapter 6]

Lecture Topics	Readings
16 Case Study: Real-Time Issues in Computerized Music	
17 Operating System Issues	[60; 29: Chapters 7 & 8]
18 OS Issues Continued	Add. Ref.: [12, 49, 51, 64]
19 Performance Analysis	[46, 29: Chapter 9]
20 Performance Analysis Continued	Add. Ref.: [1, 21, 48]
21 Hardware Issues, Hardware/Software Interfaces, Systems Integration, and Test	[4, 29: Chapters 2 & 13]
22 Systems Integration Continued	
23 Distributed Real-Time Systems, Reliability and Fault Tolerance	[28; 29: Chapters 11 & 12]
24 Reliability and Fault Tolerance Continued	Add. Ref.: [8, 9, 21, 22, 24, 25, 30, 34, 35, 41, 44, 45, 55, 56, 57, 66]
25 Case Study: Real-Time Communications	
26 Case Study: Radar Applications	
26+ Project Presentations	

By studying concepts presented in the lectures and associated readings, students should acquire the knowledge needed to perform the following actions. Homework problems are designed to sharpen skills related to these behavioral objectives.

3.1 Introduction (Lecture 1)

- Determine if a real-time software solution is needed when given a natural language description of an application's requirements.
- Identify any real-time features when given a natural language description of an application.
- Compare and contrast real-time characteristics of some existing real-time systems.

3.2 Real-Time Project Formulation: Software Life Cycles (Lecture 2)

- Describe essential phases of software development.

- List and define several software life cycle models.
- Suggest disadvantages and advantages of using a particular life cycle model in the development of real-time software.

3.3 Real-Time Requirements and Specification Methods (Lectures 3&4)

- Explain the importance of specification in the development of real-time software.
- Describe how a particular specification technique might be used in the life cycle of real-time software.
- Choose and apply an appropriate formal method for specifying system requirements when given a natural language description of the requirements for a real-time system.
- Analyze and discuss the ability of a chosen specification method to precisely and completely describe the requirements and design of a given real-time system.

3.4 Scheduling Issues and Case Study: Chimera - A Real-Time Operating System (Lectures 5&6)

- Apply basic operating system concepts such as scheduling theory to software solutions for real-time problems.
- Explain the differences between precomputed, static, and dynamic priority scheduling.
- Apply deterministic scheduling techniques.
- Clarify the issue of predictability versus throughput.
- Identify the periodic and aperiodic processes and select an appropriate scheduling technique when given the description of a set of jobs and their timing requirements.

3.5 Rate Monotonic Analysis (Lectures 7&8)

- Use rate monotonic analysis to determine if a set of tasks are schedulable when given the tasks' timing constraints and fixed priorities.
- Discuss the strengths and weaknesses of RMA.
- Discuss the analysis of distributed systems using RMA.

3.6 Language Issues, C++ Tutorial, and Ada Tutorial (Lectures 9-12)

- Describe the differences between languages which are appropriate for real-time software development and those which are not.
- Determine if a particular programming language has features which are necessary for developing real-time programs.
- Discuss the real-time features of several real-time languages.
- Establish criteria for the selection of a suitable programming language for a real-time software application.

Note: In this course, students are expected to become knowledgeable about languages used to develop real-time software but not proficient in a particular language.

3.7 Cyclic Executive and Other Software Architectures (Lectures 13&14)

- Describe software architectures used to design real-time systems such as scheduling tables, compiler-generated scheduling, Ada process model, cyclic model, and blackboards.
- Discuss the relative advantages and disadvantages of each structuring technique listed above.
- Choose an appropriate software architecture for a system to support a given real-time application.
- Design a cyclic executive to meet given application requirements.

3.8 Real-Time Kernels and Case Study: Real-Time Issues in Computerized Music (Lectures 15&16)

- Describe functions of a real-time kernel such as task scheduling, task dispatching, intertask communication, and memory management.
- Identify issues in the design of real-time kernels.
- Discuss the real-time issues involved in the design of software systems to support computerized music applications.

3.9 Operating System Issues (Lectures 17&18)

- Distinguish between real-time and time-sharing operating systems.
- List the differences between basic research and commercial real-time operating systems and name operating systems from each category.

- Explain the advantages and disadvantages of developing a proprietary kernel for a real-time application versus using a commercial product.
- Discuss features of real-time operating systems such as timing facilities, scheduling policies, and synchronization.

3.10 Performance Analysis (Lectures 19&20)

- Identify methods for specifying real-time performance requirements, designing systems to meet these requirements, and verifying performance.
- Explain how hardware characteristics can impact system performance (e.g. interrupt latency).
- Discuss the role of software benchmarks in verifying system performance.

3.11 Hardware Issues, Hardware/Software Interfaces, Systems Integration, and Test (Lectures 21&22)

- Describe basic types of hardware used in real-time systems and characteristic performance requirements for these components.
- Discuss basic hardware/software interface issues for real-time systems.
- Discuss the trade-offs of different hardware configurations when given real-world application requirements as well as the performance and price of basic hardware components.
- Discuss issues involved in integrating hardware and software to produce a functioning real-time system.
- Describe process(es) for integrating real-time software with hardware.
- Explain the role of testing and quality assurance.
- List and describe laboratory equipment and devices used to develop and test real-time systems.

3.12 Distributed Real-Time Systems, Reliability and Fault Tolerance (Lectures 23&24)

- Identify reliability, safety, and fault-tolerance requirements of real-time applications.
- Discuss the impact of reliability, safety and fault tolerance issues on the design of real-time software systems.

- Propose ways to ensure the reliability, safety, and fault tolerance of software systems.
- Explain basic concepts in the design of distributed systems such as: partitioning, allocation, communication, synchronization, concurrency control, and replication as well as the impact of real-time system requirements on these issues.

3.13 Case Study: Real-Time Communications and Case Study: Real-Time Issues in Radar Applications (Lectures 25&26)

Students should be aware of:

- Applications which require real-time communications and issues involved in providing real-time communications.
- Real-time requirements of radar systems and relevant design issues.

3.14 Project Presentations (Session After the Last Lecture)

- Write a paper describing his/her project work and results.
- Present a synopsis of his/her project to an audience of class members and invited guests.

4 Assignments and Student Performance Evaluation

Homework assignments included required reading, reading questions, application problems, and an exploratory project. We designed these assignments to help the student master concepts presented in class lectures and accomplish the educational objectives. We used the written homework assignments and project deliverables to evaluate each student's performance and to provide individualized feedback.

4.1 Readings and Questions

To guide students in their reading and motivate them to do the background reading before the relevant lecture, we required students to answer questions for each set of assigned readings. We designed questions that outlined and clarified important concepts and issues in each reading. Other questions motivated students to relate issues discussed in the readings to their understanding of software engineering practice.

At the beginning of each Tuesday class, we collected the readings questions which had been assigned on the preceding Tuesday and distributed the reading and question assignments. Students were expected to be complete these by the next Tuesday. We graded the written answers that the students submitted.

4.2 Application Problems

Students applied the engineering techniques discussed in class to the solution of real-world application problems. We tried to design challenging problems that stimulated the student to consider and select from multiple solutions. Some topics covered in the application problems were specification methods for real-time software requirements and design, real-time system modeling, resource scheduling policies, and schedulability analysis (in particular rate monotonic scheduling and analysis).

4.3 Exploratory Project

In the full-semester version of the course, we included individualized projects so that students could learn more on their own about class topics. Students received a list of projects and guidelines for defining a project. They wrote project proposals and plans for completing the proposed project deliverables. Some students completed individual projects, while others worked in groups of two. Several students worked on projects under the guidance of researchers at Carnegie Mellon University.

Topics chosen by students in Fall 1993 are listed below.

- Maintaining Global Time in Distributed Real-Time Systems
- Programming Language Features to Support the Development of Reliable and Fault Tolerant Real-Time Software
- Exception Handling in Ada and C++
- Case Study: Specifying Real-Time System Behavior Using Statecharts
- Real-Time Network Protocols
- Case Study: A Practical Guide to the Use of Real-Time Mach[1]
- Case Study: Scheduling Issues in a Car Radio System
- An Experimental Approach to the Measurement of System Overhead and Its Applicability to Rate Monotonic Analysis

1. Real-Time Mach is an operating system kernel developed by researchers at Carnegie Mellon University to support real-time applications. [64]

- Development of a Host-Based Tool for Debugging Real-Time Applications
- Case Study: Issues in the Communication Between Real-Time Mach and Chimera, Two Real-Time Operating Systems
- Real-Time Issues in Robotics

The projects included a review of relevant research papers and an analysis of topics discussed in the papers. Projects such as the case studies involving Real-Time Mach, scheduling issues in a car radio system, system overhead and its applicability to rate monotonic analysis, and a host-based real-time debugging tool involved the creation of software modules to test the ideas being studied. Each student or team of students wrote a project report and presented the project at the end of the semester. We gave students detailed guidelines for writing their project reports and presenting their projects.

4.4 Student Performance Evaluation

Our goal was to help students achieve the educational objectives by providing individual feedback. We gave students the example solutions that we used to grade the reading questions and application problems. The target level of competency was 80% or more of the total possible points. We weighted the points earned for homework assignments and class participation as follows.

- Reading Questions: 40% of the grade
- Application Problems and the Exploratory Project: 50% of the grade
- Class Participation: 10% of the grade

The 40% weighting for the reading questions encouraged students to acquire requisite background information.

5 Lessons Learned

5.1 Course Structure

Students were enthusiastic about the interleaving of background lectures and case studies. They commented that they liked the guest lectures and appreciated the opportunity to learn about the experiences of people who have developed real-time software systems. Some students consulted with guest lecturers whose expertise coincided with the topics of their projects.

We typically used over half of the class to present the lecture topics. During the remainder of the class time, students asked questions, tried practice exercises, and discussed concepts presented in the lecture. We emphasized the application of theoretical ideas to actual software engineering practice; and graduate students who had industrial engineering experience often posed application-level questions during class discussions.

Overall, students completed the required reading before the relevant lectures and asked questions in class about the readings. In order to cover all of the topics outlined in Section 3: Lecture Topics, Readings, and Educational Objectives, we unfortunately had to limit class discussion of the background reading. For that reason, we gave the students answer sheets for the reading questions.

The drawback of answer sheets is that different questions must be assigned to students in successive classes if these students can easily access the answers from students in the previous class. One way to minimize this disadvantage is to organize a collection of papers and related questions for each topic area. The course instructor would select different papers and/or questions for successive classes.

5.2 Course Content

The class of Fall 1992 overwhelmingly said that we needed more case studies and that they enjoyed the guest lectures. The half-semester course did not include an overview of hardware devices used in real-time systems, hardware-software interfaces, or system integration and test. Also, the half-semester version allowed students little time to explore in more depth course topics that especially interested them.

Therefore in the Fall of 1993, we lengthened the course to a full semester by adding lectures to cover the topics listed above and by inviting additional guest lecturers to present case studies in their areas of expertise. We added one week tutorials in rate monotonic scheduling and analysis and in Ada-support for the design and implementation of real-time systems. We also selected relevant chapters from a textbook to fill-in gaps in the course materials and to provide overviews of topic areas [29]. Students pursued self-defined exploratory projects in the full-semester course.

Essential components for creating this course are a set of readings which adequately cover the course topics and several case studies which illustrate real-time software development issues across the software life cycle. Ideally, computing professionals who have built significantly complex real-time software systems should present the case studies. But researchers and/or instructors who have carefully studied a particular real-time application from a software engineer's point of view and who preferably have developed real-time software could help students learn how to transfer real-time theory into practice.

5.3 Future Improvements

Several students from the Fall 1993 class thought we spent too much time on the presentation of software life cycle models and real-time software project formulation. However, they did not think we should remove these topics. Two factors may account for the students' opinion about the time allocated to these topics.

1. Some students taking the course (e.g. the MSE students) had been exposed to these concepts through their industrial experience and in other software engineering courses.
2. Some students in the class were more interested in real-time computing theory than in industrial engineering practices.

We plan to shorten our discussion of these topics at the beginning of the semester and to try to motivate students to appreciate the importance of software engineering practices which enable cost-efficient development of high-quality real-time systems.

We will also review the educational objectives. In particular, we want to modify or delete objectives not adequately covered by the lectures and class assignments. Another issue is the coordination of topics and terminology covered in the readings and class lectures with those presented by the guest lecturers. Even though we gave the guest lecturers a list of topics to be covered prior to their lectures, repetition of topics did occur. We would like to note that some repetition of key ideas helps improve student performance on the class assignments.

6 Conclusions

Former students have commented that this course helped to prepare them for subsequent courses. Three MSE students continued their study of real-time computing. They pursued independent studies of lecture topics that they plan to use when they return to industry. Two undergraduate seniors discussed their class projects during job interviews and obtained positions involving the development of real-time communications systems and multimedia applications.

In the future, we plan to write about the readings questions and homework assignments. Since this course is essentially a survey rather than a project course, we did not discuss in detail the self-defined projects. We will take a slightly different approach in the third version of this course and plan to present the results of this approach later. At that time we would also like to discuss more completely administrative aspects such as class size, creating a course from "scratch", and course maintenance.

As one would expect, creating and teaching a course which surveys the role of the real-time software engineer is challenging and time-consuming. But positive student feedback is a reward for the effort. By writing about her experiences, the author hopes to inspire the reader to become actively involved in the transition of computing theory into software engineering practice.

7 Acknowledgments

The author thanks Cliff Mercer and Jeff Otten for their help in preparing and teaching the first version of this course, in Fall 1992. She acknowledges with appreciation Roger Dannenberg, Nancy Mead, and Mary Shaw for their advice and support.

The interweaving of academic research with industrial practice that characterizes this course is in part due to the contributions of the following people who have been guest lecturers: Paul Crumley, Roger Dannenberg, Jorge Diaz-Herrera, Sam Harbisson, Walter Heimerdinger, Mark Klein, Joseph Kownacki, Nancy Mead, Cliff Mercer, Ragunathan Rajkumar, Daniel Roy, David Stewart, and Jay Strosnider.

8 References

1. Abbott, R.; and Garcia-Molina, H. (1988). *Scheduling Real-Time Transactions: A Performance Evaluation*. Princeton University, Dept. of Computer Science Technical Report CS-TR-146-88, Princeton, N.J.

2. Attiya, H.; and Lynch, N. A. (1991). Theory of Real-Time Systems: Project Survey. *Foundations of Real-Time Computing: Formal Specifications and Methods*. van Tilborg, A. M.; and Koob, G. M., eds. Kluwer Academic Publishers, Boston.

3. Avrunin, G. S.; and Wileden, J. C. (1991). Automated Analysis of Concurrent and Real-Time Software. *Foundations of Real-Time Computing: Formal Specifications and Methods*. van Tilborg, A. M.; and Koob, G. M., eds. Kluwer Academic Publishers, Boston.

4. Bagrodia, R.; and Shen, C. (1991). Integrated Design, Simulation, and Verification of Real-Time Systems. *11th International Conference on Distributed Computing Systems*, May 20-24, pp. 164-171.

5. Baker, T.; and Pazy, O. (1991). Real-Time Features for Ada 9X. *Proceedings of the 12th IEEE Real-Time Systems Symposium*, Dec. 4-6, pp. 172-180.

6. Baker, T.P.; and Shaw, A. (1988). The Cyclic Executive Model and Ada. *Proceedings of the 9th IEEE Real-Time Systems Symposium*, Dec. 6-8, pp. 120-129.

7. Blazewicz, J. (1987). Selected Topics in Scheduling Theory. *Annals of Discrete Mathematics*, Vol. 31, pp. 1-60.

8. Conn, H. C. et al. (1986). *Software Tools and Techniques for Embedded Distributed Processing*. Noyes Publications, Park Ridge, N. J.

9. Crichlow, J.M. (1988). Operating Systems for Distributed and Parallel Computing. *An Introduction to Distributed and Parallel Computing*. Prentice Hall, Englewood Cliffs, N.J.

10. Dannenberg, R.B. (Aug., 1993). Software Support for Interactive Multimedia Performance. *Interface*, Vol. 22, No. 3, pp. 213-228.

11. Davis, A. M. (Sept. 1988). A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM*, Vol. 31, No. 9, pp. 1098-1115.

12. Faulk, S. R.; and Parnas, D. L. (March 1988). On Synchronization in Hard-Real-Time Systems. *Communications of the ACM*, Vol. 31, No. 3, pp. 274-287.

13. Faulk et al. (Sept. 1992). The Core Method for Real-Time Requirements. *IEEE Software*, Vol. 9, No. 5, pp. 22-33.

14. Gonzalez, Jr., M. J. (1977). Deterministic Processor Scheduling. *Computing Surveys*, Vol. 9, No. 3, pp. 173-204.

15. Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; and Rinnooy Kan, A. H. G. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, Vol. 5, pp. 287-326.

16. Haikala, I.; and Marijarvi, J. (1992). (Continuing) Education of Software Professionals. *Proceedings of the Software Engineering Education: SEI Conference*, Oct. 5-7, Springer-Verlag, Berlin, pp. 180-193.

17. Hatley, D. J.; and Pirbhai, I. A. (1987). *Strategies for Real-Time System Specification*. Dorset House, New York.

18. Hayes-Roth, Barbara. (May 1990). Architectural Foundations for Real-Time Performance in Intelligent Agents. *Real-Time Systems*, Vol. 2, Nos. 1&2, pp. 99-125.

19. Hood, P.; and Grover, V. (1986). *Designing Real-Time Systems in Ada*. SofTech, Inc., Technical Report 1123-1.

20. Hoover, C. L. (1993). *TAP-D: A Model for Developing Specialization Tracks in Graduate Software Engineering Education*. Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-93-181, Pittsburgh, Pa.

21. Howes, N.R. (1991). *Real-Time Ada Design Methodologies and Their Impact on Performance*. Institute for Defense Analysis Paper P-2488, Arlington, Va.

22. Huang, J. (1991). *Real-Time Transaction Processing: Design, Implementation and Performance Evaluation*. University of Massachusetts at Amherst, Dept. of Computer and Information Science Technical Report 91-41, Amherst, Mass.

23. Isherwood, D. (1991). Dos and Real Time? *IEE Third International Conference on Software Engineering for Real-Time Systems*, Sept. 16-18, pp. 79-85.

24. Jahanian, F.; and Mok, A. K. L. (Sept. 1986). Safety Analysis of Timing Properties in Real-Time Systems. *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 9, pp. 890-904.

25. Jha, R.; and Eisenhauer, G. (1989). Distributed Ada-Approach and Implementation. *TRI-Ada Proceedings*, Oct. 23-26, pp. 439-449.

26. Kligerman, E.; and Stoyenko, A. D. (Sept. 1986). Real-Time Euclid: Language for Reliable Real-Time Systems. *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 9, pp. 941-949.

27. Krishna, C.M.; and Lee, Y.H. (March 1992). Workshop Report: 1991 Workshop on Architectural Aspects of Real-Time Systems. *Real-Time Systems*, Vol. 4, No. 1, pp. 85-87.

28. Lala, J.; Harper, R.; and Alger, L. (May 1991). A Design Approach for Ultrareliable Real-Time Systems. *Computer*, Vol. 24, No. 5, pp. 12-22.

29. Laplante, P. A. (1993). *Real-Time Systems Design and Analysis: An Engineer's Handbook*. IEEE Press, Piscataway, N. J.

30. Lawson, J. T.; and Mariani, M. P. (1979). Distributed Data Processing System Design -- A Look at the Partitioning Problem. *Tutorial: Distributed Systems Design*, IEEE Computer Society Press, pp. 225-230.

31. Lee, P.; and Nehman, W. (Nov.-Dec. 1991). An Overview of Real-Time Issues and Ada. *ACM Ada Letters*, Vol. 11, No. 9, pp. 83-95.

32. Lee, I.; Davidson, S.; and Gerber, R. (1991). Communicating Shared Resources: A Paradigm for Integrating Real-Time Specification and Implementation. *Foundations of Real-Time Computing: Formal Specifications and Methods*. van Tilborg, A. M.; and Koob, G. M., eds. Kluwer Academic Publishers, Boston.

33. Lehoczky, J. P.; Sha, L.; Strosnider, J. K.; and Tokuda, Hide. (1991). Fixed Priority Scheduling Theory for Hard Real-Time Systems. *Foundations of Real-Time Computing Scheduling and Resource Management*. van Tilborg, A. M.; and Koob, G. M., eds. Kluwer Academic Publishers, Boston.

34. Le Lann, G. (1990). Critical Issues for the Development of Distributed Real-Time Computing Systems. *Proceedings of the Second IEEE Workshop on Future Trends of Distributed Computing Systems*, Sept. 30-Oct. 2, pp. 96-105.

35. Leveson, H.G. (June 1986). Software Safety: Why, What, and How. *Computing Surveys*, Vol. 18, No. 2, pp. 125-163.

36. Liu, C. L.; and Layland, J. W. (1973). Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61.

37. Locke, C. D. (March 1992). Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executive. *Real-Time Systems*, Vol. 4, No. 1, pp. 37-53.

38. Locke, C.D.; Vogel, D.R.; and Mesler, T.J. (1991). Building a Predictable Avionics Platform in Ada: A Case Study. *Proceedings of the Twelfth Real-Time System Symposium*, Dec. 4-6, pp. 181-189.

39. Mann, P.; Mason, A.; and Norris, M.T. (1991). Industrial Training for Software Engineers. *Proceedings of the Software Engineering Education: SEI Conference*, Oct. 7 & 8, Springer-Verlag, Berlin, pp. 99-113.

40. Marchewka, C. (1991). Teaching Software Engineering for Real-Time Design. *Proceedings of the Software Engineering Education: SEI Conference*, Oct. 7 & 8, Springer-Verlag, Berlin, pp. 235-244.

41. McQuown, K. L. (1989). Object Oriented Design in a Real-Time Multiprocessor Environment. *Proceedings of TRI-Ada*, Oct. 23-26, pp. 570-588.

42. Mercer, C. W. (1992). *An Introduction to Real-Time Operating Systems: Scheduling Theory*. Carnegie Mellon University, School of Computer Science Working Paper, Pittsburgh, Pa.

43. Mok, A. K. (1991). Towards Mechanization of Real-Time Systems Design. *Foundations of Real-Time Computing: Formal Specifications and Methods*. van Tilborg, A. M.; and Koob, G. M., eds. Kluwer Academic Publishers, Boston.

44. Mullender, S., ed. (1989). *Distributed Systems*. ACM Press, New York.

45. Muppala, J.; Woolet, S.; and Trivedi, K. (May 1991). Real-Time Systems Performance in the Presence of Failures. *Computer*, Vol. 24, No. 5, pp. 37-47.

46. Park, C.; and Shaw, A. (May 1991). Experiments with a Program Timing Tool Based on Source Level Timing Scheme. *Computer*, Vol. 24, No. 5, pp. 48-57.

47. Parnas, D.L.; Clements, P.C.; and Weiss, D.M. (March 1985). The Modular Structure of Complex Systems. *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 3, pp. 259-266.

48. Puschner, P.; and Koza, C. (Sept. 1989). Calculating the Maximum Execution Time of Real-Time Programs. *Real-Time Systems*, Vol. 1, No. 2, pp. 159-176.

49. Ready, J. F. (Aug. 1986). VRTX: A Real-Time Operating System for Embedded Microprocessor Applications. *IEEE Micro*, Vol. 6, No. 4, pp. 8-17.

50. Salem, F.K.; AL-Rowaihi; and Rodd, M.G. (1991). An Object-Oriented Approach to Modeling Real-Time Software. *IEEE Third International Conference on Software Engineering for Real-Time Systems*, Sept. 16-18, pp. 69-71.

51. Saponas, T. G.; and Demuth, R. B. (1992). The Distributed iRMX Operating System: A Real-Time Distributed System. *Mission Critical Operating Systems*. Agrawala, A. K.; Gordon, K. D.; and Hwang, P., eds., IOS Press, Amsterdam.

52. Sha, L.; and Goodenough, J. B. (April 1990). Real-Time Scheduling Theory and Ada. *Computer*, Vol. 23, No. 4, pp. 53-62.

53. Sha, L.; and Sathaye, S.S. (Jan. 1994). Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems. *Proceedings of the IEEE*, Vol. 82, No. 1, pp. 68-82.

54. Sha, L.; and Sathaye, S.S. (Sept. 1993). A Systematic Approach to Designing Distributed Real-Time Systems. *Computer*, Vol. 26, No. 9, pp. 68-78.

55. Shin, K. (May 1991). HARTS: A Distributed Real-Time Architecture. *Computer*, Vol. 24, No. 5, pp. 25-35.

56. Son, S. H.; Lin, Y.; and Cook, R. P. (1991). Concurrency Control in Real-Time Database Systems. *Foundations of Real-Time Computing: Scheduling and Resource Management*. van Tilborg, A. M.; and Koob, G. M., eds. Kluwer Academic Publishers, Boston.

57. Spector, A. Z. (1989). Achieving Application Requirements on Distributed Systems Architectures. *Distributed Systems*. Mullendar, S., ed., ACM Press, New York.

58. Sprunt, B.; Sha, L.; and Lehoczky, J. P. (June 1989). Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, Vol. 1, No. 1, pp. 27-60.

59. Stankovic, J. A. (1988). Real-Time Computing Systems: The Next Generation. *Tutorial: Hard Real-Time Systems*. Stankovic, J. A.; and Ramamritham, K., eds., IEEE Computer Society Press, pp. 13-38.

60. Stankovic, J. A.; and Ramamritham, K. (July 1989). The Spring Kernel: A New Paradigm for Real-Time Operating Systems. *ACM Operating Systems Review*, Vol. 23, No. 3, pp. 54-71.

61. Stewart, D.B.; and Khosla, P.K. (May 1991). Real-Time Scheduling of Sensor-Based Control Systems. *Proceedings of the IFAC/IFIP Workshop*, May 15-17, pp. 139-144.

62. Stewart, D.B.; Schmitz, D.E.; and Khosla, P.K. (Nov.-Dec. 1992). The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 6, pp. 1282-1295.

63. Stewart, D.B.; Volpe, R.A.; and Khosla, P.K. (1992). Integration of Real-Time Software Modules for Reconfigurable Sensor-Based Control Systems. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 7-10.

64. Tokuda, H.; Nakajima, T.; and Rao, P. (Oct. 1990). Real-Time Mach: Towards a Predictable Real-Time System. *Proceedings of the Usenix Mach Workshop*, pp. 1-10.

65. Ward, S.J.; and Mellor, P.T. (1985). *Structured Development for Real-Time Systems*. Vols. 1-3. Yourdon Press, Prentice-Hall Co., Englewood Cliffs, N. J.

66. Weihl, W.E. (1989). High-Level Specifications for Distributed Programs. *Distributed Systems*. Mullender, S.J., ed., ACM Press, New York.

67. Wood, D.P.; and Wood, W.G. (1989). *Comparative Evaluations of Four Specification Methods for Real-Time Systems*. Carnegie Mellon University, Software Engineering Institute Technical Report CMU/SEI-89-TR-36 (ESD-89- TR-47), Pittsburgh, Pa.

68. Xu, J.; and Parnas, D. L. (1992). Automated Pre-Run-Time Scheduling in Hard-Real-Time Systems. *Proceedings of the Ninth IEEE Workshop on Real-Time Operating Systems and Software*, pp. 91-95.