AC 2008-1359: A COMMON US-EUROPE CURRICULUM: AN APPROACH FOR REAL-TIME SOFTWARE INTENSIVE SYSTEMS

Andrew Kornecki, ERAU

MSEE, PhD, Professor; engaged in a variety of research projects sponsored by the FAA, NSF, Florida State, and the industry (~$700K as the PI, ~$2.5M as co-PI); author and co-author of over 80 refereed papers in journals and conference proceedings; construction of real-time and safety critical software, embedded systems, computer simulation and aviation software, control and computer engineering education; teaching in undergraduate and graduate engineering programs on three continents; established ERAU Real-Time Software Laboratory; consulting and providing training for industry and government in real-time safety critical software; served on committee of the National Academies of Science and Engineering; currently serving on the US/EU international RTCA SC205/WR71 committee on development of aviation systems software guidance material (http://faculty.erau.edu/korn/)

Thomas Hilburn, Embry-Riddle Aeronautical University-Prescott

Ph.D., Professor, Department; Visiting Scientist at the Software Engineering Institute, Carnegie Mellon University; worked on software engineering research projects with the FAA, the Mitre Corporation and the NSF; current interests include software processes, object-oriented design, formal specification techniques, and curriculum development, and he has authored/co-authored over 40 referred papers in these areas; an IEEE Certified Software Developer, SEI-Certified PSP Developer, and the editor for the ACM/IEEE-CS Computing Curriculum-SE project (http://faculty.erau.edu/hilburn/)

Wojciech Grega, AGH

MSEE, PhD, DSc: control, optimization, modeling, industrial computers science; author and co-author of more than 100 papers and books. Coordinator or main researcher in 12 national and international projects; coordinator of Tempus Project S-JEP 11317-96, (1997-1999); Vice Dean of the Faculty (1994-96); Head of the Control Laboratory, Head of the Faculty Commission for Education; elected member of the EAEEIE (European Association for Education in Electrical and Information Engineering), EU Tempus Academic Expert (expert list EAC 73/02) (http://aq.ia.agh.edu.pl/Aquarium/EN/staff/wgr.html)

Jean-Marc Thririet, LAG

Professor in Université Joseph Fourier – Laboratoire d'Automatique de Grenoble (LAG UMR 5528 CNRS-INPG-UJF) since September 2005; previously an Associate Professor in Université Henri Poincaré Nancy 1 – Centre de Recherche en Automatique de Nancy (CRAN UMR 7039 CNRS-UHP-INPL); networked control systems, distributed systems, dependability; teaching activities deal mainly with communication networks and automatic control; the coordinator of the EIE-Surveyor Thematic Network (www.eie-surveyor.org) from the European Commission (2005-2008); member of the THEIERE thematic network (2000-2005) and partner of the Tempus Project S-JEP 11317-96; elected member of the Council of the EAEEIE (treasurer from 1999 to 2005).
(http://www.lag.ensieg.inpg.fr/fr/pagePersoITA.php?id=Jean-Marc.Thiriet@inpg.fr&type=p)

Miroslav Sveda, BUT

PhD: formal specifications, real-time and embedded systems, computer networks and communication protocols; author and co-author of more than 100 papers and book chapters; coordinator or principal investigator in 15 national and international projects including COST 247 (Verification and Validation Methods for Formal Descriptions), EC Copernicus COP94-774 SEIHPC, and InterpRISe - Interregional Co-operation for Promoting Regional Innovation Strategies in Europe; member of the IEEE Technical Committee on Engineering of

Computer-Based Systems, IFIP WG10.1, and Board of Editors for Journal of Universal Computer Science; Chair of Curriculum Committee for Ph.D. studies in Information Technology, Curriculum Committee for MSc. and BSc. studies in Information Technology and member of Scientific Board at the Faculty of Information Technology, Brno University of Technology; and Curriculum Committee for Ph.D. studies in Informatics at the Faculty of Informatics, Masaryk University in Brno (http://www.fit.vutbr.cz/~sveda/.en)

A Common US-Europe Curriculum: An Approach for Real-Time Software Intensive Systems

Abstract

With the increasing importance and demand for efficient development of high quality Real-Time Software-Intensive Control systems (RSIC), the education of modern engineers it is critical. RSIC systems need to meet stringent safety and reliability requirements and often are developed by companies operating across national boundaries. This paper describes an approach and preliminary results of research leading to establishment of a framework for creation of multinational, engineering programs, which will produce graduates capable of working efficiently in multidisciplinary teams engaged in international collaboration on industrial RSIC projects. The emphasis is on projects which require conformance to specific national and international standards mandated by regulatory authorities. A key element of the framework is the identification of appropriate educational objectives and outcomes for the program based on industry surveys and the analysis of accreditation criteria. The proposed RSIC curriculum model is designed to be used by engineering schools, both in the USA and the EU. This model will address the nations' needs for researchers and developers of real-time safety-critical systems who are capable of engaging in projects spanning the nations' boundaries and promoting a student-centered, transatlantic dimension to higher education and training.

Introduction

Systems like aircraft avionics, air traffic control, space shuttle control, medical equipment, and nuclear power stations are heavily software-centric, implementing reactive and time-critical software, where safety is the issue and the margin for error is narrow. It is vital for future software developers to understand basic real-time application concepts: timing, concurrency, resource sharing, inter-process communication, interrupts and handling of external devices are of primary importance. The area of real-time safety-critical control systems is one of the most challenging fields of computing, relying on designs developed according to the latest advances in science and modern principles of engineering practice.

The study discussed in this paper is focused on the creation of an international curriculum framework centered on RSIC – an important aspect of the computer-system-control-software engineering education [1,2]. The study explores the 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. This model will address the nations' needs for researchers and developers of real-time safety-critical systems who are capable of engaging in projects spanning the nations' boundaries and promoting a student-centered, transatlantic dimension to higher education and training. RSIC is an increasingly important aspect of computer, system, control, and software engineering education. This study explores the mechanism for involving students from multilingual, geographically separated institutions in a coordinated educational experience. It will expose them to the problems, methods, solution techniques, infrastructure, technologies, regulatory issues, and tools in the domain of dependable real-time safety-critical software-intensive control systems.

The last twenty years have witnessed significant advancements in the state of computer science education (and in related fields such as computer engineering, information systems, and software engineering). The Association for Computing Machinery (ACM), the IEEE Computer Society (IEEE-CS), and the Computer Sciences Accreditation Board (CSAB) have provided guidance in developing viable and dynamic quality curricula. Degree programs have moved from language and coding-centred 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 have produced a set of guidelines for curricula in computer engineering, computer science, information systems and software engineering [3], which advise the inclusion of a significant amount of software engineering theory and practice. Unfortunately, we still do not see many programs which devote considerable time to software engineering areas essential to effective commercial software development. This is especially true for the education required to develop software-intensive systems where time-criticality, safety and reliability are key issues and the margin for error is narrow. It is imperative that software developers understand such basic real-time concepts as timing, concurrency, inter-process communication, resource sharing, interrupts handling, and external devices interface. Hardware and software used to coordinate various computer-controlled functions has become larger and more complex to meet the increasing computational demands. The need to understand the system implications of the software engineering activity is imperative for creation of such real-world software. The same observation can be extended to nearly all areas of modern computing application from home appliances to banking, from toys to nuclear reactor controls, from entertainment gadgets to medical equipment.

In the light of decreasing computing enrollment and the outsourcing gloom, questions like: "will proficiency in both computer science and communications give students a global edge?" were asked [4]. Similarly, Humphrey and Hilburn [5] observed: "Because of the growing impact of software and its historically poor performance in meeting society's needs, the practice of software engineering is in need of substantial changes. One challenge concerns preparing software professionals for their careers; the field must drastically change its approach to software engineering education if it hopes to consistently provide safe, secure, and reliable systems. ". It is time to undertake a comprehensive analysis of the computing education.

We feel it is imperative that software developers understand basic real-time concepts of timing, concurrency, inter-process communication, resource sharing, hardware interrupts handling, and external devices interface. Industry needs computing graduates with knowledge of dependable time-critical reactive systems and those who understand how the software will interact with the operating system and the environment. In addition, they need to be able to work as part of a multidisciplinary team and meet rigorous engineering process and certification standards. It may also be necessary for them to function in multinational companies. These issues need to be integrated into computing curricula becoming potentially a part of several courses; each course can contribute to the overall objective of understanding real-time dependable software-intensive systems.

Edger Dijkstra's statement: "Computer Science is no more about computers as astronomy is about telescope"[6] sparked an early debate on teaching computing science, pointing to a fundamental problem: if a student can understand separate areas like operating systems,

compilers, programming languages, and database systems - can he or she understand how a computing system functions as a whole. Incidences of safety violations and security attacks attributed to computer systems show that the interaction of various components is the primary issue. As Dijkstra noted, a computer specialist needs to apply a systems thinking approach with emphasis on the whole system functionality and the interdependence of its various components. Such ideas should be discussion topics for future software engineering curricula.

The research presented in this paper, engaging faculty of universities from four countries (USA, Poland, France, and Czech Republic) uses a two-pronged approach. The first includes active partner collaboration on identification of the learning objectives and outcomes, description of the curriculum core and supporting units, development of guidelines on the implementation and assessment, identification of the technology infrastructure, and the description of faculty and staff requirements, pedagogy and delivery concepts, accreditation issues and constraints, etc. On-site research by the project faculty and selected students is enhanced by frequent communications and dedicated working sessions at the partners' sites. The second part of the approach is a practical case study on how the proposed framework can be implemented by the partner institutions. This will include identification of existing or easily modifiable courses, which can be used as units in the RSIC curriculum. The case study will also include a description of the laboratory infrastructure, necessary administrative procedures (admission, scheduling, and credit transfer), an assessment methodology, and experimental development and delivery of a selected RSIC unit within the partners' institutions. This experimental concurrent delivery will not include student mobility and engage only on-site students.

Educational Objectives and Outcomes

There is a general agreed upon set of non-technical skills and behaviors expected from engineering school graduates (oral and written communications, professional ethics, team skills, etc.). The starting point for designing a specific program curriculum is to identify the technical knowledge areas and skills required from the graduating students. The program educational objectives can be defined in terms of the expected graduates' proficiency, specifying the profile of graduates and their competencies. An often used phrase defining the objectives is that the graduates of the program "be able to" perform certain specific tasks. Examples may be: analyze the problem, elicit requirements, design a circuit, apply a method, use a tool, etc.

There are two popular means to define an objective. One is a "know how" objective in terms of describing a task one performs. Another is a "knowledge" objective by describing a topic that one understands and is able to convey knowledge about. For example:

- "Know How" to manage database: e.g. to install the database software, to manage users, to create/update/delete database record, etc.
- "Knowledge" of a database concept and model: e.g. to describe the attributes of a database logical model, to give examples of such models, to identify security vulnerabilities, to describe SQL syntax, etc.

The Accreditation Board of Engineering Technology (ABET) defines Program Educational Objectives (PEO) and Program Outcomes (PO). Program Educational Objectives are broad statements that describe the career and professional accomplishments that the program is

preparing graduates to achieve. An example would be: “graduates will pursue successful careers as professional software engineers”. Program Outcomes are narrower statements that describe what students are expected to know and be able to do by the time of graduation. The PO relate to the skills, knowledge, and behaviors that students acquire in their matriculation through the program. An example would be: “graduates will be able to work effectively as part of a software development team.” The PO can be assessed during the course of studies and immediately after students’ graduation.

The graduates of any high quality engineering program are expected to meet the following general program educational objectives:

[A] Demonstrate professionalism in their work and grow professionally through continued learning and involvement in professional activities.

[B] Contribute to society by behaving ethically and responsibly, and by incorporating knowledge of history and culture into one’s professional decisions.

[C] Communicate effectively in oral, written, and newly developing modes and media.

[D] Assume a variety of roles in teams of diverse membership.

The major areas of proficiency for the graduates of an international RSIC program (3-4 years of university/college education) have been identified based on the results of industry surveys and discussion at the consortium meetings. These areas expand the general objectives with those specific to the RSIC program:

[E] Demonstrate understanding of analysis and design as applied to modern software-intensive control systems

[F] Demonstrate understanding of processes and techniques and the role of modern engineering tools necessary to engineering practice as applied for creation of software-intensive systems

[G] Demonstrate understanding of quality assurance and hardware/software integration for creation of safe and dependable systems

The Program Outcomes are characteristics/abilities of the graduating students that can be evaluated at the program completion:

1. An ability to apply knowledge of mathematics, science, and engineering to solve technical problems
2. An ability to design and conduct experiments, and an ability to analyze the data
3. An ability to analyze and understand the operation of a control system or component to meet desired needs (feedback, stability, system dynamics, robustness)
4. An ability to apply advanced software engineering techniques to implement real-time concepts (timing, scheduling, concurrency, synchronization)
5. An ability to support assurance of the quality of a software-intensive system across its life-cycle including assurance of its dependability using established standards and guidelines (verification, validation, testing, safety, reliability, security, standards, guidelines)

6. An ability to integrate hardware and software on variety of platforms with various interfaces and protocols
7. An ability to use a defined lifecycle process in development of software-intensive system
8. An ability to function on multi-disciplinary teams
9. An understanding of professional and ethical responsibility
10. An ability to communicate effectively
11. An ability to work effectively in an international environment
12. An understanding of the impact of engineering solutions in a global and societal context
13. A recognition of the need for and an ability to engage in life-long learning including ability to pursue graduate studies
14. An understanding of contemporary issues in software engineering, especially in the RSIC area

The program outcomes help to identify the topics necessary for preliminary curriculum design. It is critical to understand the role of general education requirements which complements the engineering facet of the curriculum and facilitates the main objective of university mission: to produce valuable and contributing members of the society.

Curriculum

Despite well established computing curricula and the variety of excellent engineering offerings in colleges and universities on both sides of the Atlantic, at this time, there is no international, interdisciplinary curriculum that directly focuses on real-time control systems, dependable software development, safety, reliability and the certification issues in highly regulated industries like aerospace, medicine, transportation, and nuclear energy – a curriculum that would also include globalization aspects of the modern engineering profession.

The goal of the proposed curriculum is to expose students to problems, methods, solution techniques, infrastructure, technologies, regulatory issues, and tools in the domain of dependable real-time safety-critical software-intensive control systems. The study will lead to creation of such a curriculum framework, the identification of implementation and assessment mechanisms, a collection of data necessary to evaluate the process, and guidelines for expansion of the proposed approach to other engineering programs.

The basic assumption is that the students who join the program have appropriate pre-requisite preparation including mathematics (calculus, linear algebra, differential equations, probability statistics, and discrete math), college level engineering physics, introduction to digital systems and computer organization, and computer programming (including rudimentary data structures and operating system principles). The challenge is whether or not such pre-requisite structure can be accomplished within the course of only a three-year curriculum, as it is the practice in some European countries.

The industry surveys and discussion on the RSIC curriculum outcomes and objectives allowed us to identify five focus areas. The example topics (not an exhaustive list) for the first two topic

areas (Software Engineering and Digital Systems) are additionally assigned to either a Basic (B) or an Advanced (A) category. Please note that there may be more than one course in a specific topic area on the same level.

1. Software Engineering
- Basic Level: programming and the discipline of software engineering, system/software lifecycle, project management and planning, requirements solicitation, methods and practices
- Advanced Level: modeling and formal representation, architectural and detailed design with appropriate notations and tools, software construction, testing, quality assurance, and maintenance.

2. Digital Systems
- Basic Level: digital system concepts and operation, design of combinatorial and sequential circuits, concepts and operation of microcontrollers/microprocessors, assembly language, rudimentary interfacing and exception handling.
- Advanced Level: large scale integration devices and tools, interfacing, advanced memory management, fault tolerant hardware

3. Computer Control
- Basic Level: concepts of feedback control, time and frequency domains, Laplace and Z-transforms, state analysis, stability, controllability and observability, controller design, control algorithms, optimization, use of tools and applications, etc.

4. Real-Time Systems
- Basic Level: timing and dependability properties of software intensive systems, RTOS concepts and applications, concurrency, synchronization and communication, scheduling, reliability and safety, etc.

5. Networking
- Basic Level: data communication, network topology, analysis and design, information security, algorithms, encryption, bus architectures, wireless, etc.

The discussion strongly suggests that there shall be an obligatory capstone team project to bring together the knowledge and skills acquired during the course of studies.

The consortium of four universities discussed the institutions' curricula, which are dedicated to preparation of specialists to work in similar RSIC domains. The discussion exhibited significant differences (Table 2). The undergraduate programs could be completed between six and eight semesters. The number of weeks in semesters ranged from 13 to 16. The number of hours/week ranged from 18 to 30 (considering also some discrepancies on the way how the hours are accounted for). The total number of curriculum contact hours range between 1,900 and 3,150.

The methodology uses the European Credit Transfer and Accumulation System (ECTS) (http://ec.europa.eu/education/programmes/socrates/ects/index_en.html#1) as the measure of student workload. The calculation of the ECTS hours includes only the hours students spend in a direct contact with the instructor in terms of the lectures, recitations, labs and exams. It does not

include student time used to study, prepare or consult with the instructor outside the classroom. It is expected that for each contact hour the student would need at least two hours for study and preparation. The data collected from the four partners identifying the workload necessary to successfully complete the undergraduate program (e.g., Baccalaureate level, B.Sc.) are presented in Table 1.

Table 1: Approximate workload for current computing/control degree programs

school	# of semesters	# of weeks	total hours	hours/week	ECTS
ERAU/USA	8	16	1900-2200	~16	N/A
AGH/Poland	7	15	2400-2900	~25	30
BUT/Czech Rep.	6	13	2300-2450	~30	30
LAG/France	7	15	2900-3150	~29	30

It is critical to identify the minimal number of contact hours required to complete the entire RSIC program and the percentage of effort assigned to the following four curriculum categories:

- General education (languages, humanities, social science)
- Math and Science (mathematics, physics)
- Basic (programming, computer science fundamentals, control fundamentals, software engineering fundamentals, digital fundamentals)
- Advanced (software engineering advanced, digital advanced, real-time, advanced control, networking, project)

The assignment of the workload into these four categories of courses was based on a typical related program in the partner curricula (Table 2).

Table 2: Assignment of the workload into four categories
(Approximate data from a typical curriculum sample)

	General Education	Math and Science	Basic	Advanced	TOTAL
ERAU, USA	384	512	528	624	2,048
AGH, Poland	315	555	990	1,090	2,850
BUT, Czech R.	208	312	845	1,040	2,405
LAG, France	294	784	980	882	2,940
Average	**300**	**541**	**836**	**909**	**2561**
% distribution	***12*%**	***21*%**	***33*%**	***35*%**	***100*%**

The proposed draft curriculum framework vertical outline for RSIC, which matches the partners' current curriculum framework, includes the following:

Year 1: math, programming, digital systems
Year 2: math, physics, software engineering, microprocessors
Year 3: math, software engineering, control, real-time systems, networking
Year 4: advanced courses, team projects

The proposed RSIC curriculum contains, as a minimum, the following courses:

Basic Level:

- Programming (B)
- Software Engineering (B)
- Digital Systems (B)
- Computer Control (B)
- Real-Time Systems (B)
- Networking (B)

Advanced Level:

- Advanced Control (A)
- Advanced Digital Systems (A)
- Advanced Software Engineering (A)
- Advanced Networking (A)
- Introductory Design Experience (A)
- Capstone Design Experience (A)

Assessment

To help ensure achievement of program educational objectives and outcomes, an assessment and evaluation process, based on student performance and other indicators, must be in place. The assessment process for Program Educational Objectives is a long-term process based on such instruments as surveys of past alumni and employers, alumni focus group meetings, examination of successful industrial projects involving program alumni, etc.

The assessment process for Program Outcomes may involve such instruments as an annual review of student performance in selected indicator courses by program faculty, feedback from graduating seniors and recent graduates, feedback from an industry advisory board, activities of the department curriculum committee, analysis of data reflecting student cooperative education activities, involvement in professional societies, student choice of minor programs, student Portfolios, etc.

The indicator courses, defined by the faculty, are critical in assessing program outcomes. Each program outcome should have one or more courses that contribute to its achievement. Performance in indicator courses also provides information to the faculty regarding performance in prerequisite courses.

A survey of graduating seniors and the exit interview can be a source of useful information regarding the curriculum. While senior information is valuable, graduating seniors may lack sufficient context to correctly identify the degree to which the program achieves program outcomes; thus, information from the senior survey is considered only moderately reliable. In contrast, feedback from the program alumni employed for two or more years, as well as the feedback from industry employing graduates, provides better evidence of the degree to which the program has achieved the desired outcomes.

In Table 3, a proposed process for the assessment and evaluation of achievement of program educational objectives and program outcomes is described. This process is meant to provide a

framework for carrying out assessment and evaluation; the actual process might vary from one RSIC program to another. The process has been tested for over a decade in one of the consortium partner organization supporting successful ABET accreditation of two engineering programs.

Table 3: Proposed Assessment Process

Step	Description	Schedule
Purpose	This table provides a high-level description of the steps and activities that are part of the assessment process.	
Determine Constituencies	A discussion at the faculty meeting identifies the constituencies for the RSIC program. (e.g., students, employers, faculty)	Initially
Determine Objectives and Outcomes	Based on the needs of the constituencies, the faculty, determine RSIC Program Educational Objectives and Program Outcomes	Initially
Determine Assessment Items and Process	The program faculty identify what will be assessed and how and when achievement of objectives and outcomes will be evaluated (with the use of elements such as indicator courses, surveys, interviews, data collection methods)	Start of Assessment
Collect Data and Opinion	Information and opinion about achievement of Program Outcomes is collected.	Annually
Assess Program Outcomes	The data collected is used to assess whether the Program Outcomes are being achieved.	Annually
Evaluate Program Educational Objectives	The data collected is used to evaluate whether the PEOs of the RSIC program are achieved. There is a determination of whether the objectives need to be modified.	Every Three Years
Modify Program	Based on outcomes assessment and on the results of program educational objectives evaluation and review, the faculty may make changes in the program, its educational objectives, and/or the program outcomes.	Every Three Years
Evaluate Assessment Process	The faculties evaluate the effectiveness of the assessment process and make changes as appropriate.	Every Three Years

Conclusions

Creation of RSIC systems engages nearly all 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. We believe the RSIC curriculum project has the potential to have broad impact on the future of engineering education.

The project is also intended to strengthen international co-operation and the global links in engineering education. An interdisciplinary specialization in RSIC was selected to produce not only educational artifacts in a domain in high demand by industry, but also (what is more important) a process and a methodology for creation of engineering programs with a compatible

quality assurance and assessment process. The intention of the research is prepare a base for graduates of such programs to be better prepared for projects requiring interdisciplinary and multicultural viewpoints. This enhances mobility of the future workforce and facilitates the student advancement and future career changes.

References

1. Wojciech Grega, Andrew Kornecki, Miroslav Sveda, Jean-Marc Thiriet,(2007), Developing Interdisciplinary and Multinational Software Engineering Curriculum, *Proceedings of the ICEE'07*, Coimbra, Portugal, Sep. 2007
2. Andrew Kornecki, Wojciech Grega, Miroslav Sveda, Jean-Marc Thiriet, Thomas Hilburn, ILERT – International Learning Environment for Real-Time Software Intensive Control Systems, *Proceedings of RTS'07 - International Conference on Computer Science and Information Technology*, Wisla, Poland, Oct 2007
3. ACM/IEEE-CS Joint Task Force on Computing Curricula Editor, *Software Engineering 2004,Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, August 2004. http://www.acm.org/education/curricula.html
4. David A. Swayne, Qusay H. Mahmoud, Wlodek Dobosiewicz, An "Offshore-Resistant" Degree Program, *Computer,* Vol. 37, No. 8, pp. 104, 102-103, Aug., 2004
5. Watts Humphrey, Thomas Hilburn, T., The Impending Changes in Software Education, *IEEE Software*, Vol 19, No 5, pp. 22-24, September / October, 22 – 24, 2002.
6. Edgar W. Dijkstra, On the cruelty of really teaching computer science, 1988 http://www.cs.utexas.edu/~EWD/ewd10xx/EWD1036.PDF