Balancing the mathematical content of computing curricula

Raymond Boute

Abstract

The main point of this note is the urgent necessity of establishing a proper balance in the mathematical content of university curricula in computing. Especially in situations where various local factors have inhibited curricula from growing into maturity, serious efforts are needed to establish conditions where "university level" is more than a pretence.

Indeed, the important role of computing technology in modern economy ensures such a multitude of tasks to be done at all levels that even departments that are "latecomers" in the field (or only indirectly concerned with computing) can, with considerable near-term impunity, find research niches and provide education far below the level that would be considered minimal in, for instance, electronics engineering. For this complacency and self-deception, however, the students will have to pay a heavy price by finding the half-life time of their education unnecessarily short and their own obsolescence a continuous threat, due to insufficient fundamental background for lifelong learning.

By analogy with education in more mature engineering disciplines such as electronics, an outline is given of the mathematical prerequisites for a computing engineering curriculum deserving the label "engineering". The minimalist approach typical for curricula inhibited by local traditions is clearly inadequate. It is also crucial to satisfy these prerequisites as early as possible in the curriculum, otherwise none of the other courses can be raised above the descriptive/technician level since the students lack proper preparation.

This note is a preliminary version; a few of the aspects are also discussed in [6].

0. Introduction and motivation

Engineering at the university level Technical, vocational and trade schools play a useful role in society by providing skilled electricians, mechanics, plumbers etc.. Indeed, it is instructive to ponder how many technical tasks can be fulfilled without scientific understanding, just based on ad hoc insights, available technology, and established routine procedures.

Engineering, however, is characterized[1] by the application of scientific principles. The use of mathematics is the most distinguishing element, and hence the issue considered here. Indeed, as Parnas observes, referring to computer engineering: *"Professional engineers can often be distinguished from other designers by the engineers' ability to use mathematical models to describe and analyze their products"* [28].

Although this epistemological principle is established *de facto* in engineering disciplines with a longstanding scientific tradition, in some computing curricula it is still underdeveloped.

When colleagues who teach classical engineering courses (say, electronics) happen to look at the computing courses in such curricula, they often make the remark that these courses are merely descriptive, lacking in intellectual content, and altogether little more than inflated programming courses—an opinion reinforced by the kind of assignments given to students (writing software and reports). Unfortunately, this remark is all too often justified.

Complacency as an impeding factor The indifference regarding the level of computing curricula is easily explained by the economic situation of the recent decades, where computer and information technology have seen continued growth with only rare periods of faltering.

[1]See the *Oxford English Reference Dictionary*, the *Webster's New World Dictionary* etc.

Due to this fast pace, there always are plenty of urgent development tasks to be done at all levels. Even people with no computing background who invest a minor effort in self-study can contribute usefully. With the urgent demand for quantity, and industry still uncertain about the exact qualifications software designers need, level and scientific basis become secondary.

Likewise, there are ample research niches for turning out Ph.D.s and papers that evidently are considered acceptable in the area of computing, but whose scientific level would be inadequate in more classical engineering areas, say, electronics.

Many students choose computing as they are looking for a mathematically less demanding engineering degree and for a sheltered way to reach a level that others attain by self-study. Some are already content if it helps them understand the computer magazines in newsstands. University administrators often prefer the easy way of large enrollment over quality.

So, considering this complacency in industry, academia and students, why devote effort to providing more serious computing curricula? We discuss only two of many reasons.

(a) Students deserve better Students enter the university for a university-level education. This implies a solid scientific basis to ensure (i) a deeper understanding of the material, extending its half-life time far beyond that of its technological embodiments of the moment; (ii) a systematic framework to facilitate the mastery of new subjects throughout the later career by lifelong learning. Indeed, economics change, and employment may become selective.

By its power of abstraction and assistance to reasoning, mathematics is the crucial element of a scientific basis: mere descriptive or intuitive approaches always fall short. The university is the best environment for acquiring this element, because mathematics needs concentration and gradual familiarization, and learning assistance is readily available. Moreover, once this opportunity is missed and the basis is deficient, repair during a later career will be much more difficult; for most this will even prove impossible, meaning there is no second chance.

Students put their trust in the university to provide the best possible curriculum, tacitly assuming that the required professional knowledge and sense of responsibility is available. They cannot verify this assumption in the short term. Specifically, in case demanding topics are kept out of the curriculum and computing courses are descriptive with insignificant mathematical content, students have no frame of reference for knowing that this is inadequate.

In that case, their trust has been violated and their professional obsolescence will proceed faster, while those responsible will not be available for making amends.

(b) A matter of professionalism No one would consider giving an electronics programme at the university level without using the benefits from differential and integral calculus or from transform methods. This background is assumed and routinely used throughout all other courses. Often more advanced mathematics is also taught.

The only possible excuse for a program with less math is when design tasks are excluded, e.g., in some Technicians Level programs: *"technicians generally do not perform electronic design work and, thus, the math skills required are not as rigorous as that of a 3-year technology program"* [23]. Still, many entry-level technicians programs *do* include calculus [24]!

In brief, the idea of omitting calculus from a university EE curriculum as being "too difficult for engineers" would be met with justified ridicule.

It is a sign of immaturity for software engineering that the idea of "leaving the mathematics out" is not treated likewise. Instead, occasionally one even hears that the mathematical methods arising from computing science are "too advanced" for undergraduate students!

Clearly, the chasm between software design and true engineering is so huge that it can be bridged only in the long term by education, teaching software engineering as true engineering.

1. Teaching "Software Engineering" as true Engineering

Professionalism in software engineering The recommendations of Dijkstra [12] and Gries [14, 15, 16] for software education are well-known, as are Parnas's observations [28]. This is also the rationale for the BESEME project [26]. A recent series of articles in the *Communications of the ACM* provides additional arguments in the same direction [8, 17].

A particularly refreshing outlook is offered by Holloway [18], who argues that the traditional jestification of formal methods by increased reliability or productivity involves many hidden assumptions as well as superfluous ones. He presents a much simpler argument in the form of a syllogism: software engineers aspire to be true engineers; true engineers use appropriate mathematics; hence software engineers should adopt the same approach.

Especially countries that, at least at the time of the 1999 TIMSS survey [32], represent the best tradition in mathematics education of the western world (Asia being the forefront) have a special duty in fostering this human resource in all engineering areas, computing included.

Formal Methods: the mathematics originating from Computing Science A serious danger to curriculum design is the misconception that "Formal Methods" is a rather restricted subject that can be taught in a small number of courses placed anywhere in the curriculum and yet can cover all CS-generated mathematics[2] relevant to computer and software engineering.

Gopalakrishnan [13] attributes this misconception to the fact that "Formal Methods" was introduced in Computing Science to indicate the use of mathematics—a practice so evident in other branches of engineering that a separate appellation may be considered redundant.

Still, the specific term could be justified by the fact that computer engineering requires a more formal kind of mathematics than the classical mathematical and engineering disciplines, a point also emphasized in Lamport's recent book on specifying systems [20]. The definition of formality, its importance in computing, and further references are discussed in [6].

However, more directly important for curriculum design are the following observations.

a. The body of nonclassical mathematical knowledge relevant to software engineering and sufficiently mature to be taught is staggering [13], in fact, considerably larger than the amount of classical mathematics usually included in computer engineering curricula.

b. Computing Science has generated a new mathematical style that is much more formal, yet practical, and even advantageous for teaching and research in classical mathematics.

Yet, this body of knowledge is rather recent (say, 35 years or less[3]) and the practical formal approach even more. As a result, their representation in computing curricula is very uneven.

Some curricula are entirely designed around a mathematical core as outlined in [26], creating a situation nearly as favorable as in classical engineering. Other curricula are designed around a pre-existing local body of lecturers who never studied the material themselves, and without hiring or otherwise involving a sufficient number of qualified lecturers from elsewhere.

The damage need not be permanent. Indeed, according to Max Planck [29]:

> *An important scientific innovation rarely makes its way by gradually winning over and converting its opponents: it rarely happens that Saul becomes Paul. What does happen is that its opponents gradually die out, and that the growing generation is familiarised with the ideas from the beginning.*

Still, this optimistic view holds only insofar as the new generation is given the opportunity to become familiarised with the ideas! This depends on the wisdom of curriculum designers. Unfortunately, it has been observed that curriculum design can also amount to censorship.

[2]The mathematics from Computing Science (Table 0, column CM), as opposed to classical math (col. EM).
[3]More recent than the classics by Turing and Von Numann or the "new math" educational disasters.

2. Balancing the mathematical content of computer engineering curricula

The mathematics content of engineering curricula Given the preceding observations and the large body of mathematical knowledge and educational insights relevant to computing generated in the past two decades [13], we propose a mathematics content as in Table 0.

	EM. Engineering Math.	CM. Computing Engineering Mathematics
Basic (general)	Analysis, Linear algebra Probability and Statistics Discrete mathematics (combinatorics, graphs)	Formal proposition and predicate calculus Relations, (h.-o.) functions, orderings Lambda calculus (basics) Lattice theory, Induction principles, ...
Targeted (modelling)	Physics, Circuit theory, Control theory Stochastic processes Information Theory, ...	Formal languages and automata Formal language semantics Concurrency (parallel, mobile calculi etc.) Type theory, ...
"Advanced"	Functional analysis Distribution Theory Hilbert and Banach spaces Measure theory, ...	Category theory Unified algebra Modal logic Co-algebras and co-induction, ...

Table 0: Mathematics for classical engineering and for computer/software engineering

Legend: "Basic" and "targeted" refer to undergraduate levels. "Basic" is domain-independent, "targeted" is more domain-specific, e.g., *Computing Mechanics* in Denning's setting [10]. The designation "advanced" is in quotes because it is debatable: given today's state of the art, some topics are arguably useful at the undergraduate level. EM is the mathematics for classical engineering, with electronics (EE) as a reference. CM is the mathematics for Computing Engineering (CE), and is meant to *complement*, not replace EM.

Balancing engineering mathematics and computing mathematics On one hand, one might argue against CM topics that they are not yet common everyday practice. However, the EM topics are not used daily either, yet universally recognized as being important enough for the intellectual formation of future engineers to be part of every EE curriculum. Indeed, the main thing in the mathematics is not the bare facts but learning to reason and model. Moreover, giving CM a central role in education is the only way for enabling future computing engineers to break the vicious circle and boost everyday industrial software design practice to a level of professionalism comparable to what is considered minimal in electronics engineering.

The other extreme is arguing that EM is of little use for software and higher-level hardware design, and hence superfluous in computing curricula. Some might applaud this as a refreshing break with tradition in computing curricula where an excessive amount of EM leaves no room for CM. Yet, omitting EM in a computing curriculum would be succumbing to the blatant utilitarianism that increasingly endangers the university level of education. For university-level engineers, mathematics is not just a tool but a part of their cultural heritage.

A proper balance would be a 50-50 allotment, assuming the equivalent of at least 1/3 of the undergraduate programme is devoted to the scientific basis outlined in the first two rows. *This implies more attention to the topics from CM than is customary in most current and even proposed computing curricula [25], but there is a very good reason for proposing it.*

Indeed, observe that successful applications of formal methods in industrial projects take place mostly at the Ph.D. level. The readership of the large body of useful results generated over the past decades about crucial (but difficult) areas such as concurrency often remains restricted de facto to researchers, without reaching the classroom. This situation is unsound, since much material is sufficiently mature and useful even in the undergraduate curriculum.

Educational approach An overview of contents is just a list, and the approach is at least as important.

The benefits of calculating formally and developing intuition in formal calculation cannot be overemphasized [6]. This differs from the traditional approach where, for instance, predicate calculus is taught informally, and quantifiers ($\forall$, $\exists$) appear only as shorthand for natural language ("for all", "there exists"), without developing this into a calculus that makes quantified formulas a useful reasoning tool [4, 15].

Indeed, although in Lethbridge's survey [21] software practitioners did not list formal logic along with calculus among the courses they consider useless, neither did they include it in the list of topics that made a lasting impression. This indicates that logic is not sufficiently taught in a way that makes it useful for everyday practice [16]. It is also important to keep in mind that high school has offered the students little preparation for logic.

For these reasons, the idea of covering logic in a "quick course" must be resisted.

Similarly, computer tools may help, but can never be a substitute for solid mathematics. There is a certain analogy with the role of packages like Maple or Mathematica in calculus courses. However, the successful use of such packages by students makes it easy to overlook how heavily this relies on the fact that high school provides significant preparation for the underlying mathematical concepts and conventions. Hence analogous expectations are not justified for logic software: the situation is different beyond comparison.

Researchers and educators note that symbolic computation packages are meant to eliminate "wearisome calculations" for the [experienced] user, but also warns that such calculations play an essential role in the development of the understanding of beginning students [30].

Still, simple tools may help students by lowering the level of abstraction and facilitating the discovery of their own mistakes. However, this assumes a receptive attitude: students are often less than enthusiastic about learning "yet another language; are C++ and Java not enough?", or blame the tool for errors in their own specification or hidden assumptions. Such unreceptive attitudes are seldom the students' fault but generally the result of poor curriculum design or an inadequate education environment where minimalism prevails[4].

Should education reflect current SWE practice? Many SWE courses are based on textbooks describing various "methodologies". Such an approach is favored by students who prefer material that makes fewer mathematical demands, yet gives the impression of direct usefulness in industry. This impression is debatable, since different companies make different choices about software design procedures, which therefore are best learned on the job.

In fact, Bruce et al. [8] observe that just-in-time learning is the province of on-the-job training. This is very apt for the "methodology" topics that, according to Lethbridge's survey [21], software practitioners currently encounter most in their everyday work.

More importantly, since current SWE practice does not approximate the level of professionalism of classical engineering, education should not mirror it, lest the vicious circle be perpetuated. Raising professionalism requires sustained injection with fresh graduates having a superior background. Students and industry are best served by focusing on solid foundations, principles and topics that can only be mastered effectively via university education.

A psychological problem is that the mathematics of computing is relatively demanding. This makes it easy to oppose its integration into the curriculum with the unfair argument that it does not solve all problems of the software crisis. Evidently, such arguments (not always unintentionally!) blur the distinction between necessary and sufficient conditions.

Indeed, on the heavy road towards professionalizing software engineering, mathematics can modestly claim only necessity, where easier "methodologies" freely claim sufficiency.

[4]A typical error is giving in to beginners' insistence that learning a new programming language be given ample classroom time rather than result from self-study. This yields cheap credits for the students, but the time thus wasted forces omitting harder topics that *truly* benefit from exposition and explanation in class!

3. Some final remarks

Impeding factors Realizing curricula matching the above rationale is increasingly difficult.

Most successful are universities where CS curricula developed early (about 1965-70) under the direct influence of computing pioneers. Universities that started later (say, 15 years) but took computing seriously as a scientific discipline were still able to catch up by attracting people from elsewhere with suitable background and vision. Problematic are latecomers where computing is just a side activity of various departments and where compensation by influx of personnel from elsewhere is minimal, and incentive to improve the situation is low.

There are strong social pressures to reduce the mathematical content of engineering curricula. It is at least strange that the academic community has done so little to resist these pressures, since epistemology quite transcends ephemeral issues like social preferences, and it is a disservice to the student community to provide less than the level it has the right to expect at a university. By lack of solid tradition, computing education is especially vulnerable.

Another factor are local traditions and vested interests. The importance and difficulty of obtaining "buy-in" from colleagues for establishing a high-quality educational programme in computing has been emphasized by Page [26], Wing[34] and others. The fact that any serious improvement of the curriculum would require that, borrowing Page's euphemism, "most instructors must revise notes" is a major (hidden) motive for opposing change.

In the worst case, even the potential for gradual improvement is curtailed, for instance, by tolerating only a token of CM-related material and placing it late in the curriculum. First, students will be adversely motivated towards such material, and see it as ballast by lack of opportunity to apply it in other courses. Moreover, lecturers who might aim for a gradual increase of mathematical methods in their courses are prohibited from doing so because students lack the background, unless a major proportion of the course is set aside for ad hoc patches. It is simpler to lay uniform, solid foundations early, letting later courses benefit.

The European context In Europe, the current Bachelor/Master reform following the Sorbonne/Bologna agreements has reduced the space allotted to the fundamentals at many universities. Indeed, the new Bachelor programme is only three years in many countries, as opposed to four in the U.S. [33]. This short duration, together with the pressure to have more courses specific to the chosen technical specialty early in the curriculum, has compromised the scientific basis, in particular the mathematical foundations, to a very large extent.

These developments cause the continuation an earlier worldwide trend whose disastrous consequences for the intellectual formation of students have already been observed even in well-established disciplines such as mathematical analysis and physics. We give a few examples, found at various occasions on the web (e.g., Amazon customer reviews). About Rudin's classic analysis text [31], Mikhail Ostrovskii noted that, based on educational experience with undergraduates, it has become "too advanced for the 21st century". Conversely, the degree to which some recent books have succumbed to the downward trend leads to bitter criticisms:

> [The author] appears to vainly grasping at the microscopic attention spans of his weakest readers. After all, we live in an era where teachers must entice their TV-addicted, suburbanite students to work, using books with beautiful covers and lots of flashy pictures inside. Many teachers no longer dare to challenge their students with difficult readings or problem sets because they will be harrassed by endless complaints and negative evaluations (bad news for profs seeking tenure).
> [...] another reviewer [...] summed it all up far better than I. To paraphrase...
> If you don't know any linear algebra
> Or if you don't know anything about differential equations
> And are glad of it because you just don't like math
> Then [this book's author] is your man."

We omit the reference, but Google provides the full text within a second. It is sad to observe that, in well-established disciplines as mentioned, many serious teachers must recommend for their classes textbooks of rougly 30 years ago if they want level and quality.

How much graver are the consequences for younger disciplines such as the mathematics for software engineering, especially formal logic, which rarely had an established position to start with! In some computing curricula, the scientific basis is simply discarded, necessarily restricting the level of all other courses. Ultimately the students are the victims.

What is being squandered within the time span of a few years will take a generation to recover when the need is realized again. At the same time, one hears complaints from various sides about the gradual shift of engineering towards Asian countries.

These European developments are just a large-scale version of what occurred in The Netherlands about 15 years ago. The erstwhile Minister of Education, in collusion with some colleagues from other countries, reduced the nominal duration of the (Dutch) engineering curricula from 5 years to 4, claiming industry support, ignoring all arguments from universities, and making promises about a "second phase" (Masters degree) that never materialized. Few years after the first graduates emerged, companies complained about the reduced quality, and Philips even openly stated henceforth preferring engineers from Leuven over those from Eindhoven. Only several years later was the original 5-year duration re-established.

The authoritarian way in which the Sorbonne/Bologna agreements and their interpretations by local politicians are imposed on the universities—with only a token opposition and considerable opportunism from the latter side—suggests that we are witnessing a rerun of the same story.

In particular, it is virtually certain that industry, after perhaps an initial impression of more "immediate gratification" lasting a few years, will realize the lack of substantial foundations in the new graduates as compared to those from preceding curricula. The new generation will face faster obsolescence, Asian countries will have taken a serious lead, and the surprise of the responsible politicians in government and academia will be feigned: they are fully aware of the damage they are inflicting upon education, yet are more keen on leaving their imprint on history even though they are incapable of doing so constructively.

References

[1] Vicki L. Almstrum, "Investigating Student Difficulties With Mathematical Logic", in: C. Neville Dean and Michael G. Hinchey, eds., *Teaching and Learning Formal Methods*, pp. 131–160. Academic Press (1996)

[2] Vicki L. Almstrum, "What is the Attraction to Computing?", *Comm. ACM 46*, 9, pp. 51–55 (Sept. 2003)

[3] Eerke Boiten and Bernhard Möller, *Sixth International Conference on Mathematics of Program Construction* (Conference announcement), Dagstuhl (2002). `http://www.cs.kent.ac.uk/events/conf/2002/mpc2002`

[4] Raymond T. Boute, *Functional Mathematics: a Unifying Declarative and Calculational Approach to Systems, Circuits and Programs — Part I: Basic Mathematics.* Course text, Ghent (2002)

[5] Raymond Boute and Hannes Verlinde, "Functionals for the Semantic Specification of Temporal Formulas for Model Checking", in: Hartmut König, Monika Heiner, Adam Wolisz, eds., *FORTE 2003 Work-in-Progress Papers*, pp. 23–28. BTU Cottbus Computer Science Reports (2003). `http://www-rnks.informatik.tu-cottbus.de/events/forte2003/inhalt/detailed_program.html`

[6] Raymond Boute, "Can lightweight formal methods carry the weight?", in: David Duce et al., eds., *Teaching Formal Methods — Practice and Experience.* Oxford (Dec. 2003) `http://wwwcms.brookes.ac.uk/tfm2003/`

[7] Raymond Boute, "Integrating formal methods by unifying abstractions", in: Eerke Boiten et al., eds., *Fourth International Conference on Integrating Formal Methods*. Canterbury (Apr. 2004) `http://www.cs.kent.ac.uk/events/conf/2004/ifm/`

[8] Kim B. Bruce et al., "Why Math?", *Comm. ACM 46*, 9, pp. 40–44 (Sept. 2003)

[9] C. Neville Dean and Michael G. Hinchey, *Teaching and Learning Formal Methods*. Academic Press, London, (1996)

[10] Peter J. Denning, "Great Principles of Computing", *Comm. ACM 46*, 11, pp. 15-20 (Nov. 2003)

[11] Edsger W. Dijkstra and Carel S. Scholten, *Predicate Calculus and Program Semantics*. Springer-Verlag, Berlin (1990)

[12] Edsger W. Dijkstra, *Under the spell of Leibniz's dream*. EWD1298 (April 2000). `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1298.pdf`

[13] Ganesh Gopalakrishnan, *Computation Engineering: Formal Specification and Verification Methods* (Aug. 2003). `http://www.cs.utah.edu/classes/cs6110/lectures/CH1/ch1.pdf`

[14] David Gries, "Improving the curriculum through the teaching of calculation and discrimination", *Communications of the ACM 34*, 3, pp. 45–55 (March 1991)

[15] David Gries and Fred B. Schneider, *A Logical Approach to Discrete Math*. Springer (1993)

[16] David Gries, "The need for education in useful formal logic", *IEEE Computer 29*, 4, pp. 29–30 (April 1996)

[17] Peter B. Henderson, "Mathematical Reasoning in Software Engineering Education", *Comm. ACM 46*, 9, pp. 45–50 (Sept. 2003)

[18] Michael Holloway, "Why engineers should consider formal methods", *Proc. 16th. Digital Avionics Systems Conference* (Oct. 1997) `http://techreports.larc.nasa.gov/ltrs/PDF/1997/mtg/NASA-97-16dasc-cmh.pdf`

[19] Philip N. Johnson-Laird, (example problems in the psychological study of human reasoning), `http://www.princeton.edu/~psych/PsychSite/fac_phil.html`

[20] Leslie Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Pearson Education Inc. (2002)

[21] Timothy C. Lethbridge, *The Relevance of Education to Software Practitioners: Data from the 1998 Survey*. University of Ottawa CS TR-99-06 (July 1999). `http://www.site.uottawa.ca/~tcl/edrel/`

[22] Zohar Manna and Amir Pnueli, *The Temporal Logic of Reactive and Concurrent Systems — Specification*. Springer-Verlag, New York (1992)

[23] N.N., *St. Lawrence Program for Electronics Engineering Technician* `http://www.passpathways.on.ca/resources/PASSWEB/page93.html`

[24] N.N., *Selland College Electronics Technology Programme* `http://selland.boisestate.edu/academic_programs/programs/ETAAS.htm`

[25] N.N., *ACM Curricula Recommendations* `http://www.acm.org/education/curricula.html`

[26] Rex L. Page, "Software is discrete mathematics", *Proc. 8th ACM SIGPLAN intl. conf. on Functional Programming*, 1, pp. 79–86 (2003). `http//www.cs.ou.edu/~beseme/besemePres.pdf`

[27] David L. Parnas, "Education for Computing Professionals", *IEEE COMPUTER 23*, 1, pp. 17–22 (Jan. 1990)

[28] David L. Parnas, "Predicate Logic for Software Engineering", *IEEE Transactions on Software Engineering 19*, 9, pp. 856–862 (Sept. 1993)

[29] Max Planck, (quote) http://www-gap.dcs.st-and.ac.uk/~history/Quotations/Planck.html

[30] Raymond Ravaglia, Theodore Alper, Marianna Rozenfeld, Patrick Suppes, "Successful pedagogical applications of symbolic computation", in: N. Kajler, *Computer-Human Interaction in Symbolic Computation.* Springer, 1999. http://www-epgy.stanford.edu/research/chapter4.pdf

[31] Walter Rudin, *Principles of Mathematical Analysis.* McGraw-Hill (1976)

[32] Trends in International Mathematics and Science Study
http://nces.ed.gov/timss/results.asp

[33] U.S. Department of State, "Bachelor's and Associate Degrees", in:
http://educationusa.state.gov/admissions/undergraduate/education.htm

[34] Jeannette M. Wing, "Weaving Formal Methods into the Undergraduate Curriculum", *Proc. 8th Intl. Conf. on Algebraic Methodology and Software Technology (AMAST)* pp. 2–7 (May 2000)
http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/calder/www/amast00.html

[35] Steven Zucker, "Teaching at the University Level", *Notices of the AMS 43*, 8, pp. 863–865 (1996)

Appendix: the situation in Belgium

Wasting opportunities At the time of the 1999 TIMSS survey [32], the Flemish part of Belgium represented the best tradition in mathematics education of the western world, Asian countries having the lead. This entails a special responsibility in fostering this resource.

Unfortunately, precisely here we have seen unprecedented squandering in the recent past. The entrance examination for engineering students was considerably reduced a few years ago, and altogether eliminated starting 2004 by ministerial decision, the openly stated purpose being to attract more engineering students—an illusory or at best short-lived effect.

Whereas the earlier entrance requirements for engineering have long sustained a track in secondary education with a solid mathematical content, already now secondary school mathematics teachers are reporting the negative effects of the reduced requirements on the student motivation for this track, which is rapidly approaching extinction. At some universities, the use of computer tools is invoked as a reassurance that we can afford such developments, which obviously is just another epistemological illusion [30] with dire consequences.

What is being squandered within the time span of a few years will take a generation to recover when the need is realized again. At the same time, one hears complaints from various sides about the gradual shift of engineering towards Asian countries. Don't they deserve it?