Guest Editorial Informatics and Electronics Education: Some Remarks

I. Introduction

THIS paper focuses on some vexed issues in the preparation of computer science (CS) curricula, including the systematization of the different areas of the technology, the definition of topics and subtopics to teach, their distribution across the various computing courses, and the relationships between them. It is a common opinion that such didactical difficulties are caused by the fast evolution of information technology. To show how this opinion has little ground, a comparison is made between the methods adopted by educators of informatics and of electronics; in fact, also the electronic technology advances at a similarly fast pace. This analysis brings evidence how the experts in electronics have a well-described theoretical basis that can be logically mapped and progressively taught to give a clear understanding of the field. The same cannot be said in CS. Educators in this field are not aided by underpinning theories in an effective manner. Experts have a mass of concepts at hand, but no network of logical connections to order them. This paper does not present a practical solution to these educational difficulties, but observes how the missing assistance of computer theorists may result in unnecessary expenditure of energy and considerable additional effort.

In the following discussion, the terms "computer science (CS)," "information and communication technology (ICT)," "computing," and "informatics" will be used synonymously. This simplified terminology will facilitate the discussion in this paper. Two opening remarks introduce the themes to be examined.

A. First Remark

By the mid-1960s, computer science education (CSE) had become a very active field. The widespread success of computers in business and organizations led to a high demand for skilled practitioners. ICT courses made it possible to acquire the techniques and knowledge necessary to provide vital ICT services to Western economies. The scientific community began to tackle the problems arising from that endeavor; as a result, the amount of contributions in the literature dealing with CSE is immense, and continues to grow. However, CSE researchers' myriad of activities does not always provide proportionally satisfactory outcomes [1].

This paper overlooks the social, motivational, and psychological issues addressed by educators and focuses on the difficulties that could be broadly termed as *content issues.* Content issues cover the definition of CS subject areas, the topics and subtopics to be taught, their distribution across the various computing courses, and the relationships between them. Content issues draw attention to the need for systematization of the different areas of ICT, the identification of common ground between them, and the analysis of the core discipline. This kind of question arises at any level of the educational process. For instance, Yehezkel and Haberman believed that the education programs for majors are inadequate to meet industry needs [2]. The courses presented in many universities lack sufficient emphasis on software engineering notions to provide a formative educational basis for a career as a software practitioner. CSE researchers work to bridge the gap between the subject matter taught in institutions of higher education and that required by the "real world" [3], [4]. Shaw *et al.* saw software processes and methods as an important part of software engineering training that should be more accurately learned [5]. Waychal looked into some topics at the intersection of software engineering and human sciences that are usually overlooked in the lessons that are mostly oriented toward technical arguments [6].

Content issues also emerge in the initial stages of the CS curricula. Problems encountered in introductory lessons of informatics are a common concern in many universities, as evidenced by CSE experts' discussion in the literature. Some thinkers provide insights from the intellectual point of view [7], [8]; experimentalists report from their standpoint [9]; others search for more effective didactical approaches [10]. A survey of research studies dealing with introductory courses in CS can be found in [11]. In conclusion, modern literature reveals CSE experts' perception of content issues.

B. Second Remark

Scholars dealing with content issues frequently note that informatics is a rapidly changing discipline and ascribe the difficulties they encounter to the rapid growth of the technology and the digital market. The view that unites the vast community of ICT educators could be summed up with these terms:

> "*Informatics is a novel and fast evolving discipline. Hence, the pedagogy of informatics meets severe obstacles*." (Statement 1)

The literature exhibits a broad range of positions about this statement. Some make only glancing reference to the dynamic evolution of ICT; it is a background element recognized as the root cause of the problems under discussion. Other writers place the rate of change of CS at the center of their reflections. For instance, the book *Computer Science Education in the 21st*

Digital Object Identifier 10.1109/TE.2016.2528891

Century [12] includes 13 chapters, and it may be said that all the chapters draw inspiration from the rapid and pervasive progress of computers. To show the prevalence of the view expressed in the statement earlier, the author of this paper has selected a sample of papers that treat an assortment of arguments on CSE from the basis that computing evolves rapidly: Reference [13] considers the evolution of introductory courses in universities; reference [14] expresses the need to merge teaching and research to form a scholarship of computing that is an integrated and sustainable whole; Tucker suggests innovative strategic directions in CSE [15]; reference [16] makes ten succinct remarks on computing as an evolving discipline; reference [17] observes that CS reinvents itself every 5–7 years and educators have to continually modify the curriculum, either changing existing courses or introducing new ones. The problems associated with rapid ICT progress were felt by the Association for Computing Machinery (ACM) and other professional bodies, who called for the creation of standardized CS curricula. Their published guidelines, which will be examined in the following, have the explicit goal of remaining valid as technology progresses. In addition, government reports and official documents cite the evolution of ICT as a source of problems in the education area and beyond [18], [19].

Statement 1 could be considered as the statement of a theorem whose first line is a hypothesis and whose second line is the thesis. The assumption "*Informatics is a novel and fast evolving discipline*" is established on the basis of facts and is true beyond any doubt. However—to the best of the author's knowledge—nobody has substantiated the conclusion "*Hence, the pedagogy of informatics meets severe obstacles*," which may be shared in certain situations but is not universally true. No scholar has proved that the end point follows from the premise everywhere.

Mathematicians teach that while a proof is required to show a theorem is true, a single example is sufficient to disprove it. Presently, there are various fast-paced sectors such as nanotechnology, environmental science, and genetics, which do not face great obstacles in their related educational activities. Similarly, electronics—a field very close to and partially overlapping computing—demonstrates that Statement 1 has little ground as a general rule.

II. Fast-Evolving Electronics

Progress in electronics has revolutionized lifestyles and social structures from a number of standpoints [20]. Electronics is at the heart of advanced economies, and no field of industry can function properly without it. A vast range of devices, from familiar electronic appliances such as mobile phones and digital TVs to high-end equipment such as robots, medical appliances, communication satellites, and environmental facilities, depend on these technologies [21]. Electronics continues to gain importance in modern society and even drives the advancement of CS. Despite its rapid growth, electronics courses do not face content issues such as those discussed earlier in Section I-A. Curricula begin with the fundamental principles of electronics—for example, Faraday's law, Kirchhoff's equations, Ohm's law, and Maxwell's formulas—and proceed toward specialized topics [22]. The literature on electronics education (EE) reveals a broad consensus about how it ought to be done. The reader can get an idea of the scarce amount of works dealing with EE from the following experiment. If one enters "learning electronics," the search engine of the IEEE Electronic Library displays six results; if he enters "learning programming," the results are 123 (data accessed January 2016). In electronics, there is broad agreement on the way to design appropriate lessons, and no organization has felt the need to promote the publication of standard curricula and official guidelines similar to those created for informatics.

As electronics technology advances, teachers update their lessons in electronics in a rather straightforward manner. A new discovery does not threaten the overthrow of the entire educational program. For example, electronic power courses can be improved following available criteria [23].

The reader perhaps objects that informatics has developed so much that now it consists of different subdisciplines. However, even electronic experts have incorporated new areas into their discipline—such as bioelectronics, microelectronics and nanoelectronics, photonics, power electronics, quantum electronics, and medical electronics—whose formal introduction into EE did not subvert the broader curriculum. New subject matter replaces obsolete material, or is added to the curriculum without great discussion [24]. New topics give substance to new professional figures and do not raise lively debates comparable to the discussion occurring in CSE [25].

In summary, electronics demonstrates that rapid innovation in technologies does not necessarily pose heavy pedagogical difficulties. EE does not experience obstacles greater than those common to teachers of any traditional discipline. The case of electronics thus proves that Statement 1 has no ground as a theorem.

III. Theories

The parallel between CSE and EE may be felt to be oversimplified. It will be useful to explore the features of the two fields. In 1968, the ACM published an extensive report on the CS curriculum to help teachers organize CS courses in the most appropriate manner. That report was subsequently updated in two further reports issued at approximately 10-year intervals: Curriculum 78 [26] and Curricula 91 [27]. The Institute of Electrical and Electronic Engineers (IEEE) Computer Society published an independent curriculum for CS and engineering in 1983 and subsequently collaborated with the ACM to write curricula in 1991 and 2001 [28]. In 1998, UNESCO requested that the International Federation for Information Processing (IFIP) carry out a curriculum project for different categories of professionals acting or interacting with informatics [29]. This work may be considered a successor of an earlier IFIP/UNESCO curriculum delivered in 1994. A consortium of 11 major companies dealing with computers in Europe published guidelines for experts who prepare courses on computing [30]. More recently, Computing Curricula 2005 [31] produced by the ACM, IEEE, and the Association for Information Systems identified the distinctive features of five disciplines of computing and laid down the skill set that every

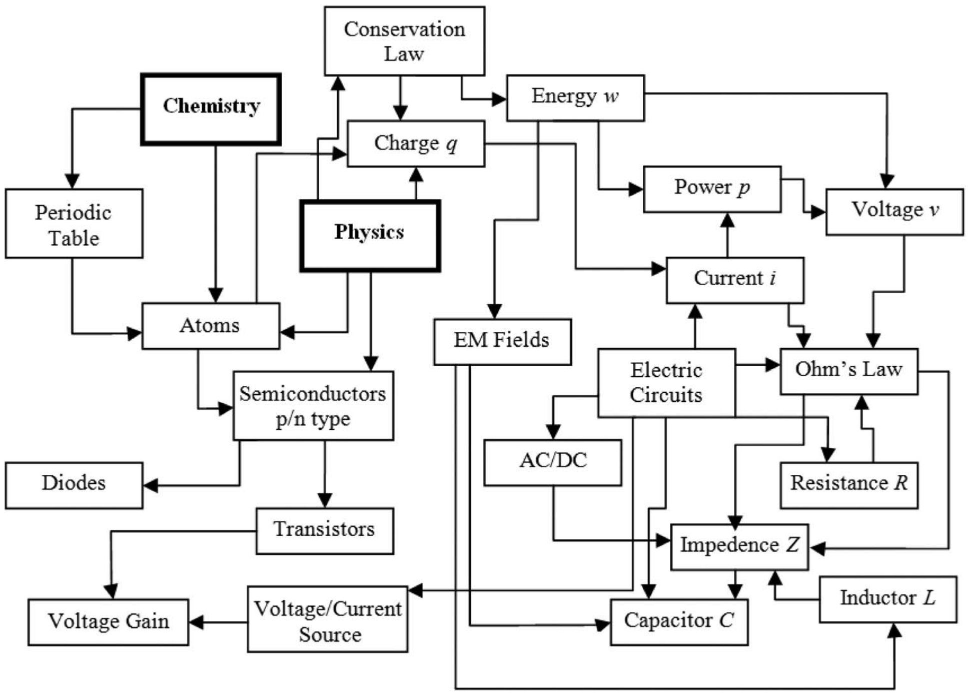

Fig. 1. Concept map of electronics (partial copy from [34], reprinted with permission).

graduate in each respective discipline should acquire. Various professional entities brought forth guidelines for CSE covering topics such as software engineering [32], liberal arts, and humanities. In 2013, the ACM put forward a draft version of the Computer Science Curricula [33], which redefined the knowledge units and provided concrete guidance on curricular structure and development in a variety of educational contexts. In summary, it may be said that the authors of curricula focused on *what* should be taught, rather than on *how*. The experts were deeply involved in solving various content issues, they arranged the knowledge areas and the themes to be taught. They also established the relative importance of the various topics, their sequential and logical relationships, and the targets to be reached. A typical challenge is the contrasting needs of students majoring in CS and those just wishing to program or to use a spreadsheet. The introductory approaches have commonalities, but they also have key differences. This issue is well expressed at the beginning of [33]: "*An important challenge for introductory courses, and a key reason the content of such courses remains a vigorous discussion topic after decades of debate, is that not everything relevant to a computer scientist (programming, software processes, algorithms, abstraction, performance, security, professionalism, etc.) can be taught from day one*." Such heavy efforts and vexed debates on content issues are practically absent in EE—why?

The answer appears to be self-evident. Electronics has shared principles that give order to all the topics and subtopics to be taught, regardless of whether a course is being held in a high school, a college, or a university. Educators follow a well-described theoretical basis that can be logically mapped and progressively taught from beginning to end, having a clear picture of where the topics fit in the scheme of things. The strong theoretical underpinning in electronics casts light on the entire matter and minimizes pedagogic challenges.

By way of illustration, the concept map of electronics (see Fig. 1) links the various topics both to one another and to the closely related disciplines of chemistry and physics. This concept map is directly inspired by the theoretical achievements of electronics—for example, Ampère's circuital law, Coulomb's electrostatic law, the Lorentz force in electromagnetic fields, Joule's laws, Kirchhoff's circuit equations, Maxwell's equations of classical electromagnetism, Ohm's equation for conductors, and Thévenin's theorem for resistive circuits—which have the property of being connected to one another with precision.

The same cannot be said for CS. Published literature shows an assortment of concept maps of informatics used for educational purposes—for instance, [35]–[37]—but these do not derive from the CS theories strictly speaking since the theoretical achievements underpinning CS do not offer a consistent support. Mathematics has infiltrated computing as extensively as some people consider it to be a branch of CS, but CS has no unified theoretical scheme. In other words, several theories certainly underpin informatics, but this wealth of theories does not make a coherent knowledge base. To discuss this aspect of informatics, some theoretical frameworks regarded as relevant references in the literature are discussed here.

Relational algebra was proposed by Codd as a basis for database query languages [39].

Category theory formalizes mathematical structures in terms of collections of objects and arrows, also called *morphisms.* Categories are currently used to study data types and their properties [40].

An *information theory* was developed by Shannon to calculate the limits on reliably communicating data [41]. Further applications of this theoretical work include—among others—lossless and lossy data compression. In addition, many other information theories have been devised, as will be discussed in the following.

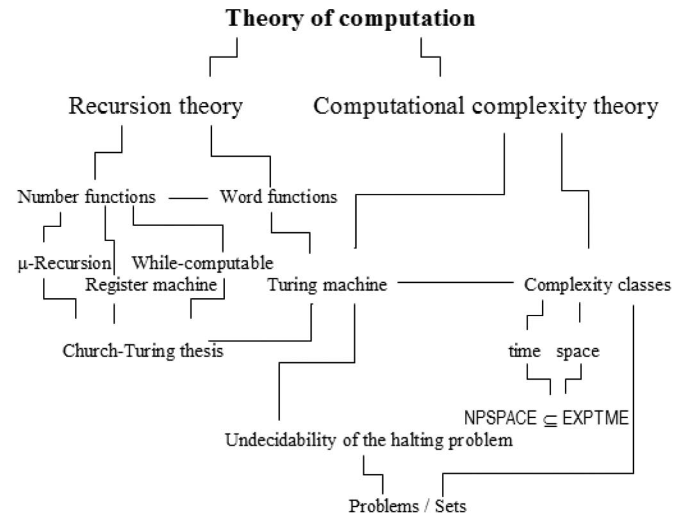

Fig. 2. Concept map of the theory of computation (partial copy from [38]).

Lambda calculus or *λ-calculus*, which was first formulated by Alonzo Church, is a formal system in mathematical logic for expressing computation based on functions using variable binding and substitution [42].

Theory of computation deals with how efficiently a problem can be solved using an algorithm [43]. This construction includes the *computability theory*, also called *recursion theory*. Recursive methods of computation play the major role in this field, as well as explore the possibility of calculating a complicated function by means of a mechanical process [44]. *Computational complexity theory*, which is associated with this framework, focuses on classifying computational problems according to their inherent difficulty and then relating these classifications [45]. EXPTIME and NPSPACE are examples of significantly complex classes (see Fig. 2).

Propositional calculus, also called *propositional logic* or *sentential calculus*, deals with the study of compound propositions formed with the use of logical connectives such as "and," "or," and "not" [46].

Combinatorics is a field of mathematics that studies countable discrete structures [47]. It includes several subfields such as *design theory, partition theory, order theory*, and *graph theory*.

Semiotics treats signs and their meanings [48] and is often divided into three branches: *semantics*, which studies the relations between signs and the things to which they refer; *syntactics*, which inquires the relationships between signs in formal structures; and *pragmatics*, which studies the relations between signs and sign-using agents or interpreters.

The laws of software evolution, which have been formulated by Lehman and Belady starting in 1974 [49], describe a balance between forces that drive changes in programs, as well as forces that slow down software evolution.

Cryptography is heavily based on mathematical computations and schemes that demonstrate if and when a secret code can be broken [50].

The reader can note how these constructions focus on specialized arguments that appear as narrow questions with respect to the broad scenario addressed by the authors of curricula. Each framework deals with a limited theme, whereas informatics education experts take a 360° view of the domain. Computing theories do not offer a real epistemological contribution because of their partial range of vision. Each construction was designed with a different goal in mind and does not contribute to establishing the globally consistent frame needed. Some sections of the listed theories may be logically connected, for example, one can relate the subtopics within the theory of computation (see Fig. 2), but this arrangement might merely help specialist education programs.

The partial results accomplished by Ohm, Lorentz, Kirchhoff, Maxwell, and others have the virtue of offsetting mutual deficiencies. The various equations of electronics are similar to the pieces of a jigsaw. As an image can be achieved by assembling the jigsaw, so scientists can achieve an understanding of electronics. The partial theories of CS, however, do not offer a similar benefit; they do not—nor can they—provide the needed assistance to the educators. This is not simply a question of shared concepts or engendering a spirit of cooperation; significant incongruities emerge in the published works.

Theories that treat the same topic may not be connected either logically or causally, or by shared characteristics. As a case in point, the theory of computation gives support to imperative programming [51]; λ-calculus is able to explain functional programming [52]; relational algebra sustains database languages, and a special model formalizes object-oriented programming. All these theoretical frameworks refer to programming, but do not make a consistent corpus of equations and explanations.

Two or more theories may deal with a single topic, but pursue different targets. For instance, over 30 theories of information have been put forward to illustrate the prismatic nature of information [53]. These diverge in many respects, such as the approach they follow, the starting points of the enquiries, the purposes of the authors, or the relation with technology; a shared criterion for selecting them has not been established, and the concept of information still remains puzzling.

Several researchers develop very abstract frameworks; that is to say, the intended tenets have a weak relationship with reality and appear somewhat unmanageable to practitioners. For example, the following abstract models of computation coexist with the *Turing machine*: the *μ-recursive functions*, *Markov algorithms*, the *register machine*, and the *P'' machine* [54]. There are evident links between combinatorics and digital techniques—for example, cryptography and Shannon's theory—but a rigorous justification of digital and analog signals is missing [53].

Some theoretical frameworks contradict each other. For Turing, the program is the solution to a mathematical problem. By definition, a correct mathematical solution is eternal, and thus, a software program should never change. In contrast, Lehman addresses the problem of software evolution and fixes the laws that regulate this phenomenon. Semioticians investigate the meanings of signs in a systematic manner, but Shannon openly rejects semantics and claims it is "*irrelevant to the engineering problem*" [41].

Some CS theories seem intriguing as they cross various disciplines. For example, λ-calculus has applications in mathematics, philosophy, and linguistics, as well as in computing, but these do not contribute to setting up the exhaustive knowledge framework of informatics.

In conclusion, neither a single theory nor the ensemble of theories underpinning computing leads to the comprehensive framework that should help educators to prepare curricula and resolve the concerns of those who have examined computing theories since the inception of the field. Writers soon became aware that theoretical inquiries constitute the weak side of informatics [55]. Hartmanis made some remarks about the nature of theoretical CS and the research conducted in this area [56]. Demeyer observed how computing is a field deeply rooted in mathematics; this leads to abstract theories, but the field's extensive involvement engineering results in a dualism that tends to compound the problem of CS identity [57]. Tedre pointed out that, despite the short history of computing, there is a great variety of different approaches, definitions, and outlooks on computing as a discipline; he developed an insight into the scientific status of CS after the contribution of various thinkers [58]. Other scholars quarrel about computing theories with a special feeling for educational needs. Barnes *et al.* argued about the interplay between CS constructions and the CS curricula [35]. Hoare guessed that the cultures of science and engineering, which dominate the digital domain, should cooperate toward a common target, such as a grand unified theory of programming [59]. Denning, who is an active advocate of computing as a domain of science on par with the traditional physical and social sciences, has recently relaunched the idea of unifying CS under the umbrella of general principles [60].

Other educators decide to follow the guidance of modern theories on computing and conclude with some critical remarks upon those experiences [61], [62].

The shortage of suitable theoretical support has nontrivial consequences; the following paragraph comments on the work of educators who prepared computing curricula and wrestled with demanding content issues. They achieved a noteworthy job in several stages, but the results do not appear to be fully satisfactory. Van Veen *et al.* and Atlee *et al.* argued on the effectiveness of the curricula lately delivered [63], [64]. The American National Science Foundation recently notes: "*Despite the deep and pervasive impact of computing and the creative efforts of individuals in academic institutions, undergraduate computing education today often looks much as it did several decades ago*" [65].

In response to this criticism, various researchers try to untangle the content issues of informatics, to improve the knowledge organization of this discipline and facilitate the preparation of curricula [66]. Others clarify the representation of the topic areas pertaining to CS curricula and their interrelationships [67]. Some scholars have observed how the use of the same language in different topic contexts compounds the content issues. The Computing Ontology Project has been sponsored by the ACM to establish a common language for educational cooperation across the breadth and depth of computing-related domains [68].

The conventional framework for the CS body of knowledge has been updated in ACM/IEEE Curriculum 2013 [34]; this includes 18 modules that are all key computing technologies. In other words, the lack of shared fundamentals yields curricula strongly orientated to practical services, such as coding a program, using spreadsheets, or emailing a message. It is natural for teachers—lacking a robust theoretical support—to commit themselves to "service-orientated activity." Technical themes guide the lessons, even in introductory courses [69], [70]. It is evident how this didactical style seriously circumscribes the depth of the discourses on computing. Denning complained that computing is still widely perceived as the typical programmer's field [71]. Ben-Ari argued that the common difficulties experienced by novices could be explained by CSE being heavily weighted on the side of bricolage and a too early use of the computer [72]. Finally, a teaching process, even when optimized and updated, turns out to be provisional if it is based on specialist subjects because specialist solutions change with time. The selection of topics and the preparation of a curriculum become particularly stressful when educators are required to update a significant lesson. An example of this would be the dispute about the substitution of the language Pascal with Java or C++. The *objects-first vs. procedures-first* debate has lasted over a decade, and still renders little guidance on what would be an appropriate way to introduce programming [73].

IV. Conclusion

This paper does not intend to establish the superiority of one discipline over another, nor to develop an abstract discussion around the common myth expressed in Statement 1. Rather, the author means to show how some CSE experts, influenced by this belief, meet various difficulties. In particular, scholars attack the demanding content issues of curricula without sufficient theoretical assistance and overlook the role of the underpinning principles of informatics. In practice, they have a mass of concepts, but lack an effective network of logical connections to order them. The CS educators who deal with the breadth and depth of computing must undertake ponderous additional work that the colleagues of other rapid evolving sectors omit or are far less involved in.

This paper has been written with the goal of helping to mature a realistic vision on the part of the scientific community. The author's goal was to illustrate the impact of the theories on CSE researchers' output and to show how inexact ideas lead scholars to lose energies and to double efforts beyond an individual's will and skill. Finally, the author of this paper expects that a research project will be undertaken to carry out a comprehensive study of computing. This innovative initiative would benefit several scientific sectors.

Paolo Rocchi
IBM
00144 Rome, Italy

Research Center on Information Systems (CeRSI)
LUISS Guido Carli University
00197 Rome, Italy
(e-mail: procchi@luiss.it)

References

[1] N. Wirth, "Computing science education: The road not taken," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ.*, Aarhus, Denmark, 2002, pp. 1–3.

[2] C. Yehezkel and B. Haberman, "Bridging the gap between school computing and the 'real world'," in *Proc. Int. Conf. Inf. Secondary Schools, Evol. Perspectives*, Vilnius, Lithuania, 2006, pp. 38–47.

[3] T. Clear, "Software engineering and the academy: Uncomfortable bedfellows?" *SIGCSE Bull.*, vol. 36, no. 2, pp. 14–15, Jun. 2004.

[4] P. Runeson, "A new software engineering program-structure and initial experiences," in *Proc. 13th Conf. Softw. Eng. Educ. Training*, Austin, TX, USA, 2000, pp. 223–232.

[5] M. Shaw, J. Herbsleb, I. Ozkaya, and D. Root, "Deciding what to design: Closing a gap in software engineering education," in *Proc. Int. Softw. Eng. Educ. Modern Age*, St. Louis, MO, USA, 2005, pp. 28–58.

[6] P. Waychal, "The calling of the third dimension," in *Proc. 7th Int. Workshop Coop. Human Aspects Softw. Eng.*, 2014, pp. 123–126.

[7] B. Stroustrup, "What should we teach new software developers? Why?" *Commun. ACM*, vol. 53, no. 1, pp. 40–42, Jan. 2010.

[8] M. Guzdial, "What's the best way to teach computer science to beginners?" *Commun. ACM*, vol. 58, no. 2, pp. 12–13, Feb. 2015.

[9] I. Miliszewska and G. Tan, "Befriending computer programming: A proposed approach to teaching introductory programming," in *Proc. Inf. Sci. Inf. Technol.*, vol. 4, 2007, pp. 277–289.

[10] A. E. Tew, W. M. McCracken, and M. Guzdial, "Impact of alternative introductory courses on programming concept understanding," in *Proc. 1st Int. Workshop Comput. Educ. Res.*, Seattle, WA, USA, 2005, pp. 25–35.

[11] A. Pears *et al.*, "A survey of literature on the teaching of introductory programming," *ACM SIGCSE Bull.*, vol. 39, no. 4, pp. 204–223, Dec. 2007.

[12] T. Greening, Ed., *Computer Science Education in the 21st Century*. New York, NY, USA: Springer-Verlag, 2000.

[13] A. T. Chamillard and L. D. Merkle, "Evolution of an introductory computer science course: The long haul," *J. Comput. Sci. Colleges*, vol. 18, no. 1, pp. 144–153, Oct. 2002.

[14] R. Lister, "After the gold rush: Toward sustainable scholarship in computing," in *Proc. 10th Australasian Comput. Educ. Conf.*, Darlinghurst, NSW, Australia, 2008, pp. 3–18.

[15] A. B. Tucker, "Strategic directions in computer science education," *ACM Comput. Surveys*, vol. 28, no. 4, pp. 836–845, Dec. 1996.

[16] J. Liu, "Computing as an evolving discipline: 10 observations," *Computer*, vol. 40, no. 5, pp. 112–111, May 2007.

[17] A. N. Kumar, R. K. Shumba, B. Ramamurthy, and L. D'Antonio, "Emerging areas in computer science education," in *Proc. 36th SIGCSE Tech. Symp. Comput. Sci. Educ.*, St. Louis, MO, USA, 2005, pp. 453–454.

[18] House of Commons, "Harmful content on the Internet and in video games," Culture, Media and Sport Committee, Tenth Report of Session 2007–08, vol. 2, London, U.K.: Blackwell, 2007.

[19] United States Congress House, "Improving management and acquisition of information technology systems in the Department of Defense," Committee on Armed Services, Subcommittee on Emerging Threats and Capabilities, U.S. Government Printing Office, 2011.

[20] F. Clifton, *Inventing the Future: How Sciences and Technology Transform our World*. Westchester, CA, USA: Hughes Aircraft Co., 1993.

[21] D. L. Morton and J. Gabriel, *Electronics: The Life Story of a Technology*. Baltimore, MD, USA: Johns Hopkins Univ. Press, 2007.

[22] D. Crecraft and D. Gorham, *Electronics*. Boca Raton, FL, USA: CRC Press.

[23] L. Zhao and F. Mak, "Building learning bridges among undergraduate courses of electronics, power electronics and electric drives lab," in *Proc. Frontiers Educ. Conf.*, Las Vegas, NV, USA, 2010, pp. F2F.1–F2F.5.

[24] E. A. Mc Shane, M. Trivedi, and K. Shenai, "An improved approach to application-specific power electronics education: Curriculum development," *IEEE Trans. Educ.*, vol. 44, no. 3, pp. 282–288, Aug. 2001.

[25] UNESCO, "UNESCO ICT Competency Framework for Teachers," 2011. [Online]. Available: http://unesdoc.unesco.org/images/0021/002134/213475e.pdf

[26] R. H. Austing, B. H. Barnes, T. Della Bonnette, G. L. Engel, and G. Stokes, "Curriculum'78: Recommendations for the undergraduate program in computer science: A report of the ACM curriculum committee on computer science," *Commun. ACM*, vol. 22, no. 3, pp. 147–166, Mar. 1979.

[27] ACM/IEEE-CS Joint Curriculum Task Force, "Computing Curricula 1991," IEEE Computer Society Publication, 1991.

[28] The Joint Task Force on Computing Curricula ACM-IEEE, "Computing Curricula 2001 Final Report," 2001. [Online]. Available: http://www.computer.org/education/cc2001/

[29] F. Mulder and T. van Weert, "ICF/2000 Informatics Curriculum Framework 2000 for Higher Education," UNESCO/IFIP, 2000.

[30] Career Space, "Curriculum Development Guidelines/New ICT Curricula for the 21st Century: Designing Tomorrow's Education," Office for Official Publications of Computing/ICT, 2001.

[31] The Joint Task Force on Computing Curricula, "Computing Curricula 2005: An Overview Report," 2005. [Online]. Available: http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf

[32] The Joint Task Force on Computing Curricula IEEE-ACM, "Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." Computing Curricula Series, 2004. [Online]. Available: http://www.sites.computer.org/ccse/SE2004Volume.pdf

[33] The ACM/IEEE-CS Task Force, "Computing Curricula 2013," 2013. [Online]. Available: http://www.acm.org/education/CS2013-final-report.pdf.

[34] P. Mathys, "Analog electronic concept map," Univ. of Colorado, 2014. [Online]. Available: http://ecee.colorado.edu/~mathys/ecen1400/ConceptMaps/ElectricCircuits1.html

[35] B. H. Barnes, G. I. Davida, R. A. Demillo, and L. Landweber, "Theory in the computer science and engineering curriculum: Why, what, when, and where," *Computer*, vol. 10, no. 12, pp. 106–108, Dec. 1977.

[36] J. B. Gammack, V. Hobbs, and D. Pigott, *The Book of Informatics*. Melbourne, Vic., Australia: Cengage Learning, 2011.

[37] A. Mühling, P. Hubwieser, and M. Berges, "Dimensions of programming knowledge," in *Proc. 8th Int. Conf. Inf. Schools, Situation, Evol., Perspectives*, Ljubljana, Slovenia, 2015, pp. 32–44.

[38] Wikis, "Computational complexity theory," 2015. [Online]. Available: http://www.thefullwiki.org/Computational_complexity_theory

[39] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[40] M. Bar and C. Wells, *Category Theory for Computing Science*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1995.

[41] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[42] J. R. Hindley and P. S. Jonathan, *Lambda-Calculus and Combinators: An Introduction*, 2nd ed. Cambridge, MA, USA: Cambridge Univ. Press, 2008.

[43] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Boston, MA, USA: Cengage Learning, 2013.

[44] S. B. Cooper, *Computability Theory*. Dordrecht, The Netherlands: Chapman & Hall, 2004.

[45] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[46] A. Prior, *Formal Logic*, 2nd ed. Oxford, U.K.: Oxford Univ. Press, 1990.

[47] M. Bóna, *A Walk Through Combinatorics*, 3rd ed. Singapore: World Scientific, 2011.

[48] U. Eco, *Semiotics and the Philosophy of Language*. Bloomington, IN, USA: Indiana Univ. Press, 1986.

[49] I. Herraiz, D. Rodriguez, G. Robles, and J. M. Gonzalez-Barahona, "The evolution of the laws of software evolution," *ACM Comput. Surveys*, vol. 46, no. 2, pp. 1–28, Nov. 2013.

[50] G. Baumslag, B. Fine, and M. Kreuzer, *A Course in Mathematical Cryptography*. Berlin, Germany: De Gruyter, 2015.

[51] D. A. Z. Zuhud, "From programming sequential machines to parallel smart mobile devices: Bringing back the imperative paradigm to today's perspective," in *Proc. Int. Conf. Inf. Technol. Asia*, 2013, pp. 1–7.

[52] B. J. MacLennan, *Functional Programming: Practice and Theory*. Boston, MA, USA: Addison-Wesley, 1990.

[53] P. Rocchi, *Logic of Analog and Digital Machines*, 2nd ed. Huntington, NY, USA: Nova, 2012.

[54] F. Maribel, *Models of Computation: An Introduction to Computability Theory*. New York, NY, USA: Springer-Verlag, 2009.

[55] R. T. Boute, "On the shortcomings of the axiomatic approach as presently used in computer science," in *Proc. Conf. Des., Concepts, Methods Tools*, 1988, pp. 184–193.

[56] J. Hartmanis, "Observations about the development of theoretical computer science," in *Proc. 20th Annu. Symp. Found. Comput. Sci.*, 1979, pp. 224–233.

[57] S. Demeyer, "Research methods in computer science," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance*, 2011, p. 600.

[58] M. Tedre, "Computing as a science: A survey of competing viewpoints," *Minds Mach.*, no. 21, pp. 361–387, Aug. 2011.

[59] T. Hoare, "Science and engineering: A collusion of cultures," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2007, pp. 2–9.

[60] P. J. Denning and C. H. Martell, *Great Principles of Computing*. Boston, MA, USA: MIT Press, 2015.

[61] V. Y. Shen, S. D. Conte, and H. E. Dunsmore, "Software science revisited: A critical analysis of the theory and its empirical support," *IEEE Trans. Softw. Eng.*, vol. SE-9, no. 2, pp. 155–165, Mar. 1983.

[62] R. L. Constable, "Experience using type theory as a foundation for computer science," in *Proc. 10th Annu. IEEE Symp. Logic Comput. Sci.*, 1995, pp. 266–279.

[63] M. Van Veen, F. Mulder, and K. Lemmen, "What is lacking in curriculum schemes for computing/informatics?" *ACM SIGCSE Bull.*, vol. 36, no. 3, pp. 186–190, Sep. 2004.

[64] J. M. Atlee, R. J. LeBlanc, T. C. Lethbridge, A. Sobel, and J. B. Thompson, "Reflections on software engineering 2004: The ACM/IEEE-CS guidelines for undergraduate programs in software engineering," in *Proc. Int. Conf. Softw. Eng. Educ. Modern Age*, 2004, pp. 11–27.

[65] NSF, "NSF provides funding to transform computing education," Press Release 07-131, Oct. 2007. [Online]. Available: https://www.nsf.gov/news/news_summ.jsp?cntn_id=110158

[66] R. Kamali, L. Cassel, and R. LeBlanc, "Keeping family of computing related disciplines together," in *Proc. 5th Conf. Inf. Technol. Educ.*, 2004, pp. 241–243.

[67] L. N. Cassel *et al.*, "A synthesis of computing concepts," *SIGCSE Bull.*, vol. 37, no. 4, pp. 162–172, Dec. 2005.

[68] L. N. Cassel, G. Davies, R. LeBlanc, L. Snyder, and H. Topi, "Using a computing ontology as a foundation for curriculum development," in *Proc. 6th Int. Workshop Ontol. Semantic Web E-Learning*, 2008, pp. 21–29.

[69] K. P. Bruce, "Controversy on how to teach CS1: A discussion on the SIGCSE-members mailing list," *SIGCSE Bull.*, vol. 36, no. 4, pp. 29–34, Dec. 2005.

[70] M. Piteira and C. Costa, "Computer programming and novice programmers," in *Proc. Workshop Inf. Syst. Des. Commun.*, Lisboa, Portugal, 2012, pp. 51–53.

[71] P. J. Denning, "Great principles in computing curricula," in *Proc. 35th SIGCSE Tech. Symp. Comput. Sci. Educ.*, Norfolk, East Anglia, 2004, pp. 336–341.

[72] M. Ben-Ari, "Constructivism in computer science education," *J. Comput. Math. Sci. Teaching*, vol. 20, no. 1, pp. 45–73, Mar. 2001.

[73] H. Abelson, K. Bruce, A. van Dam, A. Tucker, and P. Wegner, "The first-course conundrum," *Commun. ACM*, vol. 38, no. 6, pp. 116–117, Jun. 1995.

Paolo Rocchi (A'11–M'12) received the degree in physics from the University of Rome, Rome, Italy, in 1969.

He has been an Assistant Lecturer with the Institute of Physics, University of Rome. In 1970, he joined IBM, Rome, as a Docent and a Researcher. Upon retirement in 2010, he was recognized as an Emeritus Docent at IBM for his achievements in basic and applied research. He is also currently an Adjunct Professor with LUISS Guido Carli University, Rome. He has written over 120 works, including a dozen books. He has carried out research, and is still active, in various fields of computing, including software evolution, computer security, education, information theory, fundamentals of computer science, and software engineering. In addition, he has conducted inquiries into probability theory, reliability theory, and theoretical biology.

Prof. Rocchi is a member of various scientific societies and has received recognition even beyond the scientific community, in the mass media.