Sciences, Computing, Informatics: who is the keeper of the Real Faith?

Laura Benvenuti
School of Computer Science
Open Universiteit Nederland
Valkenburgerweg 177
6419 AT Heerlen,The Netherlands
lbe@ou.nl

Paul E. van der Vet
Dept. Of Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede, The Netherlands
p.e.vandervet@utwente.nl

Gerrit C. van der Veer
School of Computer Science
Open Universiteit Nederland
Valkenburgerweg 177
6419 AT Heerlen, The Netherlands
gvv@ou.nl

ABSTRACT
Computing, or informatics as we call it in Europe, covers many areas. In this paper we will discuss an important difference between two of these areas: software engineering and information systems. Epistemology, the study of the question: "What grounds can we justifiably have for believing the truth of assertions about reality?", is complex in informatics. This question has different answers, depending on the area we investigate. Curricula in informatics do not discuss this difference explicitly. In our opinion, they should.

Keywords
Computer Science Education, Information Science Education, Intellectual Discipline, Cultural Differences

ACM Classification Keywords
K.3.2 [Computer and Information Science Education]: curriculum

1. INTRODUCTION
Computer applications increasingly determine how we live. Our choices depend on the information we retrieve using (mobile) apps, the same applies to the services we as citizens interact with. These applications are written by computing practitioners. Some of them are trained on the job, others have an academic or a professional degree.

We will discuss one aspect of the education of computing practitioners. When we write "computing" we also mean "informatics", unless it is perfectly clear that we intend to attribute different meanings to these two words. We choose "computing" to remain consistent with the terminology of the ACM-IEEE curriculum recommendation series. We have considered using "computer science" to indicate the discipline, but rejected the option because the curriculum recommendation series use that term to indicate one of the computing disciplines (see section 2). Incidentally, we will use "computer science", but only while quoting other authors. Finally, we will use "informatics" in the discussion of European curricula in section 2 because that is how the discipline is called in Europe.

In this paper we express our concern about the education of computing practitioners. The discipline is young, and discussion about its focus and boundaries is taking place [6]. At the same time, the industry demands skilled professionals. How curricula in higher education should approach computing is a relevant question. If computing is seen as a competence, it is appropriate to focus on guidelines and recommendations - and how to apply them. But computing is an intellectual discipline too [41] [24], where creativity is accompanied by reasoning. From that point of view, it is appropriate to focus on the discussion of choices.

This discussion is not new. Software Engineering (SE), is aware of the problems encountered by practitioners. Making choices is an important issue in the undergraduate curriculum in SE ([2], page 40, Curriculum Guideline 8 (section "exercising critical judgment")) that is covered by offering a variety of methods and their backgrounds (Guideline 7 (section "computing")).

In our opinion, confining the discussion of the philosophical underpinning of methods in the academic setting is not enough. Future programmers should be able to motivate their choices in a setting we do not know yet. They should have tools to evaluate the methods they are acquainted with. They also should develop an intellectual attitude towards the profession of programmer.

Lewis, Jackson and Waite [24] looked at side-effects of higher education in our discipline. They investigated student attitudes early and late in an undergraduate Computer Science (CS) curriculum. One of the statements they proposed to 1st semester students, 2nd semester students and senior level students was: "When I solve a computer science problem, I explicitly think about which computer science ideas apply to the problem". The staff would have wanted the students to endorse this statement, and many of them did, but student endorsement declined from 72% for 1st semester students to 44% for senior level students. Lewis, Jackson and Waite's conclusion was that the CS curriculum might fall short in helping students develop the perspective that CS is an intellectual discipline.

Undergraduate computing curricula should emphasize the intellectual aspects of the disciplines, besides the competences. After all, today's students will be designing tomorrow's world. Discussion is necessary.

In section 2, we will look at undergraduate computing curricula in European higher education; in section 3 and 4 we point at a discrepancy between the administrative classification of computing and the historical roots of the discipline. In the sections 6 and 7 we discuss the nature of guidelines in software engineering; in 8 and 9 we look at the same aspect of another computing discipline, information systems. In section 10 we compare the approaches of these two disciplines. In the sections

11 and 12 we draw conclusions for the professional practice as well as for the academic curricula.

2. COMPUTING CURRICULA

In Europe, there is little agreement on the organization of higher education in computing or informatics. Harmonization is taking place [5] but is far from being accomplished. The United Kingdom has a system that resembles the US standard [11]. France has university education and top graduate schools or "Grandes Écoles"[8][9] awarding prestigious Master's degrees. Flanders[29] and the Netherlands [30] have a dual system with institutes for Higher Professional Education (Hogescholen) and Universities. Hogescholen generally confer Bachelor's degrees; in Flanders these degrees can be "academic"[19] while in the Netherlands they never are [31]. In Italy there is one system of academic degrees [13] that includes Engineering and Architecture disciplines. We will focus here on the first academic degree in computing or informatics, the (academic) Bachelor's degree.

The name of the discipline also reflects different views. In Europe, we use the term "informatics" as an umbrella [21], except for the U.K. where "computing" is preferred, in analogy with the U.S.A. [11]. The first name suggests an emphasis on automated information processing, the second on the computerization of calculus.

Surprisingly, there is agreement on the classification of the discipline. The conferred academic title in Europe, including the UK, is B.Sc. From a legal point of view, informatics is one of the Natural Sciences. In the Netherlands, it falls under the area "Nature", with Mathematics and the Natural Sciences, that often is grouped with Technology to the area "ß-wetenschappen" (ß sciences)[22], or "Exacte Wetenschappen" (exact sciences)[32]. In Italy the Informatics curricula are classified as "Scientific" and "Engineering" [13], in France [7] Informatics is listed in the domain "Sciences, Technologies et Santé" (Sciences, technology and health). In the UK, Computing is considered part of Science, Technology, Engineering and Mathematics (STEM) [26]. There are rare exceptions, for example the University of Groningen (NL) offers a full Information Systems-curriculum in the Faculty of Arts. It confers a BA title [36].

The ACM-IEEE Joint Task Force for Computing Curricula has defined a classification for academic curricula in the USA and in the UK [3], as an overview for the Computing Curricula series with guidelines and standards. There is not (yet) agreement on the content of academic curricula in informatics in Europe, but many national institutions for curriculum evaluation are inspired by the Computing Curricula recommendations, like the Italian "Bollino GRIN"[10] and the last Dutch report on the quality of Bachelor's degrees in Informatics [20]. For this reason, we will follow the Task Force in the terminology we will use, in particular in its definition of undergraduate computing curricula, i.e. of the bachelor's degrees in computer engineering, computer science, information systems, information technology and software engineering.

3. IS COMPUTING A SCIENCE?

The Task Force for Computing Curricula defines "computing" as "any goal-oriented activity requiring, benefitting from or creating computers" ([3], § 2.1). This definition is followed by the observation, in the same paragraph, that "an information system specialist will view computing somewhat differently from a software engineer", each computing area having its focus and perspective. The definition of robust methods for creating reliable artefacts is at the core of the SE agenda, whereas the information system (IS) perspective emphasizes generating, processing and distributing information using computer technology. These differences are seen as gradual; the common interest is computing.

European administrations too tend to consider computing as one discipline, one of the natural sciences. That is not obvious at all: in computing, unlike in the natural sciences, the object of investigation is artificial. Computing has that in common with engineering, but unlike in the case of engineering, the proof of soundness of computing theories is not necessarily situated in the outside world. Computing leans heavily on mathematics and formal logic.

The question "is computing a science" is a recurrent one. In 2007, P. Denning wrote in the Communications of the ACM "Computing is a Natural Science"[14]. In his opinion, computing has never been just a science of the artificial, but is an activity revealing deep structures of various natural processes. Denning also remarked that the acceptance of computing as a science is a recent development, that has taken place in less than a generation. Three years later, G. Génova reconsiders the scientific status of computer science [16] and argues for more speculative research in the discipline. The pendulum has swung too far in the direction of experimentalism. Experimentation and speculation should go hand in hand in all sciences but in computer science in particular. This discipline owes too much to theoretical research to focus principally on experimentation, says Génova.

4. MATHEMATICAL INTERMEZZO

Computing theories have their origin in D. Hilbert's formalist school, one of the answers to the 19th Century's crisis in the philosophy of mathematics. The German mathematician and philosopher David Hilbert wanted to provide for the foundation of number theory – and mathematics – in an axiomatic way, through formal logic. If number theory turned out to be consistent , then there had to be some sort of truth in it. Wegner and Golding [39] call this a rationalist point of view: Hilbert considered mathematical knowledge independent of sense experience.

We owe the theory of calculability to Hilbert and his school [23] in the early 20th Century at the University of Göttingen, that included Ernst Zermelo and John von Neumann [18]. But Hilbert's program failed. We know now that number theory can not be grounded in formal logic. Nor can mathematics, as an extension of number theory. We still do rely on mathematics and number theory, but the reason why is less clear.

In the sections 6 to 9 we will explore the role of mathematics in two computing disciplines: software engineering and information systems.

5. VERIFICATION AND VALIDATION

One of the fundamental issues for programmers is: how do we know that the machine will always do what we want it to do? The question has two aspects: 1. Did we build the right program (Validation) and 2. Did we build it right (Verification)[37].

According to the General Principles of Software validation of the FDA [15] **"**Software verification provides objective evidence that the design outputs of a particular phase of the

software development life cycle meet all of the specified requirements for that phase." (§ 3.1.2). In large and complex projects, software is specified formally. In these cases, verification concerns the correspondence between code and formal requirements and can be supported by formal proofs.

The FDA considers software validation to be "confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled." ([15], § 3.1.2). Validation concerns the correspondence between the product and what is needed in the real world. We will focus on this aspect of computing: are we making artefacts that are "valid"?

The story is becoming less "exact" at this point. In fact, there are different opinions about what should be considered "valid": does the adjective apply to the product, to the process, to the stakeholders' requests, or to the stakeholders' needs?

In real life, the concept of "users" can be considered in a narrow way as "anybody who actively uses the system" or in a broad way "anybody who is client of the services provided by the system". The second variant of the concept includes stakeholders who never touch the system but who "benefit from", or are "victim of" others using the system. Validity, in this case, concerns not only stakeholder specifications, but also stakeholder understanding and acceptance. We will coin this broad concept "stakeholder validity".

6. SOFTWARE ENGINEERING

The first area of computing we will investigate is software engineering. The most recent version of the "*IEEE/ACM Software Engineering Curriculum Recommendations"* specifies the nature of the discipline: "Software engineering thus is different in character from other engineering disciplines, due to both the intangible nature of software and to the discrete nature of software operations. It seeks to integrate the principles of mathematics and computer science with the engineering practices developed to produce tangible, physical artifacts." ([2], page 6)

Software engineering uses mathematics; but why? Formal methods are used to describe software in order to control the process of software construction: "The mathematical and engineering fundamentals of software engineering provide theoretical and scientific underpinning for the construction of software products with desired attributes. These fundamentals support describing software engineering products in a precise manner. They provide the mathematical foundations to model and facilitate reasoning about these products and their interrelations, as well as form the basis for a predictable design process". ([2], page 23)

Software, at least critical software, is described in a formal way first and translated into code afterwards. The question "does code meet its formal specification?" (verification) is answered by formal, mathematical proof. The other crucial question (does the code do what it was designed for?) can be answered in different ways: we can validate the correspondence between formal specifications and the stakeholders' requirements, we can validate the correspondence between formal specifications and his understanding and acceptance (stakeholder validity) and we can validate the software in the same way we validate other artifacts.

7. TWO INTERPRETATIONS OF "VALIDATION" IN SOFTWARE ENGINEERING

The *IEEE/ACM Software Engineering Curriculum Recommendations* is ambiguous at this point: "Requirements represent the real-world needs of users, customers, and other stakeholders affected by the system. The construction of requirements includes an analysis of the feasibility of the desired system, elicitation and analysis of stakeholders' needs, the creation of a precise description of what the system should and should not do along with any constraints on its operation and implementation, and the validation of this description or specification by the stakeholders". ([2], page 25).

We can read "precise descriptions" in two ways: as "formal specifications" or as "description by a finite number of objective, measurable criteria".

In the first interpretation, software is meant to fulfil a formal description. In this case, the software engineer is charged with the translation between the stakeholders' requirements, understanding and/or acceptance and the formal specifications of the product. In order to increase the confidence in the outcome of this process, future software engineers are provided with a solid background in mathematics. Mathematics is also used to design software and to prove its correctness. We trust this approach because we trust mathematics. This is the research area Génova wants to safeguard.

In the second interpretation, software is considered as an artefact, an answer to well defined and measurable needs. The process of translating those needs into formal specifications and from formal specifications to software is evaluated as a whole. In this interpretation of "precise description", we trust the validation process because we trust our observations, as we do in the natural sciences.

8. INFORMATION SYSTEMS

In the *ACM/AIS Curriculum Guidelines for Undergraduate Degree Programs in Information Systems* we read: "Information Systems as a field of academic studies encompasses (…) acquisition, deployment, management and strategy for information technology resources and services (…) and packaged system acquisition (...) for use in organizational processes.(…) The systems that deliver information and communication services in an organization combine both technical components and human operators and users. They capture, store, process, and communicate data, information, and knowledge." ([1] pag.13).

The *Foundations of Information Systems* course "is designed to introduce students to contemporary information systems and demonstrate how these systems are used throughout global organizations. The focus of this course will be on the key components of information systems - people, software, hardware, data, and communication technologies, and how these components can be integrated and managed to create competitive advantage." ([1]pag.36).

The operationalization is taught in the *Data and Information Management* course, which "provides the students with an introduction to the core concepts in data and information management. It is centred around the core skills of identifying organizational information requirements, modelling them using conceptual data modelling techniques, converting the conceptual data models into relational data models and

verifying its structural characteristics with normalization techniques, and implementing and utilizing a relational database using an industrial-strength database management system." ([1] pag.40).

The requirements are translated into relational data models and implemented with a relational database. The relational database management system is supposed to be provided by the industry fully conforming to the relational model.

Relational data models are formal representations of existing structures: libraries, population registers etc.. They are implemented into relational databases, which are filled with data representing the actual situation in the real world, or at least of the portion of the real world represented by the system.

This approach is based on the assumption that formal reasoning preserves truth. Let us consider the case of the library. The librarian wants to answer questions about his library, by querying the library database. The queries are written by a database engineer. The engineer assumes that the digital representation describes the real library correctly. He formulates each query in terms of entities in the model and convinces himself by formal reasoning that the queries correspond with the questions he wants to answer. If the questions are correct and the digital representation of the library matches the library, the answers to the queries should be true.

We trust the outcome of database queries because we trust formal reasoning.

9. ONE INTERPRETATION OF "VALIDATION" IN INFORMATION SYSTEMS

In the information systems discipline, the only possibility to describe software requirements is the formal one. The issue of the correctness of database management software is delegated to the software houses, which should implement the relational model, a mathematical model. Requirements for information systems are always formal; all the communication between professionals concerning information systems is supported by mathematics. This applies both to communication between practitioners and to communication between practitioners and their software houses. The other variant does not apply here.

One could reply that, even if the specifications of information systems are formal, their content concerns the real world. Can we consider these systems as artefacts, is it possible to skip the verification/validation sequence and match the input/output with real world situations? There are two reasons to answer "not always" to this question.

The first one concerns the system's dimension. It is always possible to match the real world situation with a positive answer to a database query, but this does not apply to the negative answer. I can look up a title in a library. The answer "You will find the book at location XX" can easily be checked. The answer "This book is not available" can only be confirmed by checking every single book in the library. This is possible in a small library, but most present-day libraries are too large and, moreover, they are distributed systems; the books are stored in different locations. Confirmation of the negative answer is seldom feasible in practice.

The second reason concerns the human factor in information systems and its dynamics. What is likely to happen in a very large library, where it is impossible for the operator to overview the situation, is that the operator will trust the system's negative answer and will discard the book when he happens to find it. The system's state will match the real world situation again, but the reason of the match is far from being scientific. We cannot validate this software by matching the system's behaviour with the real world because, unlike in the natural sciences, the "world" described by information systems is made to comply to the model.

10. THE NATURE OF GUIDELINES AND RECOMMENDATIONS

We have discussed the validation of software so far. The question we focus on here is not "why do we think our software is correct" but "why do we trust our guidelines and recommendations?"

Epistemology is the branch of philosophy studying the nature and limits of knowledge. One of the discussions in epistemology concerns rationalism versus empiricism. Rationalism claims that pure reason can be a source of knowledge. The opposite position is held by empiricism, claiming that all knowledge has its origin in sense experience [34]. Sciences, and natural sciences in particular, are essentially empiricist. Scientific knowledge is refined by application of the scientific method that is based on empirical and measurable evidence. The status of mathematics is the subject of debates. [40][28].

Validation of software is so complex because it concerns both the software in question and the discipline itself. If in software engineering, software performs unwanted behaviour despite of having been tested meticulously, and the reason why cannot be found, the discipline reconsiders its guidelines. Of the two interpretations of "validation" (the correspondence of the requirements and the formal specifications versus validation by empirical evidence), the empirical variant is the one that counts here. The discipline practices the scientific method. Epistemology is principally empiricist in software engineering.

Validation by empirical evidence is undeniable if the software is meant to produce results that are observable to everybody, if it controls processes in the physical world: software that commands the Mars Lander, or the Dutch Delta Works etc. Peter Denning's claim that computing reveals deep structures of nature matches with this approach to computing.

Validation of information systems occurs principally by comparing test results with their abstract counterpart. This process always entails a translation from the system's output to the abstract model. There is some objectiveness in this translation: professionals agree on how it should take place and agree on the definition of "malfunction of the system". There is less agreement on the definition of "malfunction" among the uninitiated ones. In real-life application of information systems, errors can remain invisible. Information systems relies principally on formal methods to validate both its software and its guidelines. The underlying assumption seems to be: if the model is consistent, then there has to be some truth in it. Epistemology appears to be principally rationalist here.

11. DISCUSSION

Apparently, the scientific interpretation does not fit the discipline of computing or informatics as a whole. In particular, it does not fit the construction of software that implements abstract models. Here, there is some room for different views on the status of theories, guidelines and recommendations.

Different branches of computing seem to have different reasons to trust their guidelines. Who is the Keeper of the Real Faith? Our conclusion is: the question does not fit the discipline anymore

The discipline has expanded. There are areas where the ultimate test lies in the real world and in the perceived functioning of software. In other areas the ultimate test is formal. Practitioners (and their stakeholders) should be made aware of this issue.

Information processing is a key issue in our discipline. But there is no decisive answer to the question what information is. One of the ideas on this subject was translated into a formal model, the relational model, and laid the foundation for extremely successful technology. Working with that technology requires a formal approach to the discipline.

We doubt the rationalist assumption "if the model is consistent, there has to be some truth in it". In particular, we reject the application of the notion of "truth" or "correctness" to software producing results that cannot be evaluated by the end users. What is the opposite of "correctness" in this case: "error" or "failure"? The unskilled user will only detect the latter.

Concerning the "validation" of software implementing an abstract model, it appears adequate to add criteria to the notion of "correctness", criteria expressing stakeholder validity. Among them, we mention "applicability", "usability", "efficiency" or "effectiveness". On which criteria to focus is a choice that depends on the context, a choice that precludes other choices. Therefore, it should be made explicitly.

Concerning the role of formal models in the computing practice, there seems to be more room for discussion on this point than most undergraduate curricula suggest.

12. SO WHAT?

We question the classification of computing among technology and the natural sciences because it blurs the objectives of the academic location of a discipline. An impartial discussion about the status of computing theories is unlikely to happen inside a faculty that has its right to exist in one of the possible outcomes of that discussion.

In our view, an impartial discussion might well end in the division of the discipline in "computing" (in accordance with the natural sciences), "engineering" and "informatics" (matching other criteria, depending on the context).

With regard to professional ethics, we witness that the classification of our discipline in the domain of the natural sciences entails an attitude of taking the benefits of technology too much for granted. We are afraid that the library will fit the designed system in the end. We want the system to fit the library instead. To facilitate this, future practitioners should be educated in discussing the criteria for application of the technology they use. They should be aware of the downsides of technologies, besides their benefits.

As regards the educational programmes, we recommend academic curricula in all the computing disciplines to pay more attention to this issue. Today, students get acquainted with different branches of computing, without discussing the differences. Academic curricula cover different application areas with the corresponding methodologies, without a discussion of the reason why different contexts request different methods.

We recommend discussing the backgrounds of theories explicitly. Each field of application has its own needs and accents: aerospace application programmers follow different guidelines than Web artists. Awareness of the backgrounds of theories helps practitioners to determine if the corresponding guidelines can be applied and how to do it properly.

13. REFERENCES

[1] ACM/AIS(2010): IS2010, Curriculum Guidelines for Undergraduate Degree Programs in Information Systems,

[2] ACM/IEEE Joint Task Force on Computing Curricula (2004) *Software Engineering 2004, curriculum Guidelines for Undxergraduate Degree Programs in Software Engineering*

[3] ACM/IEEE Joint Task Force on Computing Curricula (2005) *Computing Curricula 2005: the overview report,* Proceedings of the 37th SIGCSE technical symposium on Computer science education

[4] ACM/IEEE (2008): *Computer Science Curriculum 2008: An Interim revision of CS2001*

[5] Bologna Process, the official website, http://www.ehea.info/, 29.12.2010

[6] Cassel, Lilian N. (2007): *Understanding the entirety of modern Informatics*, Innovations in teaching and learning in computer science, vol. 6(3), pp. 3-11 (ISSN 1473-7507)

[7] Comité de Suivie de la Licence: *Recommendations année 2007-2008*, retrieved from http://www.enseignementsup-recherche.gouv.fr/cid21521/remise-du-rapport-dizambourg-a-valerie-pecresse.html, 29.12.2010

[8] Commission des Titres d'Ingénieur, *Higher Education in France* ,retrieved from http://www.cti-commission.fr/The-French-higher-education-system, 28.12.2010

[9] Commission des Titres d'Ingénieur, *Les Écoles françaises d'ingénieurs (the French engineering schools)*, retrieved from http://www.cti-commission.fr/IMG/pdf/GrandesEcoles.pdf, 28.12.2010

[10] Cortesi, A and E. Nardelli (2007),*Towards an European Certification of Computer Science Curricula*, in Innovations in teaching and learning in computer science, vol. 6(3), pp. 79-86 (ISSN 1473-7507)

[11] Cowling, A.(2006): A systems model for the field of Informatics, retrieved from http://www.ics.heacademy.ac.uk/education_europe/programme.htm, 24.12.2010

[12] *Decreet betreffende de herstructurering van het hoger onderwijs in Vlaanderen*, 14.8.2003, laatste wijziging 19.10.2010

[13] *Decreto Ministeriale 4 agosto 2000: Determinazione delle classi delle lauree universitarie,* Gazzetta Ufficiale 19 ottobre 2000 n.245 - Supplemento Ordinario n.170

[14] Denning, P.J.(2007): *Computing is a Natural Science*, Communications of the ACM, vol.50, no. 7, July 2007

[15] Food and Drug Administration, (2002) *General Principles of Software validation; Final Guidance for Industry and FDA Staff*". Food and Drug Administration. 11 January 2002. retrieved from http://www.fda.gov/downloads/MedicalDevices/Device

RegulationandGuidance/GuidanceDocuments/ucm085371.pdf, 18.09.2010.

[16] Genova, G (2010): *Is Computer Science Truly Scientific?*, Communications of the ACM, vol.53, no. 7, July 2010

[17] Goodman, Nicholas D (1979): *Mathematics as an Objective Science.* The American Mathematical Monthly 86.7 (1979): 540-551.

[18] Hilbert, retrieved from http://en.wikipedia.org/wiki/Hilbert 20.11.2010

[19] Het Hoger Onderwijsregister, http://www.hogeronderwijsregister.be/advanced-search, 28.12.2010

[20] Informatica. (2007). *Visitatierapport Quality Assurance Netherlands Universities* (QAUNU), Utrecht

[21] Informatics Europe, http://www.informatics-europe.org/about.php, retrieved 29.12.2010

[22] Larsen, V. and M. Lubbe (2008): *Quick Scan jong talent en de Wetenschap*, VSNU, 2008

[23] Lolli, G(2006): *La questione dei fondamenti fra matematica e filosofia*, in S. Albeverio e F. Minazzi (a cura di), *Matematica e filosofia*, nn. 14-15 di *Note di Matematica, Storia, Cultura* (Pristem/Storia, Università Bocconi, Milano), 2006, pp. 17-35.

[24] Lewis, C., M.H. Jackson, W.M. Waite (2010): S*tudent and Faculty Attitudes and Beliefs About Computer Science*, in *Communications of the ACM*, vol. 53, no5 pp. 78-85, May 2010

[25] Meyer, B. and W. Zwaenepoel (2006), *European Computer Science Takes Its Fate in Its Own hands*, in *Communications of the ACM*, vol. 49, no. 3, pp. 21-24, March 2006

[26] Mitchell, Bill (2011): The collapse of Computing Education in English Schools, retrieved from http://blog.sciencecampaign.org.uk/?p=2616, 5.1.2011.

[27] Popper, K.R. (1957) *Science: Conjectures and Refutations* in: M. Curd, J.A. Cover (eds): Philosophy of Science, the central issues, W.W. Norton & Company, inc., 1998, pp 3-10

[28] Putnam, H (1975). *What is mathematical truth?* In: Hilary Putnam, Mathematics, Matter, and Method, 2nd ed., Cambridge University Press, 1979, 60--78

[29] NVAO, *Higher Education System in Flanders,* http://www.nvao.net/higher-education-system-in-flanders, retrieved 28.12.2010

[30] NVAO, *Higher Education System in the Netherlands,* http://www.nvao.net/higher-education-system-in-the-netherlands, retrieved 28.12.2010

[31] NVAO (2008), Nederlands kwalificatieraamwerk Hoger Onderwijs, definitieve versie 15 december 2008.

[32] NWO, *Exacte Wetenschappen*, retrieved from http://www.nwo.nl/nwohome.nsf/pages/NWOP_5S7CLQ , 5.1.2011

[33] Sowa, J.F (2000). *Knowledge Representation. Logical, Philosophical and Computational foundations*, CENGAGE Learning 2000

[34] Stanford Encyclopedia of Philisophy, *Rationalism vs. Empiricism*, revision 6.8.2008, retrieved from http://stanford.library.usyd.edu.au/entries/rationalism-empiricism/, 9.3.2011

[35] Testa, C.(1974) , *An undergraduate/graduate program in information systems,* afips, pp.327, 1974 Proceedings of the National Computer Conference, 1974

[36] University of Groningen, Faculty of Arts, Curriculum Informatiekunde, http://www.rug.nl/let/onderwijs/Bachelor/Informatiekunde/studieprogramma, retrieved 13.3.2011

[37] Verification and Validation of software retrieved from http://en.wikipedia.org/wiki/Verification_and_Validation_(software), 19.9.2010

[38] *Wet op het hoger onderwijs en wetenschappelijk onderzoek*, http://wetten.overheid.nl/BWBR0005682/Hoofdstuk6/geldigheidsdatum_23-12-2010 retrieved 23.12.2010

[39] Wegner, P. and D. Goldin: *Principles of Problem Solving*, Communications of the ACM , vol. 49, no 7, July 2006

[40] Wikipedia, Formal sciences, part of a series on Science, retreived from http://en.wikipedia.org/wiki/Formal_sciences, 18.3.2011

[41] Wing, Jeannette M., *Computational Thinking*, in *Communications of the ACM*, vol. 49, no 3, pp. 33-35, March 2006,.