BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
Vol. 56, No. 4, 2008

Dependable and certifiable real-world systems – issue of software engineering education

A.J. KORNECKI*

Computer and Software Engineering Department, Embry Riddle Aeronautical University
600 S. Clyde Morris Blvd, 32114 Daytona Beach, FL, USA

Abstract. Embedded software and dedicated hardware are vital elements of the modern world, from personal electronics to transportation, from communication to aerospace, from military to gaming, from medical systems to banking. Combinations of even minor hardware or software defects in a complex system may lead to violation of safety with or even without evident system failure. a major problem that the computing profession faces is the lack of a universal approach to unite the dissimilar viewpoints presented by computer science, with its discrete and mathematical underpinnings, and by computer engineering, which focuses on building real systems and considering spatial and material constraints of space, energy, and time. Modern embedded systems include both viewpoints: microprocessors running software and programmable electronic hardware created with an extensive use of software. The gap between science and engineering approaches is clearly visible in engineering education. This survey paper focuses on exploring the commonalities between building software and building hardware in an attempt to establish a new framework for rejuvenating computing education, specifically software engineering for dependable systems. We present here a perspective on software/hardware relationship, aviation system certification, role of software engineering education, and future directions in computing.

Key words: dependable systems, aviation systems certification, engineering education.

1. Introduction

In a recent U.S. National Science Foundation announcement a question was posed: "How can real-world systems be designed, built, and analyzed in elegant and powerful new ways?" This paper will elaborate on the three aspects that provide the background and may address the possible answer to the raised questions: (a) the scientific underpinnings of system construction, (b) the methodology and process of system development, and (c) education and training of the stakeholders (i.e. developers) and users of the systems.

Computer technology is the foundation for nearly all facets of human endeavor from games and consumer electronics to medicine and space exploration. Embedded software controls transportation, communication, aerospace, military, and medical systems – all of which must be dependable. Dependability, as defined by the IFIP 10.4 Working Group on Dependable Computing and Fault Tolerance: "the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers" [1], is the characteristic property of such systems.

Software complexity has grown to the point that verification of system dependability may be problematic. Combinations of even minor software defects in a complex system may lead to significant system failures. And since reliability is not an equivalent of safety, it can be violated even without evident system failure in complex systems. The broad discipline of computing has changed radically in the past two decades, both being affected by other domains as well as bearing significant impact on other fields of human endeavor. Modern computing involves understanding complex interactions with the real-world, integration of systems, and the need to understand the multidisciplinary nature of the computing, which requires a combination of expertise ranging from control to electrical to computer to software engineering. In addition to keeping the pace with the rapid progress of technology, the critical issue is enforcing engineering discipline when developing, verifying and validating software intensive complex systems.

One of the major problems that computing profession faces is the lack of a unified theory and approach to combine often dissimilar viewpoints presented by (a) discrete and mathematical underpinnings of computer science, and by (b) computer engineering, which focuses on building real systems considering spatial and material constraints of space, energy, and time. Modern embedded systems include both viewpoints: microprocessors running software and programmable electronic hardware created with an extensive use of software. The gap between science and engineering approaches is glaringly visible in education, where individual departments and programs tend to stay within their comfort zones leading to stove-piping of the academic programs, limiting the graduates to hearing only one side of the story.

While developing software intensive systems, the real-world faces challenges in two focus areas. Time-critical software intensive systems require rigorous engineering methods of analysis and certification to create architectures and implementation mechanisms whose composite properties and behaviors must be certifiably dependable. Furthermore, evolu-

*e-mail: kornecka@erau.edu

tion of the software-dependent world requires the knowledge and tools needed for people to participate fully in the design, development, and use of real-world systems as end-users, software engineers, and other stakeholders.

There are two directions where improvement in the current state of affairs regarding software intensive dependable systems may be sought:

- **Engineering processes and methods**. Based on established software engineering practices, the processes and methods of designing, building, and analyzing software for real-world autonomous and complex systems of the future need to consider the future societal and technological trends and thus facilitate a software development environment in the next decades. The use of software tools and modeling techniques for engineering activity leading to not only creation of computer software but also creation of complex programmable logic devices needs to be explored – in a search for potential unification of these two activities.
- **Educational pedagogy**. Based on established process-driven software engineering curricula, innovative pedagogy for educating students and training the workforce needs to be explored. The major consideration is integration of the software and hardware tracks to facilitate the concept of building modern dependable systems from the system engineering perspective. New educational ideas and activities supporting learning and application of scientific principles and engineering methods for the design, construction, and analysis of real-world software systems need to be identified.

There are numerous stakeholders intimately interested in the future of the profession and its preparation for the challenges of this century. They include not only computing faculty and academic administrators but also recognized nation leaders and futurists in the field of computing, commercial software development companies, professional computing societies and trade organizations, government policy makers and funding organizations, national research and industrial laboratories, etc.

The paper will discuss three aspects to address the issue of creating dependable software-intensive real world systems. Since modern systems are a tightly knit combination of hardware and software, we discuss their mutual relationship and the impact of the scientific foundations pointing out well known distinction between science and engineering. Aviation is an example of domain where the dependability of the system is of primary importance. The paper elaborates on the methodology and the process used to assure dependability both from the perspective of software and hardware guidance. Subsequently, we explore the role of software engineering and the future directions of computing for the real world.

2. Complex systems – the software/hardware relationship

Since early 90's in the Computer and Software Engineering department we hold regular yearly meetings of the Industry Advisory Board. The feedback received from the industry representatives, based on their experience with freshly hired graduates, shows that the majority of computing curricula do not address an integrative view of the discipline; neither have they matched industry needs and challenges raised by ever expanding and increasingly complex applications. The systems that best fit this category are in the aviation and aerospace, medical, transportation, and nuclear fields, where software plays a critical role and its dependability is of paramount importance. The system approach is the starting point. However, there is need to address the critical issues related to the computing domain.

Computer Science (CS) and Software Engineering (SE) programs produce graduates that are typically engaged in developing large industry-strength software systems. Due to their hardware focus, Computer Engineering (CE) programs graduates are prepared for designing hardware systems which typically include significant software components. The three programs differ in the points of emphasis: development of new theoretical ideas and algorithms (CS) vs. development of large and complex software systems (SE) vs. small embedded software and device drivers (CE). For contribution to building real world systems all three domains have need of good engineering practices. The development of dependable systems is the common denominator of the three domains. The proliferation of software-intensive systems in everyday life, forces industry to hire engineers familiar with time-critical reactive dependable systems, those who understand the intricacies of hardware-software interaction, the role of the computer operating system, and its environmental impact.

The vital issue in the future of software engineering, as related to the development of dependable safety critical systems, is close relation between what conventionally was considered as separate categories: software vs. hardware. Software application designers focus on the development of programs that run on microprocessors, and often overlook programmable logic devices (and the ever popular Field Programmable Logic Arrays, FPGA). FPGA is a prefabricated integrated circuit that can be configured to implement a particular design by downloading a sequence of bits. In that sense, a circuit implemented on an FPGA is literally software. However, circuit designers are still known as hardware specialists, and the algorithms ported to circuits are still known as hardware algorithms. Treating circuits as "hardware" creates problems in computing system development, in particular for embedded systems. The reason is that the graduates of typical computer science program – employed often as software engineers – are not getting adequate exposure to hardware operation. The issue extends beyond circuits and hardware and into the concept of two dissimilar computation models: (a) software focuses on temporal models based on state machines, communicating processes, and sequences of instructions ordering tasks in time and (b) hardware focuses on spatial models with data flow graphs and logic circuits executing concurrently in a parallel or pipelined fashion. Vahid observes that due to their educational bias, software developers are accustomed to defining algorithms and subroutines, "... but they're typically

weaker at creating models that also involve some amount of spatial orientation, like parallel processes, data-flow graphs, or circuits, largely because computing education in universities tends to emphasize the former with little attention given to the latter. Yet with embedded systems continuing to grow in importance, such imbalance can't persist much longer" [2].

Henzinger and Sifakis [3] write about the need to renew the computer science curriculum. The different design principles and approaches distinguish the approaches used by hardware and software designers. Software designers see the system in terms of dynamic objects and threads constituting sequential building blocks or virtual machines with their semantic interpretation of a computational model. On the other hand, hardware designers compose the system with parallel building blocks representing physical entities with appropriate data flows between them. The blocks have formal transfer function semantics described by a set of equations forming an analytical model. The computational and analytical models support two dissimilar design processes.

With the rapid progress of microelectronic technology we may expect further expansion of dedicated and programmable hardware that will be developed and verified using complex software tools. The software not only consists of the system and application programs but the complex software used to develop and verify programmable logic circuits. The software engineering principles and approaches may need to be applied to the hardware domain. On the other hand, the concepts well accepted by the hardware designers' community, like concurrent execution of spatial circuit, may influence future design of massively concurrent software. The hardware-software co-design and the system approach, necessity of understanding both sides of the embedded system are the basic tenets of the education of future dependable system developers. Considering it necessary to close the gap between hardware (electrical and computer engineering) and software (software engineering and computer science) constitutes significant paradigm shift in the education of the future cadre of dependable system developers.

3. Aviation example – the practice of building dependable systems

Aviation systems, both airborne and ground, (e.g., flight controls, avionics, engine control, air traffic control) are typical examples of safety-critical, real-time systems. Such systems continue to become more complex and are extremely software intensive thus getting necessary attention of modern software engineering community. Systems like that often operate in environments with diverse ranges of temperature, humidity, air pressure, vibration and movement, and are subject to the affects of age, maintenance and weather. Typical characteristics required of such systems are reliability, fault tolerance, and deterministic timing guarantees. Both hardware and software for such systems must address these issues; hence the concept of certification.

The term "certification" in software engineering is typically associated with three meanings: certifying product, process, or personnel. Product and process certification are the most challenging in developing software for real-time safety critical systems, such as flight control and traffic control, road vehicles, railway interchanges, nuclear facilities, medical equipment, implanted devices, etc. These are systems that operate under strict timing requirements and may cause significant damage or loss of life, if not operating properly. The general public has to protect itself, and governments and engineering societies initiated establishing standards and guidelines for software developers to follow in designing software for such systems in several regulated industries, including aerospace, avionics, automotive, medical, nuclear, railways, and others.

RTCA, Incorporated, is a not-profit corporation formed to advance the art and science of aviation and aviation electronic systems for the benefit of the public. The main function of the RTCA is to act as a Federal Advisory Committee to develop consensus-based recommendations on aviation issues, which are used as the foundation for Federal Aviation Administration Technical Standard Orders controlling the certification of aviation systems.

In 1980, the RTCA convened a special committee (SC-145) to establish guidelines for developing airborne systems and equipment. They produced a report, "Software Considerations in Airborne Systems and Equipment Certification", which was subsequently approved by the RTCA Executive Committee and published in January 1982 as the RTCA document DO-178. After gaining further experience in airborne system certification, the RTCA decided to revise the previous document. Another committee (SC-152) drafted DO-178A, which was published in 1985. Due to rapid advances in technology, the RTCA established a new committee (SC-167) in 1989. Its goal was to update, as needed, DO-178A. SC-167 focused on five major areas: Documentation Integration and Production, System Issues, Software Development, Software Verification, and Software Configuration Management and Quality Assurance. The resulting document, DO-178B, provides guidelines for these areas [4]. The document identifies a set of objectives to be met depending on the criticality of the specific system, in terms of level of assurance from A (the most critical) to E (not critical), as identified by preceding system safety assessment.

Another RTCA document DO-254 [5] prepared by SC-180, was released in 2000 addressing design assurance for complex electronic hardware. The guidance is applicable to a wide range of hardware devices, from integrated technology hybrid and multi-chip components, to custom programmable micro-coded components, to circuit board assemblies (CBA), to entire line-replaceable units (LRU). This guidance also addresses the issue of commercial off-the-shelf (COTS) components. The document's appendices provide guidance for data to be submitted, including: independence and control data category based on the assigned assurance level, description of the functional failure path analysis (FFPA) method applicable to hardware with the highest design assurance levels (DAL), and discussion of additional assurance techniques like formal methods to support and verify analysis results.

The implementation of Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) systems has resulted in increased interdependence of systems providing Air Traffic Services (ATS) and systems onboard aircraft. CNS/ATM systems include ground, airborne, and space-based systems. DO-278, resulting from deliberations of RTCA Special Committee 190 (SC-190), provides guidelines for the assurance of software contained in non-airborne CNS/ATM systems [6]. The guidance applies to software contained in CNS/ATM systems used in ground or space-based applications shown by a system safety assessment process to affect the safety of aircraft occupants or airframe in its operational environment. Similar to the DO-178B the guidance is objective-based where specific objectives must be met depending of the level of system criticality identified by the system safety assessment.

It should not be construed that the above approach used by the civil aviation system developers is perfect. However, the discipline and the process mandated by the guidance, which is hardly ever taught in the academic settings, allows developers to make arguments about system dependability while making flying population to take flights and treat air travel with more confidence.

4. Role of software engineering education

The statement by Dijkstra "Computer Science is no more about computers as astronomy is about telescope" [7] had sparked an early debate on teaching computing science pointing to a fundamental problem one of consistency between the understanding of separate areas like operating systems, compilers, programming languages, and database systems with understanding how a computing system functions as a whole. Incidents of safety violations and security attacks attributed to computer systems show that the interaction of various components is the primary culprit. As Dijkstra wrote, a computer specialist needs to apply a more systems – oriented approach with emphasis on the whole system functionality and the interdependence of its various components. One of the software engineering curriculum guidelines [8] states that "The curriculum should have a significant real-world basis". The question is how to best provide this "real-world" experience?

The last twenty years have witnessed noteworthy advancements in the state of computer science education (and in the 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 encouragement, support, and guidance in developing quality curricula that are viable and dynamic. Degree programs have moved from the initial 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 [9], which recommend the inclusion of a significant amount of software engineering theory and practice. The demand for adequately prepared software engineers is growing. However, with hundreds of computing programs in the U.S. colleges and universities, there is less than twenty ABET accredited software engineering programs which devote considerable time to software engineering areas essential to effective commercial software development. The demand is especially high 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. Certainly, there are excellent computer engineering programs that provide their graduates with exposure to such concepts. However, due to the hardware-focused nature of such programs, the software development is typically not treated with a sufficient depth.

Decreasing computing enrollment and the outsourcing gloom provoked questions like: "will proficiency in both computer science and communications give students a global edge?" [10]. Similarly, Humphrey and Hilburn [11] observed the need to undertake a comprehensive analysis of computing education: "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".

Software Engineering programs often "received deep criticism and subjective evaluation by many traditional computer scientists who see them merely as an opportunity to provide industrial training in programming (but who fail to understand the complexities of software)" [12]. An additional facet the profession is struggling with is the need for understanding the hardware platform and environment interfaces and thus related dependability issues for software-intensive systems. CMU-ISRI technical report [13] identifies principal foundations of software engineering in terms of computer science concepts and engineering knowledge complemented by social and economic context of the engineering effort. The report also identifies core competencies, capabilities, and pedagogical principles.

Computer science is not the same as software development. Incorporating the practices of software engineering into undergraduate computing programs, based on solid background provided by conventional computer science courses, is important for the software industry. Introduction of process scripts, requirements, design and code reviews, and metrics familiarizes the students with the software engineering discipline. a rigorous repeatable process helps to create an environment where the software products are developed more efficiently and with fewer defects [14]. It is also critical to realize that the computing specialists are part of larger community of system developers and provide them opportunity for a domain specific education [15]. However, many excel-

lent computer science academic programs do not include adequate software development component, because elite schools may think that teaching practical skills should be left to the technical vocational institutes and computer training schools.

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. From the perspective of accreditation requirements, the curricula must define program educational objectives describing the career and professional accomplishments that the program is preparing graduates to achieve as well as the related program outcomes describing what students are expected to know and be able to do by the time of graduation.

5. Future directions of real world computing

Alan Kay once said: "The best way to predict the future is to invent it" [16]. He went on to say that things that can be described can actually be built. In the same vain, Watts Humphrey wrote in his column [17] that "... programming is a very effective way to meet many kinds of human needs. As long as people continue devising new and cleverer ways to work and to play, we can expect the growth in computer applications to continue. The more we learn about computing systems and the more problems we solve, the better we understand how to address more complex and interesting problems. Therefore, because human ingenuity appears to be unlimited, the number of programs needed in the world is also essentially unlimited. This means that the principal limitation on the expanding use of computing systems is our ability to find enough skilled people to meet the demands".

Decision software installed in unmanned autonomous systems (UAS) needs to be both reliable and safe. However, trusting decisions made by autonomous control software may require new methods and processes to guarantee safety. The difficulty lies in determining how these intelligent systems will operate in a dynamic environment and with little or no human oversight. New paradigms will be needed to assure safety. Intelligent control adds a whole new dimension to certification issues for flight control technologies; they already involve the most rigorous testing, which embedded computer systems can endure [18]. UAS autonomous control is a revolutionary leap in technology. Such control replaces decision-making that required years of training for human operators. Neglecting autonomous control certification research today will dramatically increase tomorrow's cost of ownership for future users. The Air Force Research Laboratory investigates Verification & Validation (V&V) technologies uniting the aerospace community to address the problem. These issues may also need to be addressed in the education of future software engineers.

The proceedings from the 2007 Conference on Future of Software Engineering presented the state of the art in areas of programming environments, empirical methods, architectural challenges, performance and reliability, testing and analysis, mechatronics, complex systems, academia/industry collaboration, globalization, and educational challenges. The editors write: "Software engineering is a rapidly evolving field of research and practice. It is a highly diverse and vast realm of knowledge, spanning from management and process issues in software development to system issues such as safety, quality, and deployment.... As a result, it is difficult for anyone to follow, even at a high level, how the various elements of software engineering research are evolving and what to expect in the future" [19].

An increase in the use of software systems has a direct impact on the software engineering process being used. Barry Boehm identifies eight future trends of software intensive systems and the implications of these trends on software process [20]. A major debate in software engineering centers on the use of plan-driven processes versus more agile ones. While each is thought to be best applicable to the development of specific systems, the challenge is to develop methods by which a balance can be achieved in systems that lay in between. It becomes also essential to educate software engineers that are not only familiar with each particular method, but are also able to identify and apply the most suitable one applicable to the specific project and organization.

Recent discussion [21] engaging computing faculty from Cornell, Stanford, Princeton, and Berkeley identifies new directions in the computing that may impact education. An interesting angle of the discussion has been to promote the role of computing "...as a sort of universal science. We're beginning to pervade everything" and that computing "...like math, is unique in the sense that many other disciplines will have to adopt that way of thinking. It offers a sort of conceptual framework for other disciplines, and that's fairly new". In the short term, computing innovations may include high-quality machine translation, reliable speech understanding, lightweight, high-capacity e-books, theft-proof electronic wallets, self-healing software, including adaptive networks that reconfigure for reliability; robotics for mine safety, planetary exploration; prosthetics for medical/nursing care and manufacturing, etc. Nanotechnology and quantum computing could well be fundamental ingredients in the next revolution in computing. Massively parallel computation based on swarms of conventional chips underlies another potential revolution. Trustworthy computing will finally overcome its historical market-failure problems and become a commonplace requirement. These innovations often require better tools, extended programming languages and new processor architectures.

With the rapid progress of microelectronic technology, we can expect further expansion of dedicated and programmable

hardware that will be developed and verified using complex software tools. The software consists of not only the system and application programs but also the complex software used to develop and verify programmable-logic circuits. SE principles and approaches might need to be applied to the hardware domain. On the other hand, concepts accepted by hardware designers, such as concurrent execution of spatial circuits, might influence future design of massively concurrent software. Hardware-software co-design, a system-based approach, and the related necessity of understanding both sides of the embedded-system spectrum (that is, hardware and software) are the basic tenets of the education of future dependable-system developers.

6. Conclusions

The future trends in development of both hardware and software, specifically considering the proliferation of the software in the systems, have a growing impact on the environment and our way of life. An analysis of standards, guidelines, and practices for dependable software-intensive systems development, is the foundation for identifying the critical issues and the challenges that industry needs to address in the future. Engineering methods of systems development must consider metrics to permit organizations the assessment of both process and product. Special attention needs to be placed on engineering methodologies requiring close interaction between hardware and software platforms and the resulting system approval/certification from the safety and security perspective. Due to the nature of the future autonomous systems, the issues of engineering systems operating in an ever-changing operational environment will be considered.

Contemporary systems use increasing numbers of computers and dedicated hardware to process the growing amounts of data. Software intensive control systems are distributed connected by an arcane bus structure. For example, a modern aircraft bus supports flight controls, displays, weather radar, data links, propulsion control, actuation terminals, power systems, and fuel and stores management. Hardware and software used to coordinate these various computer-controlled functions has become larger and more complex to meet the increasing onboard computational demands. Most of the control system applications are in safety-critical areas and the controlling software must be highly safe and reliable. Rigorous methods for adaptive software verification and validation must be developed to ensure that software failures do not impact the system operation, to eliminate unintended functionality, and to demonstrate that certification requirements can be satisfied. 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.

There is an immediate need to undertake a gap analysis between the future trends and the existing educational environment. The result of such activity would be a base for creation of a framework for an education and training paradigm that has the objective of producing qualified professionals, capable of working efficiently and effectively on certifiable software intensive projects. The paradigm will focus on dependable systems and will emphasize the importance of computing fundamentals, the software engineering discipline, software/hardware commonalities, and multidisciplinary teams.

The recent work of the U.S. computing community under the National Science Foundation CPATH program confirms a need for the rejuvenation of computing curricula and the injection of a more general computational thinking as the base for understanding the broader impact of computing in modern world. The ongoing collaboration with European partners would also facilitate this impact outside the American educational system. Examples of such could be a recent engagement of the author in the Atlantis Program funded by the European Commission and the U.S. Department of Education [22, 23]. In collaboration with universities in Poland, France, and Czech Republic, the research identifies a framework for engineering education focusing on Real-Time Software Intensive Control Systems and preparing a platform for coordinated curricula and student exchanges.

Acknowledgements. The author owes his interests in both the systems approach and engineering education to his mentor and doctoral thesis advisor, Professor Henryk Górecki. After having spent nearly 30 years abroad, the author still cherishes the memories of his early years as a student. He considers this an opportunity to express his appreciation and gratitude to Professor Górecki for his guidance, wealth of knowledge, and friendship.

REFERENCES

[1] J.C. Laprie. "Dependable computing and fault tolerance: concepts and terminology", *Proc. 15th IEEE Int. Symposium on Fault-Tolerant Computing*, 2–11 (1985).

[2] F. Vahid, "It's time to stop calling circuits 'hardware", *Computer* 40 (9), 106–108 (2007).

[3] T.A. Henzinger and J. Sifakis, "The discipline of embedded systems design", *Computer* 40 (10), 32–40 (2007).

[4] RTCA SC-167, *DO-178B, Software Considerations in Airborne Systems and Equipment Certification*, RTCA Inc., Washington, 1992.

[5] RTCA SC-180, *DO-254, Design Assurance Guidance for Airborne Electronic Hardware*, RTCA Inc., Washington, 2000.

[6] RTCA SC-190, *DO-278, Guidelines For Communication, Navigation, Surveillance, And Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance*, RTCA Inc., Washington, 2002.

[7] E.W. Dijkstra, "On the cruelty of really teaching computer science", http://www.cs.utexas.edu/∼EWD/ewd10xx/EWD 1036.PDF (1988).

[8] ACM/IEEE-CS, "Joint task force on computing curricula editor", *Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, http://www.acm.org/education/curricula.html (2004).

[9] ACM/IEEE-CS, "Joint task force on computing curricula", *Computing Curricula 2005, The Overview Report*, http://www.acm.org/education/curricula.html (2005).

[10] D.A. Swayne, Q.H. Mahmoud, and W. Dobosiewicz, "An 'offshore-resistant' degree program", *Computer* 37 (8), 102–104 (2004).

[11] W. Humphrey and T. Hilburn, "The impending changes in software education", *IEEE Software* 19 (5), 22–24 (2002).

[12] H.Saiedian, D.J. Bagert, and N.R. Mead, "Software engineering programs: dispelling the myths and misconceptions", *IEEE Software* 19 (5), 35–41 (2002).

[13] M. Shaw, *Software Engineering for the 21st Century: a Basis for Rethinking the Curriculum*, Technical Report CMU-ISRI-05-108, Carnegie Mellon University, Pittsburgh, 2005.

[14] T. Hilburn, I. Hirmanpour, and A.J. Kornecki, "The integration of software engineering into a computer science curriculum", *Lecture Notes in Computer Science* 895, 87–98 (1995).

[15] I. Hirmanpour, T. Hilburn, and A.J. Kornecki, "A domain centered curriculum: an alternative approach to computing education", *SIGCSE Bulletin* 27 (1), 126–130 (1995).

[16] A. Kay, "Predicting the future", *Stanford Engineering* 1 (1), 1–6 (1989).

[17] W. Humphrey, "The future of software engineering I", *News@Sei* 4 (1), (2001).

[18] V. Crum, D. Homan, and R. Bortner, "Certification challenges for autonomous flight control systems", *Proc. AIAA Guidance, Navigation, and Control Conference and Exhibit* AIAA2004, 5257 (2004).

[19] A. Dearle, "Software deployment, past, present and future", *Proc. Int. Conf. on Software Engineering, Future of Software Engineering*, 269–284 (2007).

[20] B. Boehm, "Some future trends and implications for systems and software engineering processes", *Systems Engineering* 9 (1), 1–19 (2006).

[21] G. Anthes, "Computer science looks for a remake", *Computer World*, http://www.computerworld.com/careertopics/careers/story/ 0,10801,110959,00.html (2006).

[22] W. Grega, A.J. Kornecki, M. Sveda, and J-M. Thiriet, "Developing interdisciplinary and multinational software engineering curriculum", *Proc. ICEE'07*, 109 (2007).

[23] A.J. Kornecki, W. Grega, M. Sveda, J-M. Thiriet, and T. Hilburn, "ILERT – international learning environment for real-time software intensive control systems", *Proc. RTS'07 – ICC-SIT* 2, 943–948 (2007).