TAP-D: A Model for Developing Specialization Tracks in Graduate Software Engineering Education

Carol L. Hoover
July 1993
CMU-CS-93-181

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

There is an increasing demand for application-specific software. For example, the software to control a machine on a factory floor is different from the software to manipulate large databases. The software engineer building software to control a motor that powers a piece of machinery needs some understanding of the motor's servo system; whereas a software engineer who designs the software to manage large databases for the NASA Space Station needs specific knowledge about database models as well as the types of data handled on a long-term space vehicle. Specialization tracks within the Master of Software Engineering (MSE) Program at Carnegie Mellon University would enable students to gain application knowledge while developing fundamental software engineering skills. This report proposes a model for establishing specialization tracks within a graduate software engineering program.

Keywords: Real-time applications, software engineering education, and specialization tracks.

Table of Contents

List of Tables

TAP-D: A Model for Developing Specialization Tracks in Graduate Software Engineering Education

1 The Purpose of Specialization Tracks

There is an increasing demand for software engineers who are experts in developing software for specific types of applications. For instance, the software engineer designing software to control a motor that powers a piece of machinery needs some understanding of the motor's servo system and of the way in which analog signals from a motor can be represented as discrete data in a computer. A software engineer designing the software to manage large databases for the NASA Space Station needs specific knowledge about database models, the types of data stored for a long-term space vehicle, and the ways in which this data will be used.

The author uses the term *application domain* to denote a set of applications with common properties that dictate the type of software designed for applications in the domain. For instance, the software to control a machine on a factory floor is different from the software to manipulate large databases. Software that directs the movement of a lathe performs complex mathematical calculations and communicates with custom hardware; whereas database software involves algorithms to format, correlate, and store data efficiently as well as to guarantee data consistency and integrity.[1]

One can define *application domains* by the problems to be solved (applications with similar requirements and common solutions) or by the solutions (computer technology which is applied). In the previous examples, the domain of machine control applications includes those applications with basic motor control requirements. On the other hand, the domain of database applications involves problems which are solved using database technology. An *application subdomain* is a subset of the applications in a particular domain. The set of problems solved using distributed databases is a subdomain of the database domain.

The professional work environment seldom provides software engineers sufficient time to acquire in-depth domain-specific knowledge. In the case of the solution-based domains, the software engineer becomes an expert by developing a thorough understanding of the underlying computer technology and its application. Software engineers with expertise in problem-based domains have in-depth knowledge about the requirements and software solutions for problems in the application domain. For instance, software engineers must understand basic control theory essential to the movement of factory automation equipment before they can specify and design software for machine control.

1. An *application domain* refers to a set of applications with common properties and characteristics which require similar software solutions. *Application domain knowledge* then refers to specific knowledge that a software engineer needs to develop software for applications which are members of a particular domain. For instance, an understanding of strict timing constraints is needed to develop suitable scheduling algorithms for real-time applications.

A *specialization track* is a directed course of study in which one or more graduate software engineering students accumulate and apply knowledge geared towards a chosen domain of applications. For example, a student might specialize in software for factory control applications. Factory control applications are within the real-time domain. Another student may be interested in a more general study of real-time systems. More will be said later about the selection of an *application domain* for a *specialization track*.

In her discussion of computing education for the 1990's and beyond, Mary Shaw identifies an industrial demand for computer specialists with core expertise in computer science as well as in a specialty area such as architecture, astronomy, chemistry, or psychology. She describes the need for joint degree programs which not only enable students to develop core competency in computer science and another field but which also integrate the two fields. One way she says this can be done is by teaching specific computational models and selected techniques from areas of computer science which depend on relevant applications in the joint field. [Shaw90]

In a similar manner, tracks in a graduate software engineering program enable students to develop expertise in specialization areas as well as in software engineering. A *specialization track* integrates the computing theory and practice with the application domain knowledge needed to develop domain-specific software.

Because of the increase in software applications and the growing emphasis on quality and productivity, companies producing applications software prefer to hire software engineers who already have domain knowledge and software engineering skills. It is expected that software engineering graduates who pursue a specialization track should have better employment prospects than software engineers who do not.

In a 1987 article published in *Computerworld*, Allman surveyed hiring trends for computer-related research and development positions. She reported that companies prefer to hire people who have graduate educations and can easily become specialists in a company's area of interest. According to Allman, staffing personnel at Sandia National Laboratories in Albuquerque, New Mexico, seek people with interdisciplinary backgrounds in computer engineering or computer science and mathematics. Sandia needs people with both software development and mathematical expertise to develop computational and cryptographic systems. [Allman87]

The trend in specialization seems to be holding for the 1990's. Despite layoffs and downsizing, high-tech companies in Massachusetts seek people who are hands-on, sharp, individual contributors with specialized skills, wrote Guisbond in a 1990 *Computerworld* article. McMahan, a recruiting director quoted in Guisbond's article, states that those being sought are people with substantial experience in relational databases, workstation software, graphical user interfaces, software engineering, and applications programming in C. He thinks it is very important that people, early on in their careers, pick an area of specialization and develop in-depth skills in that area.[2]

The offering of specialization tracks in a graduate software engineering program is a value-added feature. The student not only acquires expertise in the principles of software engineering but also in a particular application area. The opportunity to pursue a specialization track should therefore enhance the value of a graduate software engineering program. Hopefully, companies will be more willing to fund the graduate education of employees who propose to follow a specialization track which matches the needs of the company.

In an article about training at Texas Instruments, Moore and Freeman discuss Texas Instruments' desire for software engineering graduates knowledgeable in real-time applications. The company's Defense Systems and Electronics Group (DSEG) has identified the understanding of basic real-time systems concepts to be essential to the development of software for defense contracts which provide a majority of its work. After assessing the background and capabilities of 250 newly hired software engineers, DSEG found that very few had any course work which covered embedded real-time systems. DSEG recommended that universities help by offering elective courses in real-time design and programming that cover the basic concepts of real-time computing.

Furthermore, Moore and Freeman suggest that on-the-job training (OJT) is not always a very good means of learning information. They say that OJT tends to be hit-or-miss, is not well organized, and is often incomplete. They also suggest that it may take an employee longer to learn fundamental concepts through OJT than by taking a course about these topics. The DSEG's awareness of these shortcomings has led to the creation of training courses to replace OJT. [Moore88]

The purpose of this report is to specify a goal-oriented process to help a curriculum developer identify and design a specialization track for a graduate program in software engineering. The goal of this process is the creation of specialization tracks which (1) enable students to develop sufficient domain-specific knowledge and skills, (2) are geared towards emerging technologies and industrial demands, and (3) meet student interests and career objectives.

The author intends that this report will direct the creation of specialization tracks in the Master of Software Engineering (MSE) program at Carnegie Mellon University (CMU). The real-time track example and the numerous references to real-time applications are the foundations for the implementation of a real-time track in the MSE program. The real-time track should serve as a model for the development of other MSE tracks.

---

2. Guisbond based her article for Computerworld on interviews with directors and managers of recruiting firms and information systems departments. At the time Guisbond's article was written, Steve McMahan was the managing director at the Boston office of the recruiting firm Source EDP. [Guisbond90]

2 Identifying Potential Specialization Areas

Identifying application domains with high industrial demand for software is accomplished through ongoing awareness not only of technological advances but also of the current status of the software industry. Some sources of information about current and future needs in the information services and technologies industries (particularly applications software) are indicated in the next section. One should reference a number of sources to form an accurate view of the demand for application expertise in the software industry.

As was mentioned earlier, application domains can be based on the use of specific computer technologies or they can be based on problems with common requirements and solutions. The key is to identify areas of expertise that cannot easily be learned on the job. In addition, there should be considerable industrial demand for the expertise and student interest in the specialization area.

Sources of Information about Application Software Needs

- ACM and IEEE publications
- Advertisements for employment
- *Business Week*
- Conference proceedings
- *Datamation*
- Employers of MSE students
- Government reports
- Industrial reports
- Knowledge of emerging technologies that need software support
- Local industry
- Student interests and goals
- Technical journals
- Technical reports
- Technical seminars
- Trade journals
- Trade magazines
- *Wall Street Journal*

Some specialization areas with demands for software engineering and domain expertise are listed below. Application areas are grouped by the type of application domain: as either (1) a problem-based domain of applications or (2) a solution-based domain.

Specialization Areas with Demands for Software Engineering

Problem-Based Application Domains:

- Biomedical engineering applications
- Computer-Aided Design and Computer-Aided Manufacturing (CAD/CAM)
- Computer-Integrated Manufacturing (CIM)
- Industrial controls
- Industrial management
- Management of Information Systems (MIS)
- Real-time applications
- Robotics
- Speech and vision systems
- Telecommunications

Solution-Based Application Domains:

- Artificial intelligence (expert systems and neural nets)
- Computer graphics
- Human-Computer Interaction (HCI)
- Large-scale distributed systems/distributed processing
- Database technologies
- Multimedia technologies
- Networking and communications
- Parallel processing
- Scientific computing

Applications may belong to more than one area of specialization, and major issues may be important to more than one specialization area. For instance, industrial control applications are in the domains of real-time applications and of CIM. CAD/CAM is a subdomain of both the computer graphics and CIM domains, and applications from many specialization areas involve HCI issues. The human-computer interface is particularly important in the design of CAD/CAM software tools.

The scope of a specialization area may be too broad to develop as a track. In that case, a subset of the specialization area may be more appropriate. Computer-integrated manufacturing links all information related to manufacturing (sales projections, sales orders, inventories, production schedules, delivery dates, cost, etc.) to streamline production and improve productivity. A CIM track may not fit into the time frame of the MSE program of study, but a track in CAD/CAM may be feasible.

Student interests and goals as well as industrial trends are important in the selection of specialization areas. The customer of any graduate software engineering program is the student and, if one exists, an employer funding the student's education. Ideally, it should be possible to design a specialization track for an individual student. Tracks based on emerging technologies and on expertise in high demand by industry strengthen the career development of students looking for specialization areas.

3 Selecting Specialization Areas to Become Tracks

In choosing a specialization area suitable for a track, the developer should consider the industrial need for expert knowledge in this area, the career goals of current students, and the requisite resources. Requisite resources include those needed to teach courses, to direct independent studies, and to provide appropriate software development projects.

The following guidelines for selecting specialization areas consider not only the external industrial demands but also the internal structure of the organization offering the track. Each guideline is followed by a short explanation of its meaning and significance.

Guidelines for Selecting Specialization Tracks

1. *The application domain knowledge and/or computer related knowledge needed to design software for an application in the proposed area cannot be learned by interviewing experts in the specialization area or by taking one training course.*

   A specialization track should provide students with extensive application domain and computer-related knowledge that cannot be learned easily on the job.

2. *The application domain knowledge is significantly different from that of other existing specialization areas.*

   Proposed areas whose applications significantly overlap the applications of other areas should be consolidated to decrease the planning efforts. Options for further specialization within a track can be offered.

   For instance, the domain of real-time applications includes the subdomain of industrial control applications. A real-time specialization track could be a prescribed set of courses. Additional or alternative courses and an appropriate project could serve as an optional specialization in industrial control applications within the real-time track.

3. *There are faculty or guest lecturers from industry with expertise in the specialization area and with an interest in teaching the proposed courses.*

   Obviously, a specialization area cannot be offered if there are no instructors available to teach the proposed courses.

4. *The proposed sequence of specialization track courses should provide the student sufficient background to design software for applications in the area.*

   More will be said about this in the section about component knowledge bases.

5. *The courses should fit into the time frame of the graduate program.*

   The MSE program consists of core courses, electives, and a studio project. An MSE student takes a minimum of 6 full-semester, 9-12 unit electives for a total of 54-72 credit units. Those students who do not need to take a directive for remedial study during the first fall semester take 7 electives for a total of 63-84 credit units.

Therefore, a specialization track could feasibly include at most six prescribed 12-unit courses. It may also be appropriate to require at most four prescribed 12-unit courses and allow the student to select related courses and independent studies for the remaining electives. Students could pursue studies beyond the required 54-72 elective units.

4 Component Knowledge Bases: The TAP-D Model[3]

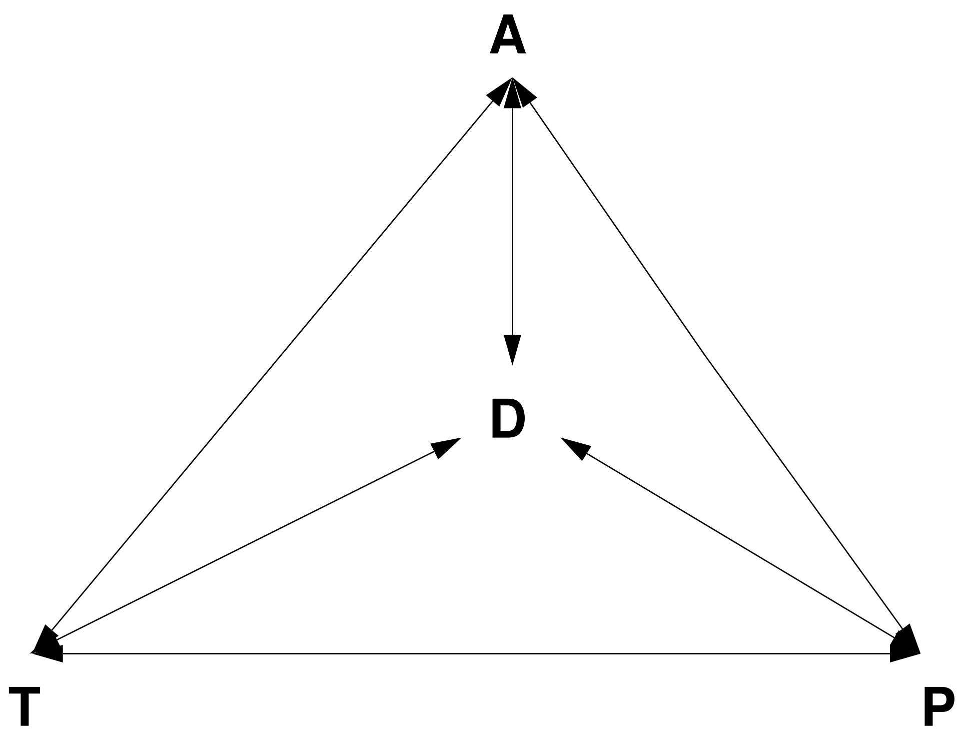

Figure 4-1: The TAP-D Model

T - Theory

A - Application Domain Knowledge

P - Practice

D - Development of a Software System for a Particular Application

In the TAP-D model, the (**T**) represents computing theory which can be used to solve problems

3. The idea of using a triangular structure to model three interacting components which contribute to the actualization of the center component was inspired by Bonnie John's model for Human-Computer Interaction (HCI). In her model, interactions between the human, the available computer technology, and the task influence the design of a human-computer interface. [John92]

in the domain. Consider, for instance, the use of Petri Nets for specifying timing constraints in real-time systems.

Application domain knowledge (**A**) includes non-computer related theoretical and practical knowledge drawn from the application area. An example is digital signal processing theory used in the design of industrial controllers. Likewise, an understanding of geometry and linear algebra is helpful to develop computer graphics algorithms. An understanding of the psychology of learning is required for the development of computer-based tutors.

Practice (**P**) refers to software development techniques geared to the application domain, e.g., the use of commercial tools to support specification and design of real-time systems.

Software developers apply domain knowledge, computing theory, and software development techniques to specify, design, and evaluate software for a particular application. The use of knowledge from one of the component bases affects the use of knowledge from another base.

For instance, a software engineer formally specifying the requirements of a motorized wheelchair for the blind may realize that no one has defined what the software should do if the motor which powers the chair stops running. After speaking to experts in the use of mobile devices for the handicapped, the developer determines that the speech and vision systems should continue to monitor the surrounding environment, should notify the user of the power failure, and should activate emergency flashing lights.

In this case, the interaction between theoretical and domain knowledge (the formal model for specifying requirements and expertise in mobile vehicles for the handicapped) produced a more complete solution. In the process of trying to completely specify the actions of the motorized wheelchair, the software engineer identified incomplete domain knowledge and worked to fill in the missing pieces.

The builder of a track must first define the computer-related theory, application domain knowledge, and practical software development techniques needed to develop software for applications in the specialization area. This is not as simple as it may seem. The builder must not only be knowledgeable in the application domain but must also be well informed about advances in state-of-the-art computing theory and software development techniques.

5 A Structured Approach to Defining a Specialization Track

Defining a specialization track is a sequential process. This process should adhere to basic educational principles. The main idea is to specify behavioral objectives that clearly state what the student should be able to do after successfully completing the track. The following subsections outline the process by which a developer can apply the TAP-D model to define a specialization track.

5.1 Define the Requisite Knowledge and Skills

The first step is to define the knowledge and skills needed by software engineers building software systems for the target application.

Apply the TAP-D model shown above to categorize the required knowledge and skills into their component knowledge bases. Remember that the theoretical understandings, application domain knowledge, and practical skills should be those needed to develop *high quality* software for a particular application.

5.2 Formulate Educational Objectives

The next step is to formulate educational objectives based on the identified skills.

Objectives should be behavioral in the sense that they should clearly define the way in which students will be expected to demonstrate the target skills. For example, an overall educational goal may be to have the student understand and apply formal specification methods suitable for real-time applications. The knowledge base is a set of ways to model requirements of real-time applications software.

The first example below is a behavioral objective to demonstrate that a student understands a set of specification methods.

Example 1:

Given text descriptions of the requirements of various real-time systems, the student will identify appropriate specification method(s) for each application and explain why the chosen specification method(s) is(are) appropriate for the corresponding application.

The accomplishment of this objective can be measured by giving the student text descriptions of requirements of real-time applications and evaluating the student's selections and explanations. Of course, the student would be expected to recognize a case in which none of the specification methods studied in class were appropriate for a particular application.

The next example is a behavioral objective to determine a student's ability to apply knowledge about specification methods.

Example 2:

Given a text description of the requirements for a real-time application, the student will formally model the requirements of the application. The accomplishment of this objective can once again be measured by giving the student a text description of the application requirements and evaluating the student's model.

5.3 Map the Educational Objectives to Courses

Next, map the educational objectives to a set of courses that fit within the time frame of the program of study.

Elective courses in the graduate software engineering program can provide application domain knowledge. Examples given through lectures and class assignments for these electives help students to understand and apply theoretical concepts and techniques to the design of software for the chosen application. At least one elective course, independent study, or project course should require the student to design and implement application software.

Theories and methods presented in the graduate program core courses may apply to applications in the domain. In this case, application examples could be presented in a core course. For instance, the designer of a specialization track for the MSE program should discuss the proposed educational objectives for the track with other members of the MSE faculty to determine if some of these objectives could be integrated with MSE core course objectives.

If educational objectives cannot be met by existing courses, then the designer of the track can propose new courses, or modifications to existing courses, to cover these objectives. For instance, a survey course may introduce a wide variety of topics related to the development of software for the application area. Armed with a general understanding of these topics, students are better equipped to explore these or other related topics in more detail. In addition, a survey course is an appropriate place for guest lectures by people with experience in developing software for the application area.

The designer of a specialization track must consider course availability. Due to conflicting faculty commitments and insufficient student demand, some courses may not be offered every year. In this case, the specification for a track should include alternative courses. Faculty affiliated with the Software Engineering Institute (SEI) should be encouraged to tape their courses so that students could view tapes of courses which are not being offered during the student's residency at CMU. Credit via independent studies may be given for viewing tapes from a previously offered course. But the student should be required to submit one or more reports, other written assignments, or programming projects to demonstrate comprehension of the concepts presented in the tapes. If possible, the instructor giving the taped lectures should supervise the independent study.

5.4 Suggest Appropriate Applications

The last step is to suggest applications appropriate for individual or team projects.

The TAP-D model is based on the idea that a software engineer applies domain-specific knowledge, computing theory, and software development practices to the design of high quality software for a particular application. Therefore, a specialization track should provide students the opportunity to develop a major software system for an application in the specialization area.

A significant software development project can be completed as a course assignment, as an independent software development project taken as an elective, or through a large-scale team project. One major component of the MSE program is the team-oriented studio project which spans sixteen months. The designers of specialization tracks should work with the studio coordinator to arrange appropriate studio projects for students following particular tracks. Students would then form teams with classmates in the same track.

The MSE studio project covers most phases of the software life cycle from requirements specification to product release. Ideally, students will hone their domain-specific skills by working on a studio project related to their specialization areas. These skills should become part of each student's software engineering tool box; and products resulting from the studio experience should become significant artifacts in a student's software portfolio.

6 Example: Development of a Real-Time Track

First, the developer of a track should define the application domain and demonstrate the software development needs for applications in this domain. The required skills should be sufficiently complex that they cannot easily be learned on the job.

The domain of real-time applications include applications whose computing requirements can be met by real-time systems. Burns and Welling cite the following definition of a real-time system.

> Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.

Burns and Welling further explain that timeliness depends on the context of the application: a missile guidance system output is required within a few milliseconds, whereas response time for a computer-controlled car assembly line may be required only within a second. [Burns90]

Laplante adds the risk of a failed system by defining a real-time system as:

> A system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure.

He explains that any system in which response times are restrained and predictable are real-time systems. In this sense, most practical systems are real-time applications. [Laplante93]

Soft real-time systems are systems whose performance is degraded but not destroyed by failure to meet response time constraints. Soft real-time commercial systems, such as automatic banking and airline reservations systems, are designed to provide fast response time but do not fail if response time is delayed. Hard real-time systems, those required for industrial, scientific, and military applications, are characterized by strict time conditions that must not be violated under any circumstances. [Halang90]

Laplante defines the term *firm* real-time system as a system in which the low probability of missing a hard (strict) deadline can be tolerated. [Laplante93] A data acquisition system for a process control application may be firm if it is defined to sample an input sensor at regular intervals but can tolerate intermittent delays.

The importance of *hard* real-time systems used for machine control, air traffic control, air defense systems, and control and safety systems of nuclear power plants, is rapidly increasing. [Halang90] End-use applications of real-time computers span a spectrum that includes transportation systems, robotics and manufacturing, aerospace and defense, industrial process control, and telecommunications. [van Tilborg 91] The real-time track for the MSE program will focus on the specification and development of software solutions to hard and firm real-time applications.

As the requirements of real-time systems become more complex, the systematic design of real-time systems has become an important issue, especially when safety-critical functions are involved. [Mok91] Developing software for real-time embedded systems, those in which computer control is tightly coupled to electromechanical systems, is difficult and challenging. Not only must this software respond in “real-time” and reliably to external events, but it often has to execute under severe hardware and memory constraints. In this application domain, issues of concurrency, reliability, efficiency, and software integration greatly complicate typical software engineering problems. [Atkinson91]

Clearly the widespread demand for software solutions to real-time problems, and the complexities and difficulties of developing these solutions, justifies the creation of a real-time specialization track. The primary goal of the MSE real-time specialization track is to enable students to develop sufficient knowledge and skills so that they can develop high quality software for real-time applications. Though courses will be specifically designed for graduate students of software engineering, other undergraduate and graduate students who have sufficient background may also derive benefit from these courses.

The first step in the specification of a specialization track is to define the requisite knowledge bases and skills that a software engineer should possess to successfully develop software solutions to real-time problems. The next step is to formulate educational objectives that demonstrate student mastery of requisite knowledge and skills. These educational objectives are then mapped to existing or new courses.

The main idea is to determine that the proposed set of courses enable the student to gain the specified knowledge and skills. The specifier of a track might use a table to determine coverage. In the example below, indices help the developer of the track to trace the knowledge base items to the educational objectives and the educational objectives to the courses.

The reader should refer to Appendix A for a list of researchers, faculty, and administrators in the CMU community who are interested in real-time computing. A real-time track team of faculty and administrators currently exists to develop the initial courses for the real-time track. Four core real-time courses will be offered during the 16-month MSE program. Students may take other real-time courses or complete real-time independent studies related to real-time computing.

Further specialization in subdomain areas such as multimedia applications, real-time communications, real-time operating systems, or industrial controls may be offered as options within the real-time track. This example defines an industrial controls option within the real-time track.

6.1 Requisite Knowledge Bases and Skills

Designers of high-quality, real-time software systems use a combination of computing theory and domain knowledge. They also apply practical software development techniques to design and implement these systems. Only those knowledge bases which apply specifically to real-time systems are listed. The reader should note that the knowledge bases listed below are examples. The author intends that curriculum developers modify and expand them as appropriate for their own programs.

6.1.1 Computing Theory

A software engineer needs an understanding of the following concepts to specify, design, and implement software for real-time applications. Traditionally, students learn about many of these concepts in computer science courses which discuss operating systems, formal specification methods, and distributed systems.

T1. Awareness of the importance of specification in the design and analysis of real-time systems.

T2. Classification of specification methods by development phases versus classification by representation. Criteria for choosing a method to specify real-time system behavior.

T3. Formal techniques for specifying real-time software system requirements and design. These methods should include the specification of timing and concurrency requirements.

T4. Formal techniques for verifying real-time software system requirements and design specifications.

T5. Awareness of the shortcomings of existing methods for specifying real-time systems behavior.

T6. Basic operating system concepts which apply to software solutions for real-time problems such as:

- asynchronous/synchronous events
- context switching/stack model
- deadlock handling and prevention
- exceptions and exception handling
- foreground/background systems
- interrupt-driven systems and interrupt handling
- intertask communication and mailboxes
- kernel or executive programs
- memory management
- preemptive priority systems
- process/task model
- round-robin systems
- scheduling
- synchronization of resource usage, critical regions, and semaphores
- timing requirements/deadline characteristics

T7. More advanced scheduling concepts and trade-offs such as:

- deterministic scheduling
- pre-computed versus static versus dynamic process priority
- predictability versus throughput
- periodic and aperiodic processes
- priority inversion
- rate monotonic systems/analysis

T8. Basic concepts in the design of distributed systems such as:

- allocation of resources
- communication protocols
- concurrency control
- deadline scheduling
- distributed control algorithms
- latency versus throughput
- partitioning
- replication and reliability
- synchronization across multiple processors

T9. How real-time and time-sharing operating systems differ.

T10. How research and commercial real-time operating systems differ.

T11. Languages that support real-time computing/specific features.

T12. System performance analysis involving:

- CPU utilization
- interrupt latency
- memory utilization
- response time calculation

6.1.2 Application Domain Knowledge

A software engineer developing real-time software systems should have knowledge in the following areas.

A1. Common characteristics of applications which require real-time software solutions. Difference between applications which require real-time solutions and those which only require adequately fast response times.

A2. Examples of real-time application areas and their specific characteristics; knowledge about application subdomains of the real-time domain.

A3. Reliability, safety, and fault tolerance issues pertaining to real-time systems.

A4. History of real-time computing including real-world examples.

Industrial Controls Option

AA1. Basic control theory.

AA2. Digital signal processing (DSP) and filters.

- Analog-to-digital conversion: sampling issues
- Data representation and precision

AA3. Characteristics of basic feedback devices

- encoders, resolvers, other servomechanisms
- absolute versus relative positioning devices

6.1.3 Software Development Practices

The following topics apply to the design and development of real-time software systems. In choosing appropriate methods, the software engineer should be aware of the advantages and disadvantages of applying a particular method to solve a problem in real-time computing.

P1. Common misconceptions about real-time computing.

P2. Semi-formal methods for specifying and verifying real-time software systems requirements and design.

P3. Commercial tools to support requirements and design specification and verification.

P4. Cyclic executive solution to scheduling requirements.

P5. Fixed priority-based solution to scheduling requirements.

P6. Ability to understand and apply guidelines for using rate monotonic analysis (developed by research staff at the SEI in Pittsburgh, Pa.)

P7. Real-time software architectures: scheduling tables, compiler-generated scheduling, Ada process model, cyclic model, and blackboards.

P8. Designing reusable, real-time software components (e.g. object-oriented design of real-time components).

P9. Choosing and using a language for a particular real-time application.

P10. Customizing a commercial real-time operating system versus developing a proprietary kernel.

P11. Techniques for designing distributed real-time software systems.

P12. Techniques for designing reliable, safe, and fault-tolerant real-time software systems.

P13. Methods for specifying real-time performance requirements and for designing systems to meet these requirements.

P14. Issues in the design of hardware/software interfaces.

P15. Hardware/software integration.

P16. Techniques for testing real-time systems: functional, stress, unit/system level, black box, white box, formal program proving, cleanroom, and statistically-based testing.

P17. Methods to measure real-time system performance.

- logic analyzers
- in-circuit emulators
- hardware simulators
- software benchmarking

P18. Verifying the reliability, safety, and fault tolerance of real-time software systems throughout the software life cycle.

P19. Methods to improve real-time software performance (e.g. compiler optimization techniques).

P20. Hardware characteristics which affect response time. Ways to reduce response times.

P21. Trade-offs involved in selecting hardware components for real-time systems.

P22. Implementation of hardware/software interfaces to external devices such as industrial machines; development of device drivers.

Industrial Controls Option

PP1. Software implementation for sampling techniques.

PP2. Software implementation of techniques to handle data representation issues.

PP3. Software implementation for filtering techniques and digital signal processing applications.

PP4. Software implementation of feedback loops.

6.2 Educational Objectives

The educational objectives are designed to demonstrate a student's mastery of the requisite knowledge and skills. The indices in parentheses after each objective reference the corresponding knowledge base items.

E1. Given a natural language specification of the requirements for a system, the student should be able to explain whether a real-time software solution is needed. **(A1, A2, P1)**

E2. Given a natural language description of an application, the student should be able to identify any real-time aspects of the application. **(A1, A2)**

E3. The student should be able to compare and contrast real-time aspects of some existing real-time systems. **(A4)**

E4. The student should be able to explain the importance of specification in the development of real-time software. **(T1)**

E5. Given the description of a specification technique for real-time systems, the student should be able to explain how the method might be used in the software life cycle. **(T1, T2)**

E6. Given a natural language specification of the requirements for a real-time system, the student should be able to choose a more formal method for specifying system requirements and apply it. **(T3, P2)**

E7. The student should be able to informally or formally verify specifications for real-time software systems requirements and design. **(T4, P2)**

E8. Given the specification for the requirements or design of a real-time system, the student should be able to analyze and discuss the ability of the chosen specification method to precisely and completely describe the requirements and design of the system. **(T5)**

E9. The student should be able to describe and preferably use available commercial tools that assist in the specification of software system requirements and designs. **(P3)**

E10. The student should be able to apply basic operating system concepts such as scheduling theory to software solutions for real-time problems. In particular, the student should implement software components to perform basic kernel tasks such as scheduling; dispatching; intertask communication; and interrupt, exception, or event detection and handling. **(T6)**

E11. The student should be able to explain the differences between precomputed, static, and dynamic priority scheduling. **(T7)**

E12. The student should understand deterministic scheduling techniques and be able to clarify the issue of predictability versus throughput. **(T7)**

E13. Given the description of a set of jobs and their timing requirements, the student should be able to identify the periodic and aperiodic processes, to devise an appropriate scheduling technique (i.e. cyclic executive versus priority-based) and to use rate monotonic analysis to determine if the set of processes is schedulable. **(T7, P4, P5, P6)**

E14. The student should be able to distinguish between real-time and time-sharing operating systems. **(T9)**

E15. The student should be able to enumerate the differences between basic research and commercial real-time operating systems and to name operating systems from each category. **(T10)**

E16. The student should be able to delineate the advantages and disadvantages of developing a proprietary operating system for a real-time application versus using a commercial product. **(P10)**

E17. The student should be able to describe the following software architectures used to design real-time systems: scheduling tables, compiler-generated scheduling, Ada process model, cyclic model, and blackboards. **(P7)**

E18. The student should design real-time software components that are easily reused. **(P8)**

E19. Given the description of a real-time software application, the student should be able to select an appropriate software architecture for the system. **(P7)**

E20. Given the description of a programming language, the student should be able to determine if the language is appropriate for developing real-time programs. **(P9)**

E21. The student should be able to discuss the distinguishing constructs of several real-time languages. **(T11)**

E22. The student should be able to establish criteria for the selection of a suitable programming language for a real-time software application. **(P9)**

E23. The student should be proficient in the use of at least one real-time programming language. Proficiency may be defined as the ability to use a language’s real-time features correctly. **(P9)**

E24. Given the description of a real-time application, the student should be able to identify reliability, safety, and fault-tolerance issues relating to the application. **(A3)**

E25. The student should be aware of the impact of reliability, safety and fault tolerance issues on the design of real-time software systems and should be able to propose ways to ensure the reliability, safety, and fault tolerance of software systems. **(P11)**

E26. Those students who develop a software system for a real-time application should verify that their requirements and design specifications as well as their implementations incorporate features to ensure reliability, safety, and fault tolerance. **(P18)**

E27. The student should be able to discuss basic hardware/software interface issues for real-time systems. **(P14)**

E28. The student should be able to integrate hardware and software components. **(P15)**

E29. The student should be able to explain basic concepts in the design of distributed systems such as: partitioning, allocation, communication, synchronization, concurrency control, and replication. The student should also be able to discuss the impact of real-time system requirements on these issues. **(T8)**

E30. Those students who choose to develop software for a distributed real-time application should be able to design software modules which not only support real-time requirements but also address the issues mentioned in the previous objective. **(P12)**

E31. The student should be knowledgeable about methods for specifying real-time performance requirements, designing systems to meet these requirements, and evaluating performance. The student should have an understanding of how hardware characteristics impact performance (e.g. interrupt latency). **(T12, P13, P20)**

E32. The student should be able to use software optimization strategies to improve real-time system performance. They should be able to evaluate the performance of their software components. **(P17, P19)**

E33. The student should be knowledgeable about different ways to test real-time systems and should be able to apply at least one method. **(P16)**

E34. Given real-world specifications of hardware performance and prices and application requirements, the student should discuss the trade-offs of different hardware configurations. **(P21)**

E35. The student should implement software components to interface with one or more external devices. **(P22)**

Industrial Controls Option

EE1. The student should demonstrate knowledge of simple feedback (closed versus open loop) control systems. **(AA1)**

EE2. The student should be able to propose solutions to sampling issues involved in analog-to-digital conversion. **(AA2)**

EE3. The student should be able to identify the problems in representing data for control systems accurately and precisely in a computer. **(AA2)**

EE4. The student should be able to propose solutions to data representation problems. **(AA2)**

EE5. The student should demonstrate knowledge of digital signal processing (DSP) fundamentals. **(AA2)**

EE6. The student should demonstrate knowledge of basic filtering techniques. **(AA2)**

EE7. The student should be able to describe differences between basic feedback devices. **(AA3)**

EE8. The student should be able to design software implementations of basic sampling techniques. **(PP1)**

EE9. The student should be able to design software solutions to problems in representing control data precisely in a computer. **(PP2)**

EE10. The student should be able to design software implementations of basic filtering techniques used in digital signal processing applications. **(PP3)**

EE11. The student should be able to design software to implement a feedback loop for a real-time control system. **(AA3, PP4)**

6.3 Outline of Courses

Students entering the Masters in Software Engineering (MSE) program who are proficient in at least one high-level language used to develop real-time software (e.g. Ada or C) and who have a basic understanding of operating systems can enroll directly in the *Introduction to Real-Time Software* course. A recommended, but not required, prerequisite for entering the real-time track is knowledge of at least one assembly language. Students with insufficient background can take courses at the undergraduate level or study independently to acquire the necessary skills.

As an introduction to real-time computing and software issues, the *Introduction to Real-Time Software* course covers many of the educational objectives specified earlier for a real-time track. This course, incorporating a combination of lectures, readings, and related assignments, provides a broad, conceptual basis for further study of real-time computing and software development.

In the first offering of this course, all students could program proficiently in at least one language used to develop real-time systems software. One student took an undergraduate course in operating systems and the real-time introductory course during the same semester. All students in the first class performed well and thought the course helped them to gain a better understanding of the issues involved in developing software for real-time systems.

The *Introduction to Real-Time Software*, *Real-Time Software Design*, *Real-Time Software Performance*, and *Real-TIme Applications Laboratory* courses are the foundation of the real-time track. The latter three of these courses are in the planning stage and will be available during the next school year if sufficient demand for them exists. They emphasize a hands-on approach to understanding real-time computing. Students design, implement, and test real-time software in these three courses. The *Real-Time Software Performance* and *Real-Time Applications Laboratory* courses enable students to experiment with different hardware/software interfaces. The *Real-Time Applications Laboratory* course will offer advanced application of concepts presented in the *Introduction to Real-Time Software* course.

Students who seek further hands-on understanding of real-time computing at the level of the hardware/software interfaces might consider *Concurrency and Real-Time Systems*. Students have the opportunity to build an operating system kernel for a target architecture and to develop device drivers in this course. The prerequisites include an understanding of the fundamentals of computer architecture and digital logic. Students with sufficient background may be able to waive the prerequisite courses and should therefore contact the Department of Electrical and Computer Engineering.

Table 6-1 contains information about the courses that students can take to develop skills in understanding real-time computing issues and in developing software for real-time systems. The table also shows prerequisite courses. The * symbol indicates that the course is offered every year, and the + symbol indicates that the course is offered when there is sufficient demand.

Table 6-1: Real-Time Track: List of Courses

Course Name	Educational Objectives	Course Number	Prerequisite Course(s)	Term Offered	Units
Intro. to Programming & Comp. Sc.	Prerequisite	15-127	None	Fall* Spring*	12
Fundamental Structures of Comp. Sc. I	Prerequisite	15-211	15-127	Fall* Spring*	12
Fundamental Structures of Comp. Sc. II	Prerequisite	15-212	15-211	Fall* Spring*	12
Operating Systems	Prerequisite	15-412	15-212	Fall* Spring*	12
Introduction to Real-Time Software	E1-E17, E20-E24, E24,E29,E33	17-880	15-211 15-412 or Equivalents	Fall+	6/12[1]
Real-Time Software Design	E9, E10, E18-E25, E30, E31	To Be Assigned	17-880	Spring+ Summer+	6
Real-Time Software Performance	E31-E34 & Advanced Concepts	To Be Assigned	17-880	Spring+ Summer+	3
Real-Time Applications Laboratory	E10-E13, E24-E30,E35 Ad. Concepts	To Be Assigned	17-880	Fall+	12
Distributed Systems	E24-E26 E29, E30	15-612	15-412 or Equivalent	Spring*	12

Course Name	Educational Objectives	Course Number	Prerequisite Course(s)	Term Offered	Units
Intro. to Electrical & Comp. Eng.	Prerequisite	18-100	None	Fall* Spring*	12
Fundamentals of Computer Engineering	Prerequisite	18-240	18-100	Fall* Spring*	12
Concurrency & Real-Time Systems	E10, E24, E25, E28, E30, & E31	18-349	15-211 18-240	Spring*	12

[1]The course offered in Fall 1992 was 6 units. A 12-unit *Introduction to Real-Time Software and Systems* is under development.

Industrial Controls Option

MSE students interested in control theory and/or digital signal processing (DSP) can find a variety of courses offered by the Electrical and Computer Engineering Department (ECE). Those students who already have fundamental programming and electrical engineering skills (including signals and systems) are best prepared to pursue a controls option under the real-time track. The courses shown in Table 6-2 are members of one of the following four categories: (1) courses which are prerequisites for other courses, (2) courses whose content covers the educational objectives specified earlier, (3) courses whose content covers some of the previously listed educational objectives and are prerequisites for other courses, and (4) courses which cover advanced DSP or control theory.

To select a sequence of courses, the student should consider course prerequisites as well as the term(s) in which a course is offered. The symbols *, **, and + indicate the frequency in which courses are offered.

* The course is offered every year.

** The course is offered every other year.

\+ The course is offered when there is sufficient demand.

For example, a student with good programming and basic electrical engineering skills who is interested in developing software for DSP applications might take *Introduction to Real-Time Software* and *Digital Signal Processing I* during the fall followed by *Digital Signal Processing II* (if it is offered) in the spring. If *Digital Signal Processing II* is not available, the student could take *Concurrency and Real-Time Systems* and/or other electives from Table 1 or Table 2 in the spring. Another student with a basic understanding of signals and systems may be interested in control theory and could take *Introduction to Real-Time Software* and *Fundamentals of Control* in the fall followed by *Real-Time Software Design* and *Real-Time Software Performance* in the spring. The student could then complete a summer independent study involving an industrial control application and follow this with the *Real-Time Applications Laboratory* course and *Linear Systems* or *Digital Signal Processing I*.

Table 6-2: Industrial Controls Option: List of Courses

Course Name	Educational Objectives	Course Number	Prerequisite Course(s)	Term Offered	Units
Calculus	Prerequisite	21-121	None Listed	Fall* Spring*	10
Calculus in Three Dimensions	Prerequisite	21-259	21-121	Fall* Spring*	9
Numerical Methods	EE3, EE4	21-369	21-259	Fall* Spring*	9
Introduction to Numerical Analysis I	EE3, EE4 Advanced Topics	21-660	None Listed Contact Instructor	Fall*	12
Intro. to Electrical & Comp. Eng.	Prerequisite	18-100	None	Fall* Spring*	12
Fundamentals of Electrical Engineering	Prerequisite	18-220	18-100	Fall* Spring*	12
Signals and Systems	EE2, EE6 and Prerequisite	18-396	18-220	Spring*	12
Fundamentals of Control	EE1,EE7, EE9,EE11and Prerequisite	18-370	18-396	Fall*	12
Control Systems Design	EE7, EE9, & EE11	18-475	18-370	Spring*	12

Course Name	Educational Objectives	Course Number	Prerequisite Course(s)	Term Offered	Units
Digital Signal Processing I	EE5, EE6, EE8, EE10	18-791	18-396	Fall*	12
Digital Signal Processing II	Advanced DSP Topics	18-792	18-791	Spring**	12
Linear Systems	Prerequisite	18-771	18-396	Fall*	12
Multivariable Control Systems	Advanced Control Topics	18-772	18-370 18-771	Spring+	12
Adaptive Control	Advanced Control Topics	18-773	18-370 18-771	Fall**	12
Optimal & Stochastic Control	Advanced Control Topics	18-775	18-370 18-771	Fall**	12

6.4 Applications for Independent Study or Team Projects

Ideally, students following the real-time track in the MSE program will participate in a studio project involving a real-time application. Over a sixteen month period, a team of MSE students work together on a studio project. They experience most of the software development life cycle, starting with the initial customer contact and the subsequent requirements elicitation, and continuing to the delivery of the software product to the customer. Lack of time usually prevents students from advancing to maintenance and product enhancement during the 16-month duration of the studio project.

In addition to other software components, the 1992-93 studio team of eight students is developing software to control the positioning of a robot's arm. Attached to the arm is an injection tool used to waterproof the tiles forming the thermal protection system of the NASA space shuttle. The arm is attached to a mobile base. A previous studio team developed software to control the movement of the mobile base. Safety is a critical factor in the development of this robot.

The 1993-94 studio team will be working on the APEX (A Planetary EXplorer) project involving the construction of a robot for lunar exploration studies. This project should provide interesting problems to be solved, such as the opportunity to develop an embedded expert system for a real-time application.

If a real-time project is not available for the studio work, then a student following a real-time track could pursue an independent study with a researcher from the SEI or CMU whose work relates to real-time computing (see Appendix A). As was mentioned earlier, the real-time domain involves the monitoring and control of systems used in applications areas such as the following: aerospace, basic machine control, communications, defense, entertainment, nuclear-reactor control, robotics, and transportation.

7 Conclusions and Guidelines for Administrating Specialization Tracks

Building a specialization track is a goal-oriented process in which the builder thinks critically about knowledge and skills required for the development of domain-specific software. The process of identifying the requisite knowledge bases and of mapping these to educational objectives may seem tedious.

The developer should select a level of detail which corresponds to the time available for building the track, as well as to the degree of control that the developer wants over the track. The developer may choose to delineate broad knowledge bases and let course designers specify the detailed educational objectives. The TAP-D model serves most effectively as a guideline rather than a prescription for creating tracks.

The model can be applied to the development of generic tracks to be pursued by more than one graduate student or to the development of individualized tracks. Faculty members, possibly with the help of application domain experts, would most likely specify the generic tracks. Faculty should encourage those students following individualized tracks to talk to experts in their chosen application domains; students should do this before defining the appropriate knowledge bases and skills for their individualized tracks.

Using the TAP-D model as a guideline, students could formulate their own educational objectives and map these to existing courses or to proposals for independent study. Ideally, a mentor knowledgeable in a student's chosen application area should help the student to identify the knowledge bases and skills. The mentor should approve the educational objectives and selected courses.

Because the full-time MSE program is an intensive 16-month course of study, students have limited time to develop individualized specialization tracks once classes begin. One way to solve this problem is to start the process of identifying and specifying an individualized specialization track during the summer preceding the start of the program. Here are some guidelines for administrating specialization tracks.

Guidelines for Administrating Specialization Tracks

1. *Admitted students should receive a description of the specialization track option within the MSE program soon after they accept admission.*

   This report could be sent accompanied by a letter explaining tracks which have already been defined as well as the opportunity for students to develop their own tracks. This mailing should include a reply card for students to indicate their interest in pursuing a specialization track. Students who prefer to develop their own tracks should indicate the application areas in which they are interested.

2. *Students opting to develop their own tracks should research their chosen application domains during the summer.*

   Advise these students to talk with domain experts in their chosen application areas. Encourage them to determine the domain-specific knowledge and skills that they need to develop software for these applications.

   The domain experts that they contact may be industrial sponsors, industrial mentors, professional contacts, or academic faculty. Provide the students with the names of CMU faculty or SEI staff members who might help them devise their tracks. The benefit of using CMU faculty or SEI staff is that the students can more easily obtain advice from these people upon their arrival for the MSE program orientation.

   Students opting to develop their own tracks are required to submit rough drafts of their proposals at least one week prior to the orientation which precedes the fall term. This would allow the MSE advisors to have time to read these proposals before the students arrive.

3. *During orientation, each student pursuing a specialization track should discuss the purpose and content of their chosen tracks with advisors.*

   Those students pursuing predefined tracks can discuss course offerings and scheduling with their advisors. Students who have drafted their own tracks can discuss the domain-specific knowledge and skills which they have identified with their advisors. The advisors should review the rough drafts and suggest appropriate changes. The advisor can then help the student map the knowledge bases and skills to educational objectives or directly to courses. In the first case, the educational objectives should then be mapped to courses.

4. *The process of developing and pursuing a specialization track should be dynamic and flexible.*

   Students should continually review their educational experiences. They may propose changes to generic or individualized tracks. Students developing their own tracks may choose to incrementally provide more detail to their proposals for a track as they learn more about their chosen application domains. Advisors should review and approve student proposals.

5. *After track completion, students who develop their own tracks should submit reports outlining their tracks and briefly reviewing their experiences.*

   Students could receive independent study credit for writing and submitting acceptable reports. These reports should be written in such a way that future students may use them to follow similar tracks.

6. *The MSE program administrators should provide students with documentation that certifies their completion of specialization tracks. For example, students who complete specialization tracks could receive certificates of completion during the MSE graduation ceremonies.*

   Students would probably appreciate and find useful for the job search proof that they have completed a specialization track.

Student development of individualized specialization tracks provides three benefits: (1) it alleviates the faculty time and expertise needed to develop appropriate specialization tracks for individual students, (2) it increases the number of predefined tracks for other students, and (3) it encourages students to take responsibility for developing their own educational goals to match their chosen application areas. The author expects that students who identify an interest in an application area and formulate educational goals that enable them to develop requisite knowledge and skills will not only be better prepared for a software engineering position in the application area but will also be more successful in managing their own careers.

As a recent study by the Conference Board suggests, employee career management and financial planning evidences the shift away from the paternalistic and protective, employer-provided practices of the past. Employees must now manage their own careers and plan for their future financial security. Cost considerations and government regulations account for corporate efforts to involve employees in the planning, saving, and investing processes that will determine their financial viability in later life. The shift of responsibility for career management from the employer to the employee is a result of factors such as corporate downsizing and restructuring, uncertain business and job market conditions, and limited opportunities for career advancement.[4]

The ability to seek help from domain experts, to identify requisite knowledge and skills, and to delineate meaningful educational goals as well as the discipline to complete a chosen course of action are essential to the management of one's career. Many companies today like to hire software professionals with broad software engineering expertise as well as domain-specific knowledge and skills. Software professionals must continually and actively seek information about career opportunities and must manage their own careers. Defining and/or following a specialization track should help MSE students prepare for self-managed, software engineering careers with enhanced opportunities to develop software for chosen application areas.

As Guisbond wrote in 1990, specialization is a dominant trend that requires people to take an informed and directed approach to managing their careers. Those people with the best job opportunities are those who prepare for anticipated technology trends and who tailor their job searches accordingly. [Guisbond90]

---

4. The Conference Board is a nonprofit, bipartisan organization founded in 1916 to improve business enterprise systems and to enhance the contribution of business to society. Researchers in this organization conducted a study of trends in career management and financial planning during the fall of 1990. The study included 28 interviews with 14 corporations and one large accounting firm. The corporations included older companies in major industries as well as younger firms in high-technology businesses. The interviews were conducted with people within these corporations who were in charge of human resource planning, benefits, training, and development. The study also included discussions with financial planning and career development consultants as well as executives. In addition, the researchers reviewed recent literature in the business press which discussed these topics. [Dennis91]

Acknowledgments

The author would like to thank Mary Shaw for her idea to incorporate specialization tracks into the MSE program and for her support in the design and implementation of these tracks. The author would also like to thank Jorge Díaz-Herrera, Ragunathan Rajkumar, Daniel Roy, and Lui Sha for their ongoing development of core courses for the real-time track.

The author also thanks Cliff Mercer and Jeff Otten for their help in preparing and presenting the *Introduction to Real-Time Software* course as well as Roger Dannenberg, Nancy Mead, and Mary Shaw for their advice on course content and organization. The author thanks James Tomayko for his ongoing efforts to coordinate the MSE Studio course with specialization tracks, in particular the real-time track. The author thanks those people who reviewed the document. Lastly, the author thanks Phyllis Lewis and Ellen Saxon for their administrative and motivational support.

8 References

[Allman87] Allman, Dena. “Riding the Waves of Technology.” *Computerworld* 21, 19 (May 11, 1987): 100.

[Atkinson91] Atkinson, Colin. *Object-Oriented Reuse, Concurrency and Distribution: An Ada-Based Approach.* New York, N.Y.: ACM Press, 1991.

[Burns90] Burns, Alan; & Wellings, Andy. *Real-Time Systems and Their Programming Languages.* International Computer Science Series. Wokingham, England: Addison-Wesley Publ. Co., 1990.

[CIT92] Carnegie Institute of Technology. *Electrical and Computer Engineering Research and Graduate Studies*. Pittsburgh, Pa.: Carnegie Mellon University, 1992.

[Dennis91] Dennis, Helen; & Axel, Helen. *Encouraging Employee Self-Management in Financial and Career Planning* (No. 976). New York, N.Y.: The Conference Board, 1991.

[Elliot92] Elliot, William F., ed. *1992-94 Undergraduate Catalog*. Pittsburgh, Pa.: Carnegie Mellon University, 1992.

[Guisbond90] Guisbond, Lisa. “Boston Blue - But Hiring.” *Computerworld* 24, 38 (September 17, 1990 - Northeast U.S. ed.): 130.

[Halang90] Halang, Wolfgang A. “A Curriculum for Real-Time Computer and Control Systems Engineering.” *IEEE Transactions on Education* 33, 2 (May 1990): 171-178.

[John92] John, Bonnie E.; Miller, Philip L.; Myers, Brad A.; Neuwirth, Christine M.; & Shafer, Steven A. *Human-Computer Interaction in the School of Computer Science* (CMU-CS-92-193). Pittsburgh, Pa.: Carnegie Mellon University, 1992.

[Laplante93] Laplante, Phillip A. *Real-Time Systems Design and Analysis: An Engineer’s Handbook.* New York, N.Y.: IEEE Press, 1993.

[Mok91] Mok, Aloysius K. Ch. 1, “Towards Mechanization of Real-Time System Design,” 1-37. *Foundations of Real-Time Computing: Formal Specifications and Methods*, Norwell, Mass.: Kluwer Academic Publishers, 1991.

[Moore88] Moore, Freeman L.; & Purvis, Phillip R. “Meeting the Training Needs of Practicing Software Engineers at Texas Instruments,” pp. 32-44. *Lecture Notes in Computer Science: Proceedings from the Software Engineering Education SEI Conference.* New York, N.Y.: Springer-Verlag, Apr. 28-29, 1988.

[MSE93] Master of Software Engineering Program. *MSE Program Description for the 1993-94 Academic Year*. Pittsburgh, Pa.: Carnegie Mellon University, 1993.

[SCS92] School of Computer Science. *Graduate Studies in Computer Science.* Pittsburgh, Pa.: Carnegie Mellon University, 1992.

[Shaw92] Shaw, Mary. *Informatics for a New Century: Computing Education for the 1990’s and Beyond* (CMU/SEI-90-TR-15). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University.

[van Tilborg91] van Tilborg, André M.; & Koob, Gary M., eds. *Foundations of Real-Time Computing: Formal Specifications and Methods.* Norwell, Mass.: Kluwer Academic Publishers, 1991.

Appendix A

The following table contains information about CMU researchers and administrators in the School of Computer Science and at the SEI who are interested in real-time computing and who are involved in work related to real-time projects. The information in this table is meant to help students locate people whose interests and activities match their interests and educational objectives.

The table may not provide a comprehensive list of all faculty and students interested in real-time computing. As students become familiar with the CMU community, they will undoubtedly find other people whose work relates to real-time computing. The interest column contains only a highlight of a person's involvement with real-time systems.

Abbreviations Used in the Table

ART - Advanced Real-Time

ECE - Electrical and Computer Engineering

EPP - Engineering and Public Policy

SCS - School of Computer Science

SEI - Software Engineering Institute

TCA - Task Control Architecture

Table A-1: List of Researchers and Administrators

Researcher/Administrator	Interests in Real-Time Systems	E-Mail Address
Mario R. Barbacci Senior Research Computer Scientist, SCS, SEI	Director of the Real-Time Distributed Systems Program	mrb@sei.cmu.edu
Clarke, Edmund Senior Research Computer Scientist, SCS	Verification of Hardware Using Temporal Logic	emc@cs.cmu.edu
Dannenberg, Roger Senior Research Computer Scientist, SCS	Computer Music, Advisor for the Development of the Introduction to Real-Time Software Course	rbd@cs.cmu.edu
Díaz-Herrera, Jorge Luis Senior Member of the Technical Staff, SEI	Real-Time Systems Software Design	jldh@sei.cmu.edu

Researcher/Administrator	Interests in Real-Time Systems	E-Mail Address
Firth, Robert Senior Computer Scientist, SEI	Real-Time Systems Design and Implementation	firth@sei.cmu.edu
Garlan, David Assistant Professor, SCS	Software Architectures for Real-Time Systems, Software Analysis of TCA	garlan@cs.cmu.edu
Goodenough, John Senior Member of the Technical Staff, SEI	Rate Monotonic Analysis for Real-Time Systems	jbg@sei.cmu.edu
Hoover, Carol L. Lecturer, MSE Program	MSE Real-Time Track Development, Real-Time Software Development	clh@cs.cmu.edu
Khosla, Pradeep K. Associate Professor, ECE	Head of the CHIMERA II Project, Real-Time Kernel for Control Applications	pkk@ece.cmu.edu
Lewis, Phyllis Program Administrator for the MSE Program	Assistance in the Administration of Specialization Tracks	plewis@cs.cmu.edu
Mead, Nancy Manager SEI Products Program	Advisor for the Development of the Introduction to Real-Time Software Course	nrm@sei.cmu.edu
Mercer, Cliff Ph.D. Student, SCS	ART Project, Real-Time Mach	mercer@cs.cmu.edu
Peha, Jon M. Assistant Professor, ECE & EPP	Real-Time Scheduling, Load Balancing in Distributed Real-Time Systems	peha@ece.cmu.edu
Rajkumar, Ragunathan (Raj) Member of the Technical Staff, SEI	Rate Monotonic Scheduling Theory, Zero-Defect Application Kernels	rr@sei.cmu.edu
Rissman, Michael Member of the Technical Staff, SEI	Real-Time Simulators	mmr@sei.cmu.edu
Roy, Daniel Senior Member of the Technical Staff, SEI	Real-Time Software Performance, Embedded System Testbed	dmr@sei.cmu.edu
Savage, Stefan Research Programmer, SCS	Research in Operating Systems Performance, ART Project	savage@cs.cmu.edu

Researcher/Administrator	Interests in Real-Time Systems	E-Mail Address
Sha, Lui Senior Member of the Technical Staff, SEI	Rate-Monotonic Scheduling Theory, Zero-Defect Application Kernels	lrs@sei.cmu.edu
Shafer, Steven A. Associate Professor, SCS Associate Director of the Robotics Ph.D. program	Robot Architecture Design, Calibrated Imaging Lab	sas@cs.cmu.edu
Shaw, Mary Professor, SCS Associate Dean of Professional Education	Software Architectures for Real-Time Systems, Advisor for Real-Time Track Development	mary.shaw@cs.cmu.edu
Siewiorek, Daniel P. Professor, SCS & ECE	Automatic Design of High-Performance, Fault-Tolerant Real-Time Systems	dps@ece.cmu.edu
Simmons, Reid G. Computer Research Scientist, SCS	Robot Planning and Control, TCA	reids@cs.cmu.edu
Stonick, Virginia Assistant Professor, ECE	Real-Time, High-Quality Video Processing; Filters for Music Synthesizing	ginny@ece.cmu.edu
Strosnider, Jay K. Assistant Professor, ECE	Real-Time Operating Systems, Networks, and Fault Recovery	jks@ece.cmu.edu
Tokuda, Hideyuki Senior Research Computer Scientist, SCS	Head of the ART Computing Project	hxt@cs.cmu.edu
Tomayko, James E. Senior Member of the Technical Staff, SEI	Coordinator of the MSE Software Development Studio, Real-Time Embedded Projects	jet@sei.cmu.edu