Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 633 – 638

ISBN 978-83-60810-14-9
ISSN 1896-7094

An RSIC-SE2004 Curriculum Framework

Thomas B. Hilburn
Embry-Riddle Aeronautical University, U.S.A., Email: hilburn@erau.edu

Andrew Kornecki
Embry-Riddle Aeronautical University, U.S.A., Email: kornecka@erau.edu

Jean-Marc Thiriet
Grenoble Université, France, Email: jean-marc.thiriet@ujf-grenoble.fr

Wojciech Grega
AGH University of Science & Technology, Poland, Email: wgr@agh.edu.pl

Miroslav Sveda
Brno University of Technology, Czech Republic, Email: sveda@fit.vutbr.cz

Abstract — **This paper addresses the problem of educating software engineering professionals who can efficiently and effectively develop real-time software intensive control systems in the global community. A framework for developing curricula that support such education is presented. The curriculum framework is based on the work of two education projects: the ILERT (International Learning Environment for Real-Time Software Intensive Control System) project, and the software engineering efforts of the ACM/IEEE-CS Joint Task Force on Computing Curricula. The authors describe a curriculum framework that integrates principles, content, and organization from the two projects, and which satisfies the intent and requirement of both projects.**

I. Introduction

THE development of software for real-time, embedded, safety-critical systems (such as for controlling and supporting systems in aviation and space, medicine and health, and atomic energy) is a complex and challenging problem. The health, safety, welfare, and productivity of the public and the world economy increasingly depend on software-intensive systems – the dependability of these systems is paramount. Unfortunately, software developers do not consistently provide safe, secure, and reliable systems; such systems are often of poor quality, and cost and schedule overruns are common. There has been significant improvement in software development methods and practices in the last twenty years. The application of software engineering best practices to such development has proven the value of such practices [1]-[3]. In order to enhance such capabilities for software engineering professionals there have been numerous calls to improve software engineering education, especially in the area of real-time embedded systems [4]-[6].

The last twenty years have also witnessed significant advancements in the state of computer science education (and in allied fields such as computer engineering, information systems, and software engineering). The Association for Computing Machinery (ACM), the IEEE Computer Society (IEEE-CS), and CSAB (which determines criteria for accreditation of programs in computer science and software engineering) have provided encouragement, support, and guidance in developing quality curricula that are viable and dynamic. Degree programs have moved from language- and coding-centered curricula to those that emphasize theory, abstraction, and design. To address the problems in software development the ACM/IEEE-CS Joint Task Force on Computing Curricula has produced guidelines for curricula in computer engineering, computer science, information systems, information technology, and software engineering. The software engineering guidelines, SE2004 (*Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*) [7], provide information and guidance on various curriculum issues: objectives, content, organization, courses, and pedagogy.

More recently the ILERT (International Learning Environment for Real-Time Software Intensive Control System) project [8] has been involved in the creation of an international curriculum framework centered on RSIC (Real-Time Software-Intensive Control) systems. The ILERT study explores a mechanism for involving students from multilingual, geographically separated institutions in a coordinated educational experience. The ultimate objective is the creation of a RSIC curriculum model, which can be used by engineering schools both in the USA and the EU.

The purpose of this paper is to discuss how the work of the ILERT project might be used to develop an SE2004 software engineering curriculum which satisfies the RSIC framework. In the following sections we provide additional detail about ILERT, the SE2004 guide, and the RSIC curriculum framework. Then, this material is integrated to outline a curriculum which satisfies both the SE2004 and the RSIC frameworks.

II. The Ilert Project

The analysis, design, implementation, administration, and assessment of international curricula will become increasingly important in the global community of the 21st century. In support of this critical issue, the European Commission and the US Department of Education have funded the ATLANTIS initiative to promote collaboration in higher education between European and American universities. One American (Embry-Riddle Aeronautical University, Daytona Beach, FL) and three European Universities (AGH University of Science and Technology, Krakow, Poland; Brno University of Technology, Czech Republic; and The University of Grenoble, France) are currently working on the framework of a new common curriculum in real time-software systems. This two-year project "Toward International Learning Environment for Real-Time Software Intensive Control Systems" (EC grant: 2006-4563/006 001, US grant: P116J060005, http://www.ilert.agh.edu.pl) was launched in January 2007. Project work is concerned with program objectives and outcomes, curriculum content and pedagogy, program administration (academic credit, course schedules, exchange of students and staff, etc.), and program assessment and accreditation. Thus far, the project has produced the following deliverables:

- An analysis of industry requirements for graduates in the RSIC domain
- An identification of the RSIC learning objectives and student outcomes
- An analysis of European Credit Transfer System (ECTS) and the mechanism of credit transfer at U.S. Colleges and Universities
- An identification of activities and data for program assessment and evaluation, and those issues and elements required to consider program accreditation
- A description of an international, interdisciplinary RSIC curriculum framework
- A preliminary design for a selected unit supporting the proposed RSIC curriculum.

For the analysis of industry requirements, a survey of USA and European industry engaged in real-time software-intensive control systems was conducted [9]. The survey consisted of two parts: General Skills and Attitudes (10 items), and Technical Knowledge Areas (15 items). For example, one of the Technical Items was "Knowledge of software design and development concepts, methods and tools". Respondents were ask to select "Essential, Important, Unrelated, or Unimportant", with a possibility to provide comment.

Survey results from 43 companies were analyzed and summarized. In the General Skills part, the highest rated skills as follows:

- Work as a part of a multidisciplinary team
- Analyze, understand and define the problem

For the Technical Knowledge part, the following were rated the highest:

- Software design and development concepts, methods and tools
- System specification and design methods

Project work continues on experimental concurrent delivery of the designed RSIC unit at the four partner sites. Finally, the project will provide reflection on a process and methodology for creation of multidisciplinary, transatlantic engineering programs, including guidelines for extension of the approach to other engineering disciplines.

III. The RSIC Curriculum Framework

This section provides information on the organization and content of the RSIC curriculum framework. The framework is a high-level curriculum specification that is detailed enough to guide the development of a RSIC program, which supports the RSIC objectives and outcomes, and yet is flexible enough to account for specializations, constraints, and requirements of various programs, institutions, and regions.

Table I.
RSIC Components

Software Engineering (SoftEng-)
Software engineering concepts and practices, software lifecycle models, project management, software processes, software construction methods and practices, software modeling and formal representation; software requirements; software architectural and module design; testing and quality assurance; software maintenance; and notations and tools.
Digital Systems (DigSys-)
Digital system concepts/operation, design of combinatorial/sequential circuits, concepts and operation of microcontrollers/microprocessors, assembly language, rudimentary interfacing and exception handling, large scale integration devices and tools, interfacing, advanced memory management, fault tolerant hardware.
Computer Control (CompCtrl)
Concepts of feedback control, time and frequency domains, continuous and discrete models of dynamical systems, state analysis, stability, controllability and observability, controller design, implementing control algorithms in real-time, integrated control design and implementation use of analysis and design tools.
Real-Time Systems (RTSys)
Timing and dependability properties of software intensive systems, RTOS concepts and applications, concurrency, synchronization and communication, scheduling, reliability and safety, etc.
Networking (Network)
Data communication, network topology, analysis and design, information security, algorithms, encryption, bus architectures, wireless, etc. distributed control and monitoring
System Engineering (SysEng)
System engineering concepts, principles, and practices; system engineering processes (technical and management); system requirements, system design, system integration, and system testing; special emphasis on the development of a RSIC system and the integration of RSIC system elements.

The basic organizational unit for the framework is a RSIC "component". A RSIC component is a curriculum unit which covers theory, knowledge and practice which supports the RSIC curriculum objective and outcomes. Table I describes the RSIC components in six identified RSIC areas: Software Engineering, Digital Systems, Computer Control, Real-Time Systems, Networking, and Systems Engineering.

The RSIC Curriculum Framework does not specify the way in which component topics might be formed into modules or courses. Component topics might be focused in one or two courses, or spread among several courses, along with other non-RSIC topics. Depending on the course rigor and the required prerequisite knowledge, the material can be at either a basic or an advanced level. The curriculum framework includes more detailed specifications of each component: prerequisite knowledge, component learning objectives, information about required facilities and equipment, and guidelines and suggestions for course design and delivery. Table II is an example of one such component specification. The full details of each competent specification will be included in the final ILLERT project report.

The RSIC curriculum framework also makes recommendations about non-RSIC courses or units that should be part of a RSIC program, as prerequisite courses or to supplement the components as part of a full degree program. The recommendations call for courses in the following areas:

- Mathematics (Differential and Integral Calculus, Differential Equations, Discrete Mathematics, Statistics, Linear Algebra)
- Physics (mechanics, E&M, thermo, fluids)
- Electrical Engineering (circuit analysis, basic electronics)

TABLE II.
SOFTWARE ENGINEERING

Description	Software engineering concepts and practices, software lifecycle models, project management, software processes, software construction methods and practices.
Prerequisite Knowledge	Ability to design, implement and test small programs (100 lines of code), written in a commonly used high-level programming language.
Learning Outcomes	Upon completion of this component, students should be able to • Describe the major problems in the development of a large, complex software system. • Describe and discuss issues, principles, methods and technology associated with software engineering theory and practices (e.g., planning, requirements analysis, design, coding, testing, quality assurance, risk assessment, and configuration management). • Working as part of a team, use a defined software development process to develop a high-quality modest sized software product (1000 lines of code). • Describe issues, principles, methods and technology associated with the use of formal modeling in software engineering. • Describe, discuss, and apply the commonly accepted principles of software quality assurance (reviews, inspections and testing). • Apply requirements engineering principles of elicitation, analysis, and modeling to the development of a requirements specification. • Describe and analyze different software architectures views and styles. • Describe and discuss the structured and object-oriented design methodologies. • Describe and discuss the principles, methods and practices of software evolution. • Show capability with various software engineering tools used for formal software modeling, requirements engineering and software design.
Facilities and Equipment	No special equipment or laboratory is required for this component. Students will need access to a computer system equipped with a program development environment (such as Eclipse). There may be a need for process, management and formal modeling tools; but, typically word processors, spreadsheets and simple scheduling tools should be sufficient.
Guidelines and Suggestions	The emphasis in this component is for students to know and understand how large complex systems should be developed, not, at this point, to be able to develop such systems. For instance, they should understand and be able to describe and discuss the activities and practices that take place in each software development life-cycle phase; they should understand the importance of requirements and the problems that ensue if requirements are not properly elicited and specified; they should understand the various elements of project management; they should come to realize that testing is not the only way to ensure quality; they should come to see software development as an engineering discipline; and they should understand the importance of discipline and process to the development of software. Case studies are particularly helpful in teaching software engineering principles – when students study examples of actual requirements, design, test and planning documents, they better understanding the nature of software engineering and what it takes to develop high-quality software products. There are many introductory software engineering textbooks that contain such examples (Pressman, Somerville, Lethbridge, Pfleeger) This component should include a software team project. It is extremely important that the software product developed be modest in scope and functionality. The purpose of the project is for students to learn about working as part of the team, to experience the software-life cycle, to see how to assure quality, to use planning and process procedures, to document the team's work, and to appreciate the difficulty of developing a high-quality software product. The emphasis should be on teamwork, quality and process. This component is intended to provide more in-depth coverage of software engineering topics than the Basic Level component. However, the coverage still must be at a level that can be covered in one or two courses. It is not intended that there would be an in-depth study of each of the separate areas of formal modeling, requirements, architecture and design, and quality and testing. Such in-depth coverage would require three or four course of study. A development project could involve the creation of a requirements and architecture specification for a software system with more complexity than the Basic level team project. This component would benefit from the use of case study documents (requirements, architecture, module design, code, test plans, project plans, etc.) for a reasonably complex system. Analysis and maintenance problems focused on the case study documents would be helpful in achieving the component learning outcomes.

TABLE III.
SEEK KNOWLEDGE AREAS

Knowledge Area	Hours
Computing Essentials	172
Mathematical & Engineering Fundamentals	89
Professional Practice	35
Software Modeling & Analysis	53
Software Design	45
Software V & V	42
Software Evolution	10
Software Process	13
Software Quality	16
Software Management	19
total hours	494

- Engineering Economics
- Introduction to Computer Science with Programming

IV. THE SE 2004 CURRICULUM FRAMEWORK

The software engineering guidelines document, SE2004 [7], provides a comprehensive and detailed set of material to support the development of an undergraduate curriculum in software engineering. Specifically it includes chapters on the following:

- The Software Engineering Discipline
- Guiding Principles
- Overview of Software Engineering Education Knowledge (SEEK)
- Guidelines for SE Curriculum Design and Delivery
- Courses and Course Sequences
- Adaptation to Alternative Environments
- Program Implementation and Assessment

The SEEK describes the body of knowledge that is appropriate for an undergraduate program in software engineering. It designates “core” material which SE2004 recommends is necessary for anyone to obtain an undergraduate degree in the field. It designates Bloom’s levels for the knowledge units [10] and makes a time allocation of “contact” hours. Table III lists the knowledge areas that make up the SEEK and indicates the minimum total time recommended for each area. The SE2004 contains a more detailed description of the content and organization of each area.

In addition to the core materials, undergraduates are encouraged to specialize in some area related to software engineering application. The following specialties are presented in the SE 2004 volume:

a) Network-centric systems
b) Information systems and data processing
c) Financial and e-commerce systems
d) Scientific systems
e) Telecommunications systems
f) Fault tolerant and survivable systems
g) Highly secure systems
h) Safety critical systems
i) Embedded and real-time systems
j) Biomedical systems
k) Avionics and vehicular systems
l) Industrial process control systems

TABLE IV.
SE2004 CORE COURSES

Number	Title
SE101	Software Engineering and Computing I
SE102	Software Engineering and Computing II
SE103	Software Engineering and Computing III
SE211	Software Construction
SE212	Software Engineering Approach to Human Computer Interaction
SE311	Software Design and Architecture
SE321	Software Quality Assurance and Testing
SE322	Software Requirements Analysis
SE323	Software Project Management
SE400	Software Engineering Capstone Project

Of these, several certainly correlate strongly with the RCIS components. This provides the motivation and rationale for proposing a framework for an SE 2004 RSIC curriculum.SE2004 also includes a set of courses and their descriptions which, when grouped together, provide for a set of courses which cover the SEEK core knowledge. Table IV describes one such set of courses.

Courses SE101, SE102 and SE103 provide an overview of software engineering with topics typically included in introductory courses in computer science, low level design, and programming. Each course, except for SE400, was envisioned to be offered over approximately 14 weeks, with 3 contact hours per week. SE400 covers a full academic year of work. Other schedules and timelines are possible. SE2004 provides detailed descriptions of each course.

In addition, SE2004 contains several models (or patterns) of how the courses could be arranged into a full three or four year curriculum. The models include three-year and four-year patterns, with versions for North America, Europe, Japan, Australia, and Israel, Table V depicts a four year “North American” curriculum pattern. In addition to the SE core courses in Table V (shaded gray), the SE2004 also contains course descriptions for supporting courses such as Discrete Math I and II, Data Structures and Algorithms, Database Systems, Computer Architecture, Operating Systems and Networking, etc.

V. A COMBINED SE 2004-RSIC CURRICULUM

In this section we present a curriculum that incorporates the RSIC requirements within the SE2004 requirements. Table VI outlines a four year RSIC-SE2004 curriculum. It represents a modification of Table V: all of the software engineering core courses were retained; and technical electives and science electives items were replaced with courses which

TABLE V
SE2004 NORTH AMERICAN CURRICULUM PATTERN

Year 1		Year 2		Year 3		Year 4	
Sem 1A	**Sem 1B**	**Sem 2A**	**Sem 2B**	**Sem 3A**	**Sem 3B**	**Sem 4A**	**Sem 4B**
SE101	SE102	SE 200	SE 211	SE 212	SE 311	SE400	SE400
Dis Math I	Dis Math II	Data Str & Alg	Database	SE 321	SE 322	SE 323	Tech Elect
Calc 1	Calc 2	Physics 1	Physics 2	Comp Arch	Prof SE Practice	OS & Network	Tech Elect
Gen Ed	Gen Ed	Gen Ed	Statistics	Sci Elect	Tech Elect	Eng Econ	Open Elect
Gen Ed	Gen Ed	Gen Ed	Sci Elect	Sci Elect	Gen Ed	Gen Ed	Open Elect

cover the RSIC curriculum components.

The courses in bold type in Table VI represent courses that directly address the requirements of the RSIC components: for example, **DigSys-1** and **DigSys-2** would cover the Digital Systems Component. Notice that **SoftEng** is designated in several courses. This was necessary in order to cover all the RSIC SE advanced topics. Due to the nature of this specific SE program, such topics would be covered in more depth than strictly required by the RSIC Framework. In addition, a number of course were added in order to support prerequisite requirements for RSIC courses: differential equations, electrical engineering and linear algebra.

The course "Prof **RSIC** Practice" is a replacement for the course NT291 (Professional Software Engineering Practice - knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner.). The RSIC course would contain much of the material from NT291; however, it would also furnish students with material and activities that support two of the non-technical RSIC curriculum outcomes:

- An ability to work effectively in an international environment
- An understanding of the impact of engineering solutions in a global and societal context

One other important item in Table VI is that **SysEng** was designated as part of the SE400. SE400 is a capstone project course and it is envisioned that the project will be a real-time software intensive control system. It relies on and will bring together the knowledge and practices learned by students in the other RSIC courses. Although students will have been introduced to some system concepts in other courses, SE400 is the ideal place to focus on system issues: requirements require system level decisions about allocation and specification; the system architecture will involve software and hardware subsystems and components; systems quality assurance measures will have to be instituted; and the procedures and activities' involved in system integration will be critical.

Although Table VI provides an outline of a RSIC-SE400 curriculum, much more analysis and detail is needed to support implementation of such a curriculum. However, it is the hope of the authors that this paper, along with the other work of the ILERT project, will motivate and support faculty who wish to create and implement educational programs which will improve the development of RSIC systems.

VI. CONCLUSION

Creation of RSIC systems engages a large variety of engineering disciplines. Due to worldwide implementation of such systems, a well prepared workforce of scientists and engineers is required. They must be able to work cooperatively in multi-disciplinary and international settings. The software intensive nature of RSIC systems require engineers who understand and can use the software engineering knowledge and practices required to build such systems. The authors feel that the RSIC-SE2004 the potential to have broad impact on the future of engineering education and on the efficient and effective development of RSIC systems.

In addition, the process and format of the RSIC-SE2004

TABLE VI
A RSIC-SE2004 CURRICULUM

Year 1		Year 2		Year 3		Year 4	
Sem 1A	**Sem 1B**	**Sem 2A**	**Sem 2B**	**Sem 3A**	**Sem 3B**	**Sem 4A**	**Sem 4B**
SE101	SE102	SE 200 **SoftEng-1**	SE 211 **SoftEng-2**	SE 212 **SoftEng-3**	SE 311 **SoftEng-5**	SE400 **SysEng-1**	SE400 **SysEng-2**
Dis Math I	Dis Math II	Data Str & Alg	**DigSys-1**	SE 321 **SoftEng-4**	SE 322 **SoftEng-6**	SE 323 **SoftEng-7**	Tech Elect
Calc 1	Calc 2	Physics 1	Physics 2	**DigSys-2**	**RTSys**	Prof **RSIC** Practice	Tech Elect
Gen Ed	Gen Ed	Gen Ed	Statistics	**CompCtrl**	**Network**	Eng Econ	Open Elect
Gen Ed	Gen Ed	Diff Eqns	Elec Eng	OS & Network	Lin Alg	Gen Ed	Gen Ed

curriculum framework could be used as a model for the development of other “integrated” curricula (e.g., a RSIC curriculum for computer engineering, or a RSIC curriculum for control engineering).

Acknowledgment

The authors would like to thank the European Commission, the US Department of Education and the ATLANTIS program for their support of ILERT project: "Toward International Learning Environment for Real-Time Software Intensive Control Systems" (EC grant: 2006-4563/006 001, US grant: P116J060005, http://www.ilert.agh.edu.pl).

References

[1] M. Cusumano, A. MacCormack, C. Kemerer, and B. Crandall, “Software Development Worldwide: The State of the Practice”, *IEEE Software*, pp.28-34, vol. 20, no. 6, Nov./Dec. 2003.

[2] N. Davis, N. and J. Mullaney, *The Team Software Process (TSP) in Practice: A Summary of Recent Results*, CMU/SEI-2003-TR-014, Software Engineering Institute, Carnegie Mellon University, Sep. 2003.

[3] C. Jones, “Variations in Software Development Practices”, *IEEE Software*, pp.22-37, vol. 20, no. 6, Nov./Dec. 2003.

[4] W. Humphrey and T. Hilburn, “The Impending Changes in Software Education”, *IEEE Software*, Vol. 19 , No. 5, pp. 22-24, Sep. / Oct., 22 – 24, 2002

[5] J. Knight, N. Leveson, “Software and Higher Education”, Inside Risks Column, *Communications of the ACM*, p. 160, vol. 49, no. 1, Jan. 2006.

[6] L. Long, “The Critical Need for Software Engineering Education”, *CrossTalk: The Journal of Defense Software Engineering*, pp. 6-10, Jan. 2006. (http://www.stsc.hill.af.mil/crosstalk/2008/01/0801-Long.pdf)

[7] ACM/IEEE-CS Joint Task Force on Computing Curricula, *Software Engineering 2004,Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, Aug. 2004. (http://www.acm.org/education/curricula.html)

[8] W. Grega, A. Kornecki, M. Sveda, and J. Thiriet, “Developing Interdisciplinary and Multinational Software Engineering Curriculum”, *Proceedings of the ICEE’07*, Coimbra, Portugal, Sep. 3-7, 2007.

[9] A. Pilat, A. Kornecki, J. Thiriet, W. Grega, and M. Sveda, “Industry Feedback on Skills and Knowledge in Real-Time Software Engineering”, Proceedings of 19th EAEEIE Annual Conference, Tallinn, Estonia, Jun29 - Jul 2, 2008.

[10] B. S. Bloom, Editor, *Taxonomy of Educational Objectives: The Classification of Educational Goals: Handbook I, Cognitive Domain*, Longmans, 1956.