Facing computer science misconceptions: an introductory course based on historical strands and career paths at a glance

Siegfried Rouvrais, Ioannis Kanellos

Facing Computer Science Misconceptions: An Introductory Course Based on Historical Strands and Career Paths at a Glance

Siegfried Rouvrais and Ioannis Kanellos
Telecom Bretagne, Institut Telecom, Université européenne de Bretagne, Brest, France
firstname.lastname@telecom-bretagne.eu

Abstract - **Incoming students in post-secondary education often perceive Computer Science (CS) under a series of stereotypes. While it is certainly difficult to define CS, it seems still possible to grasp its scope through comprehensive points of view, addressing for example its foundations, technologies, and uses. This paper proposes a retrospective analysis of a 21-hour course trial realized in a French engineering school for six consecutive years; its aim was to help recent K12 graded students to have a more objective vision of CS. We start by discussing freshmen's CS dominant conceptions, and the role they think it plays for their future career. We then give some statistical elements concerning student perception of such a course; in particular, we analyze the obtained feedbacks and the limits of intended learning outcomes. In the light of this analysis, it is possible to state that, although initially reluctant to CS due to misconceptions, some students can develop interest and expectancy for this discipline as applied to engineering. A clarified vision of CS seems moreover to contribute to better adjusting choices of minors and majors and facilitate reasoned career plans. We finally conclude on some possible future issues in this subject for our society.**

Index Terms – Computer Science misconceptions, digital native freshmen, engineering education, undergraduate introductory Computer Science course.

INTRODUCTION

Admittedly, Computer Science (CS) has worldwide unequal perceptions in different educational communities. K12 graded (or postsecondary) students are not uniformly conscious of CS essentials, even if they are passionate young folks. The reasons are multiple and rather complex and it would be rash to try to isolate one sole explication principle. They generally address intrinsic disciplinary traits, long-term educational traditions or socio-economical and cultural determinations (stereotyped practices, market orientations, media representations of what CS is or could be, trivialization of the computers through various technologies and uses in every day's life…). Pressures exerted by other disciplines seem also significant in such understandings (CS appears as the sum of some techniques for efficient calculus, as an "assistant discipline" for problem complexity reduction, as instrumentation for time and space constraints governance, as precision means, etc.). The crucial questions about the nature of computing, its limits and its real promises are rarely envisaged. "What, finally, makes all this possible"? "What is the very specificity (and power) of calculus enabling all such solutions?" "How machines realize calculi in a reliable manner giving rise to new technological and scientific paradigms and uses?" "In which sense and manner calculating machines modify our societies and our lives?" are generally questions for philosophers.

A universally accepted definition of CS is clearly impossible; or it rapidly turns into a position, transforming the initial definitional or descriptive intention into a normative attitude: CS ceases to appear as a class of theories or attested practices and becomes what "it owes to be". Normative attitude is clearly restrictive; it sometimes holds hegemony desires. Curiously, all this definition debate about CS seems to inherit the old controversy concerning the correct epistemological attitude in studying the natural language: in student's mind, what matters is its synchronicity, not its diachronic aspects. Thus, the historical view becomes often irrelevant or undervalued as far as new devices and computer techniques minimize the importance of the principles as well as the theoretical and practical potentialities.

When teaching CS, we are nowadays in front of "Millennial classes" [1] whose socio-educative features impart radical shifts to their learning expectations. Indeed, our students are digital technologies and media natives and their lives are globally peer-oriented (they all have a computer and a cell phone); they are familiar with social networks and feel at ease with instant and/or multitask activities. If they appear well adapted to information society trends and developments, they still belong to a generation of worrying rates of youth unemployment (often over 20%). Many studies point out their penchant for delaying passage into adulthood ("Peter Pan generation") and their constant state of anxiety: conscious that they come after successive recessions and that they will be obliged to switch jobs frequently, they seem to single out generic and reusable skills than foundational knowledge. In anxiety times, action prevails over understanding and performance over competence. The conditions are not optimal for liberal arts! CS is, at most, a set of techniques over what they already know by using digital technologies.

As such, in various national educational contexts worldwide, K12 graded students are not systematically attentive of CS fundamentals. When entering an engineering school (e.g. STEM), incoming students have not so often a professional ideal; uncertainty and indecision dominate in their professional future appraisal [2]. Their judgment of a CS topic is thus affected. On the other hand, they must rapidly choose minors and majors in postsecondary education, even if most of them have a very limited vision of the working world and career kaleidoscope. Some students struggle to identify career directions and need more time before feeling committed and being operational within their curriculum. Thus, as CS has become inescapable in our information societies, it seems critical to help to overpass received ideas. Give an as truthful as possible image of the CS spectrum to incoming students in higher education becomes a challenge. Such an educational aim joins a cultural one:

- To somehow release our students from resistant stereotypes and misconceptions (for instance, that CS amounts to mere office tools or to obscure source code programming);
- To provide means enabling them to participate actively in their own learning path;
- To contribute to the clarification of their future professional identity [3], beyond common rigid CS values and beliefs.

EDUCATIONAL FRAMEWORKS, CS CURRICULUM GUIDELINES, AND COURSE EXAMPLES

From a global perspective, educational frameworks are defined to facilitate program design and interoperability. For example, the international CDIO initiative for engineering [4], launched in 2002, proposes a framework for the next generation engineers. Its 12 standards are already adopted by several schools of engineering and universities throughout the world. In its syllabus (i.e. learning outcomes of this 2nd standard), the first category "disciplinary knowledge and reasoning" is subdivided into (i) knowledge of underlying mathematics and sciences (i.e. Physics, Chemistry and Biology), (ii) core engineering fundamental knowledge and (iii) advanced engineering fundamental knowledge, methods and tools. In compliance with the engineering practices, this syllabus is particularly focused on how to conceive, to design, to operate and to implement. Astonishingly, there are no recommendations concerning CS in the disciplinary knowledge of this syllabus; we only find, in the implementation category, "software implementation process and hardware software integration".

On the other hand, existing curriculum guidelines provide indications for courses concerning program design. Some of them (e.g. SWEBOK or CCSE-SEEK), are dedicated to CS and provide identified knowledge areas. Approved by the ACM and the IEEE, large and precise CS curricula are also specified [5]. As for ABET-CSAB, on the Continent, the European Quality Assurance Network for Informatics Education (EQANIE) works on quality assessment practices for CS education; its basis for accreditation, in conjunction with the European guidelines, is the Euro-Inf Framework Standards and Accreditation Criteria [6]. The first category in this syllabus is the "underlying conceptual basis for Informatics", defined as essential in order to satisfy the whole learning outcomes of the category. For instance, at the end of the first cycle, the students should have demonstrated:

- Knowledge and understanding of the key aspects and concepts of CS, including some at the forefront of that discipline;
- Awareness of a wide spectrum of CS disciplines.

For the second cycle, they should have demonstrated:

- Profound knowledge and understanding of the principles of CS;
- Either deep knowledge of a chosen specialization or broad knowledge of CS in general;
- Critical awareness on the forefront of their specialization.

In such initiative, critical attitude seems essential. But, in designing introductory courses in CS for undergraduate programs, it is crucial to take into account the incoming student background, their motivational issues, as well as theirs beliefs and values regarding CS; at least for the first cycle. In 2007, reference [7] surveyed CS introductory curricula in the U.S. Lot of units are effectively addressing computer programming concepts (using Java, for example). Some courses are also addressing other aspects. For example, Georgia Tech College of Computing proposes: "CS 1100: Freshman Leap includes seminars for incoming CS students that introduces foundational, motivational and topical subjects". Its aim is to expose students to a broad survey and explore the dedicated degree. CMU School of Computer Science proposes the "15-128: Freshman Immigration Course to acquaint incoming students with computer science". It consists of talks, ranging from historical perspectives to descriptions of current CS research programs.

As CS is the discipline on which modern life is built, we also find educational initiatives proposing consciousness-raising campaigns and recommendations for primary or secondary school CS introduction (e.g. [8]). In our specific context, both curriculum issues and the image of the CS discipline are considered, at two levels: European (e.g. the Informatics Europe Association) and French national (e.g. the French Society for Education and Research in Computer Science (SPECIF)). They all converge towards a coordinated introduction of CS essentials throughout secondary schools.

AN INTRODUCTORY COURSE ON COMPUTER SCIENCE

I. Context

Participating to the CDIO international initiative [4] since 2008, our institution, Telecom Bretagne, is an engineering school, with an affirmed focus on ICT. Since 1977, it is an integral part of the French *Grande École* system [9] and

regularly certified by national and European accreditation boards. As a generalist school, many professional sectors and domains are accessible to its graduates. In its classical curriculum, it admits two types of freshmen: (i) foreign students, often with multicultural expectations and whose decision to study in France was generally carefully pondered (in 2009, c. 10% of the 160 incoming students) and (ii) French and North African students who attended French preparatory classes, having in mind to enter into a *Grande École* after a highly competitive exam [10].

In 2003, our school reorganized its ICT engineering education program by incorporating large project-based learning experiences in its curricula [11]. There, a first semester core curriculum was mandatory for all students. It consisted of an integrative design project including several disciplines (approx. 100 hours per student, grouped by teams) and respectively 42 hours introductory blocks in five distinct disciplines: (i) CS, (ii) Economy and Humanities, (iii) Mathematics and Signal Processing, (iv) Electronics, and (v) Networks. At the second semester, students have to choose minors and majors, as well as compulsory courses. In this reform, the introduction to CS included (i) a 21-hours introduction to imperative programming (based on Java, without object-oriented concepts) and (ii) the course "A glance at Computer Science", analyzed in this paper, with a similar time frame. Both courses were interleaved in the agenda. Two years later, a 9-hours course was added, introducing to basic Unix/Linux commands and text processing good practices (based on OpenOffice and LaTEX).

II. Views and Historical Strands in the Course "A glance at Computer Science"

For new millennium learners, personal and social experience of digital technologies tends to circumscribe CS to acquired consumption categories. CS appears fractional, biased and limited to every day's life utility. Clearly, CS past becomes irrelevant, its real potential concerns only possible quantitative improvements and its future is fantasy and fiction. The focal argument of the compulsory 21-hour semester course we set up was to combine a horizontal extensive presentation of CS through three different points of view (covering three generic fields, i.e. foundations, technologies and uses), with a vertical historical approach. The horizontal and vertical presentation aimed at unifying CS essentials (the how, the why) in a dynamic perspective: past, present and future. In fact, our students do not always distinguish what is essential and what is accessory in CS, what is permanent and will always be the same and what is variable (and at which rate of variance), what is possible and what will be, anyway, impossible to do with calculus. For instance, they generally think that we can do almost everything with computers, i.e. that there are no limits for calculus and that increasing computing performances will be enough to deal in a near future with all problems; that the architecture of computers is not important and that Moore's law will be valid indefinitely. Moreover, they are not aware if the solution they propose engages NP algorithmic complexity difficulties, or if the targeted uses they imagine have social limits. In any case, they have a very limited vision of the difference between human cognition and the functioning of computers, and they do not imagine man/machine cooperation as a part of CS that allows extending the classical notion of computing.

The historical preoccupation was not the trait of an obsolete educational option: it was rather a way of shaping conscience and therefore scientific identity. It was, further-more, the vector for a holistic educational schema restituting CS through memory and imagination. We tried to give our students elements to (i) identify past achievements and cur-rent progress concerning some relevant sub-fields of CS, (ii) discover the most important principles underlying the dis-cipline, (iii) create an initial cultural background to better understand the discipline potential and limits and (iv) via a group session, survey some typical CS careers. It was de-composed in three parts:

1. The purpose of the "foundations point of view" was to demonstrate that CS relies on rigorous scientific grounds, and has unshakable and time invariant mathematical, logical and formal relevance; and that there is an important distinction to do between effective and theoretical calculus (complexity theory).
2. The "technologies point of view" aimed at explaining the recurring purpose of creating patterns applicable to numerous human ordinary problems, through calculations made by machines. This part, both hardware and software, is technology dependent and generally subject to important changes in time, as technology evolves.
3. The "uses point of view" attempted to illustrate the various interaction modes between humans and machines. More precisely, to furnish an accurate image the way computers may model and become, furthermore, the most adequate instrument for understanding situations and interacting with environments. We moreover tried to give an idea of the mutual reliance between traditional practices and CS; and, retrospectively, to reveal the broad scope, both theoretical and practical, of the latter.

The ambition of this course was, also, to put an emphasis on the interdependence of these three points of view by combining discovery, practice, reflection and criticism, applied to chosen aspects. Clearly, there were no theoretical or technical pre-requisites for this course. In accordance to the hierarchical cognitive process dimensions of the Bloom revised taxonomy [12], the intended learning outcomes concerned, essentially, factual and conceptual knowledge. Among other elements, students should be able to [13]:

- Distinguish various perspectives of CS, and classify a problem (as theoretical, technological, programming, scientific, or end-user problem);
- Review the important CS foundations (in particular, the notions of calculus and complexity);
- Support with arguments significant evolutions of CS (especially, by pointing out the relationship between

technological and socio-economical level) and recognize their impacts on other fields and disciplines;
- Imagine, for some topics gleaned throughout these points of view, what can be made (in particular, be aware of crucial directions in CS R&D);
- Map the school curriculum, including minors and majors and electives, regarding CS and related perspectives;
- Differentiate learning path in CS, express values of CS in engineering activities and professions, and locate their potential future professional identity.

The assessment comprised a written individual valuation (80% of the account) at the end of the course; in addition, four written deliverables (continuous assessment) were furnished by collaborative groups of 4 students (20%).

III. Course Structure

The course "A glance at CS" offered several 3 hour-sessions, including lectures (full promotion), collective workshop classes (classically, 3 groups of 8 students per classes), as well as binomial practices in laboratories (6 or 7 binomial groups). It was designed as follows:

1. Opening phase (short lecture): presentation of the CS as a point of view system, review of the philosophy and the learning outcomes of the course, clarification of the axes and the assessment methods;
2. CS theory phase (lecture and workshops): presentation of different points of view of the notion of calculus (Turing machine, logical, formal, algorithmic and recursive approach), relationships between the presented approaches, limits of calculus (Halting Problem and Gödel's incompleteness theorems), introduction to complexity theory (fundamental problems, P and NP classes…);
3. Computer architecture (lecture and workshops): survey of early computers and historical overview until today following a pure technological axe, definition of the von Neumann design model for digital computers, explications about the role played by the microprocessor, discussions on Moore's law…
4. Programming paradigms (lab and restructuring lecture): demonstration that abstraction and efficiency are recurring notions in tools to produce automatic computation, following a user-programmer axis, evidence of the reasons underlying distinct programming paradigms and so many programming languages (especially, as opposed to one single universal language), examples on the opposition between interpretation and compilation, synthesis of the history of programming languages…
5. Human-machine interfaces (lab): exercises on the importance of HMIs, window-based graphical user interface (e.g. from Xerox Star, SketchPad, to actual WYSIWIG interfaces), on folders, icons and mouse, presentation of some HMI methods (design, ergonomic criteria…) and tools…
6. Surveying career directions (workshop): following a problem-based learning style [14], this group session required the presence of a tutor whose task was to prompt student teams to list and understand, without external assistance, career opportunities (definition of a group of competencies associated with their perception of the engineer profession). Based on highlights, a common short definition of a generalist engineer was collaboratively established and compared to recognized definitions leading, in fine, to an exhaustive list of typical job profiles (using offers from job boards). Job categories were then defined and, for some of these jobs, the student teams were asked to describe some concrete work situations and set up a list of ten pondered core competencies. Finally, each team presented its conclusions to the others on a slide.
7. Closing phase (short lecture): it consisted of a retrospective presentation of the real limits of computing, technologies and uses, with an overview on the minor and major courses in the CS or Computer Engineering fields, offered in the institution.

Various reading assignments and media, not directly related to the assessment, were also proposed along the course to students.

RESULTS

With six year experience that involved about 1000 students (about 86% of males, around 22 years old), we had at disposal sufficient quantitative and qualitative material to evaluate this course. Feedback results were regularly collected in order to analyze activity effectiveness. Such evaluation was also vital for managing continual improvement. Indeed, at the end of the career direction workshop (phase 6, above), teams were prompted with a satisfaction questionnaire. Averaged results per group were established, due to necessary consensus for replies. Peer exchanges and debates on students viewpoints allowed us better examine the place of the CS in an engineering curriculum. Students were able to add comments to the binary questions. Some of these comments explicitly revealed profound disagreement between teams. Hereafter, the transcription of some interesting results:

- About 90% of the teams replied positively to the question: "In your opinion, is it relevant to approach CS through different points of view?". Systematically, students recognized open-minded issues and that the course permitted to distinguish the diversity of the discipline and clearly pointed out that CS is not just a question of programming. Some teams noticed however that the course should be more concrete. Only once, a team stipulated negatively that "CS remains CS, whatever is the view".
- Approximately 85% of the teams replied negatively to the question: "Do you feel that reviewing CS following an historical approach is waste?". Several comments recognized benefits of such a retrospective choice; but

others stipulated that history is often redundant and, even if CS is a relatively recent discipline, it evolves very rapidly, new states of the discipline bringing a clear break with the past. A team declared that, for invention, it is necessary to have some knowledge of the past. Another stated that it is moreover important since most of the systems operate using old principles. Some responses noticed that the historical choice was unnecessary for the theoretical axe. A team commented that students do not have enough hindsight in such a few timeframe. Another found it, straight out, boring. Some teams were rather reserved: "Despite history, what are the interests of the survey course?" or "Is general culture on CS necessary?" or even "To which extend will it be useful in the rest of the curriculum and for our engineering career?".

- Approximately 75% of the teams replied positively to the question: "Did you clearly identify the views presented in this course?". Some teams recognized that the questions of the continuous assessment were beneficial for such a purpose. However, some of the negative replying teams noticed that distinctions of the frontiers between views were not so apparent, that the selected views were sometimes superfluous.
- About 65% replied positively to the question: "Thanks to this course, are you able to consider if you are more interested on a view?". Usability (as programmer or simple user) was often rated; theoretical approach was rather depreciated. Approximately 15% were not ready to take position and the remaining part replied negatively. Some teams argued that the course helped them to identify there preferences and decide between CS and other disciplines for the near future. Some other felt it too theoretic. A 'no' answering group recognized that it was nevertheless helpful to distinguish focus areas in CS. Another group noted that the course should be optional and reserved to interested students.

Moreover, at the end of the first semester and after semester exams, each student replied individually (and anonymously) to a questionnaire relative to the CS initial core curriculum. Thanks to the various feedbacks, the course was evaluated both locally and within the context of the introduction to imperative programming.

- The vast majority of students reported that the two interleaved courses ("Imperative Programming" and "A glance at CS") were seen as quite independent. Some of them would prefer less allocated lessons (7h30 of lectures in the 21h timeframe) or additional exercises and workshops.
- 96% of the students found that those two core courses were useful for the rest of their curriculum. Some even asked to start by the C programming language rather than the Java one.
- 73% expressed that their initial knowledge was sufficient to follow the two courses. However, some of them pointed out that the courses should be adapted to levels, depending on whether they have taken an optional course on programming in the previous preparatory classes (indeed, less than a quarter of incoming students had been taught functional programming using CAML).
- 85% agreed that the learning objectives specified were corresponding to the specifications given in the course material booklets. However, for this course, several students noticed that it was hard to perceive what was finally intended. Noteworthy, one student emitted: "What is a CS exam in which it is required only text as answer to the questions? This is not serious!".

EVALUATION AND DISCUSSION

In the light of the previous feedbacks, it is clear that the overall students' satisfaction was not the main strength of this compulsory first semester course. However, results and comments permitted to state that, though initially reluctant to CS, essentially due to misconceptions and resistant usefulness criteria, some students can develop interest in CS; at least, they can better understand the relationship between CS and traditional engineering practices and improve their choices of minors and majors.

Even if slightly adjusted over the years, many students seem not clearly identify the original intentions of the course. This last was especially clear regarding the competency pondered listings of job examples exposed by student teams in the career workshop. From a practical standpoint, it highlighted that, due to rather poor experience and knowledge in CS related careers, many students founded it hard to propose a list of jobs, within a so short time, without any reference to a set of documents exemplifying requested competencies for typical work situations (e.g. offers in job boards). Consequently, the activity (phase 6) was often lowly rated by freshmen, even if they recognized that it was a precious seed to initiate awareness and perception requirements in careers. Such experience was too isolated in the first year to provide students genuine tools to analyze and evaluate, from a personal standpoint, the various careers available for them.

Recent statistics showed that almost 1/3 of our graded students work in some domain of CS. On the basis of a series of complementary feedbacks of students and instructors, gleaned after the 6-years experiment with this course, we concluded that such a career approach of CS seems more valuable for (and appreciated by) students after an internship or a professional experience than by inexperienced freshmen (see reference [3] for a detail analysis of this study).

Even more interesting: scores corresponding to the student evaluations relative to the course values reveal a clear bimodal distribution (i.e. they like it or not, rarely "sit on the fence"), corroborated by the student peer exchanges and debates. In our corpus, made up from students' responses at the end of the course, the noticed bimodal value seems radical: the enthusiasm goes together with a strong disapproval of the course. Curiously, we observed the same radicalization by collecting the opinion of our colleagues. Even if the reasons are clearly different, the origins are, in all likelihood,

an evidence feeling founded on personal experience, social perception and values. Moreover, this course was often criticized both by the program managers and the faculty staff (in terms of utility, adequacy to market exigencies, and students' satisfaction). The argument of lack of time was also dominant (our students evolve in temporalities that do not allow the culture luxury: know-how had to be generally privileged). Thus, practical and time constraints, as well as vocal individuals, pressure steadily CS curriculum change [15].

On the other hand, it appears tricky to evaluate the impact of systemic effects (for instance, the tiny 21-hours constraint, the parallelization of this course with a mere practical one, the penury of common paradigms in all exposed points of view that could strengthen the epistemological value of the lectures, furnishing weight and evidence to the whole approach…). It is clearly difficult -if not quite impossible- to assess culture and critical maturity. The remark is broad: courses on scientific culture appear nowadays as the indigent parent in technological curricula; they are not systematically supported by institutions and, anyway, their value, as a prerequisite for knowledge or operational skills for future courses, is generally suspected.

Contrarily to other disciplines, CS has the disadvantage of its own advantage: its diffuse and permanent embedding in real life. As such, it gives rise to standardized arrogant learning positions. It is generally possible to modify an opinion; but very hard to change an attitude. Today's CS introductory courses have to reconsider long-established pedagogical schemata as far as digital practices offer alternative modes of knowledge downgrading traditional school ones. In evaluating such courses, teachers owe to demonstrate prudence. Students' satisfaction is not a norm, nor a principle, and may even be dangerous (transforming the learning process to a consumption object, confining students to a closed "client-desire-object" system). Market tensions and immediate usefulness have also to be well balanced, insofar as they may confiscate the educational codes and values in force, subjugating education to immediate profitability.

CONCLUSION

In this paper we tried to show, as honestly as possible, a beautiful (counter)example resuming the difficulty of setting up a reliable introductory course in CS for postsecondary education. In our French national context, and its *Grande École* system, the position of CS is still controversial [16]. "What to teach scientists and engineers about computing?" is a question that stimulates fascinating (and even sometimes passionate) debates [17]. Through this paper, we sought to show that a course is not an isolated learning unit but rather something that participates to an educational ecology whose limits are intertwined, interdependent, indiscernible and usually unpredictable.

If to act is inescapable, to understand is not irrelevant: theory and practice are of course inseparable. CS is not the Fourth World of scientific rigor; not some new devise for surfing, networking, buying or playing. The information society arrival shows that the notion of information and, more generally, the notion of calculus, are not the characteristics of a new lifestyle, but the essential component in merging technology and social being in modern lives. The diversity of CS goes, therefore, far than concepts and techniques, as far as the humanity is invoked. It will be always a risk to deprive CS of critical thought. Consequently, it will be always a challenge for us, teachers and educators, to find the correct level of CS presentation to our students; to find solutions allowing them to escape from the comfortable but impoverishing position of an end user. Certainly, it will not be through theoretical lectures; certainly, it will not be by presenting CS as a discipline assimilated to technology; and a CS reduced to usages seems destined for failure as well.

The question is fundamental and clearly general; it concerns motivation to learn topics that are not directly, evidently and naively attracted by usefulness. The lack of motivation of our digital natives has already pointed out, together with its annoying consequences: difficulty in setting learning goals, inefficient cognitive strategies, weak effort and deficiency of persistence. Our educational mission is, perhaps, nowadays, to transform extrinsic to intrinsic motivation and to furnish, thus, issues for self-regulated learning, as [18] suggests.

ACKNOWLEDGMENTS

The authors express their gratitude towards their colleagues from Telecom Bretagne, Julien Mallet and Gérald Ouvradou, for their active participation in the design and the instruction of the presented course. They also thank Antoine Beugnard and Fabien Dagnat for their initial support and Claire Lassudrie for her participation to the career workshop sessions. They are, in particular, thankful to the lab assistants for their precious technical support.

REFERENCES

[1] Organization for Economic Co-operation and Development. "New Millennium Learners. Initial Findings on the Effects of Digital Technologies on School-age Learners", *OECD/CERI Publications: Learning in the 21st Century: Research, Innovation and Policy*. 26 pages, 15–16 May 2008.

[2] Jones, B.D., Paretti, M.C., Hein, S.F., and Knott, T.W. "An Analysis of Motivation Constructs with Fist-Year Engineering Students: Relationships among Expectancies, Values, Achievement, and Career Plans". *Journal of Engineering Education*, Vol. 99, No4, pp. 319-336. 2010.

[3] Rouvrais, S., and Chelin, N. "Engineer Professional Identity: For an Early Clarification of Student's Perceptions". In *Electronic Proceedings of the 6th International CDIO Conference*, Ecole Polytechnique, Montréal. June 2010.

[4] Crawley, E., Malmqvist, J., Ostlund, S., and Brodeur, D. 2007. *Rethinking Engineering Education: The CDIO Approach*. Springer Verlag, 286 pages. 2007.
Details: http://www.cdio.org Accessed: 5 April 2011.

[5] Joint Task Force for Computing Curricula. *Computing Curricula 2005: The Overview Report*. 62 pages. Approved by IEEE-CS and ACM. 2005.

http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf Accessed: 5 April 2011.

[6] European Quality Assurance Network for Informatics Education. *Euro-Inf.: Framework Standards and Accreditation Criteria for Informatics Programmes*, 2009. http://www.eqanie.eu/pages/quality-label.php Accessed: 5 April 2011.

[7] Forbes, J., and Garcia, D.D. "...But What Do the Top-Rated Schools Do? A Survey of Introductory Computer Science Curricula". In *SIGCSE 2007*, Covington, KY, March 7-10, 2007.

[8] ACM K-12 Task Force Curriculum Committee. *A Model Curriculum for K-12 Computer Science: Final Report*. 45 pages. A. Tucker (editor). ACM, NY, 2003. http://www.acm.org/education/curric_vols/k12final1022.pdf Accessed: 5 April 2011.

[9] Wikipedia. "Grandes écoles". http://en.wikipedia.org/ Accessed: 5 April 2011.

[10] Power, A. "France's Educational Elite". *The Telegraph*. 17 November 2003. http://www.telegraph.co.uk/expat/4190728/Frances-educational-elite.html Accessed: 5 April 2011.

[11] Rouvrais, S., Ormrod, J., Landrac, G., Mallet, J., Gilliot, J. M., Thépaut, A., and Tremenbert, P. "A Mixed Project-based Learning Framework: Preparing and Developing Student Competencies in a French Grande Ecole". *EJEE European Journal of Engineering Education*, Vol. 31 (1), pp. 83-93. 2006.

[12] Anderson, L. W., and Krathwohl, D. R. (eds.). 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. 2nd Edition. Allyn & Bacon. Boston, MA, Pearson Education Group.

[13] Hussey, T., and Smith, P. "The Uses of Learning Outcomes", *Teaching in Higher Educations Journal*. Vol. 8 (3), pp. 357-368. 2003.

[14] Boud, D., and Feletti, G. (eds.). 1991. *The Challenge of Problem-based-Learning*. London: Kogan Page.

[15] Gruba, P., Moffat, A., Sondergaard, H., and Zobel, J. "What Drives Curriculum Change?". In *ACE'04: Proceedings of the sixth conference on Australasian computing education* – Vol. 30 Australian Computer Society. 2004.

[16] Simon, G. "Computer Science and Engineers: The (Bad) French Exception". Blog post, 23 January 2011. http://peerdal.blogspot.com/2011/01/computer-science-and-engineers-bad.html Accessed: 5 April 2011.

[17] Guzdial, M. "What do Scientists and Engineers Need to Know about Computer Science?". Post on BLOG@CACM. 27 July 2010. http://cacm.acm.org/blogs/blog-cacm Accessed: 5 April 2011.

[18] Zimmerman, B. J. "Investigating Self-Regulation and Motivation: Historical Background, Methodological Developments, and Future Prospects". *American Educational Research Journal*, Vol. 45, No. 1, pp. 166-183. 2008.

AUTHOR INFORMATION

Ioannis Kanellos Professor, ICT, linguistics and philosophy, Computer Science Department, Telecom Bretagne, ioannis.kanellos@telecom-bretagne.eu

Siegfried Rouvrais Associate Professor, Software Engineering and Architecture, Computer Science Department, Telecom Bretagne, siegfried.rouvrais@telecom-bretagne.eu