Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 1. pp. 269–278
doi: 10.14794/ICAI.9.2014.1.269

Upgrading the ICT curriculum of a multidisciplinary degree evolving from practical orientation to university integration

Kris Aerts

KU Leuven@KHLim
Kris.Aerts@kuleuven.be

Abstract

In Flanders, Belgium the master degree of industrial engineering has just been integrated into the universities, imposing a shift from a more practical point of view to a tighter relation between (applied) research and education. A similar process is taking place in several European countries. This paper tells the success story of how to upgrade the ICT component of the program while maintaining both the success rate and the satisfaction of students.

As always, the challenge is set by the context. The multidisciplinary character of our degree where Electronics/ICT is just one of 6 options after a common base of 94 credit points results in many small courses – typically just 3 credit points – and a friction between students disliking ICT and a thorough preparation for consecutive ICT courses in the major Electronics/ICT.

The paper presents several design choices throughout the 4 year program, such as the compromise between objects first and algorithms first in the first course involving a tiny bit of VBA in Excel and a focus on Java; early exposure to MVC and other design principles; and the integration of our research in the master courses by gradually exposing it to the students, from a voluntary topic in a course with capita selecta, over master thesises to a devoted course open to every student. This helped us in formulating our research topic more precisely, especially for newcomers to the subject.

The paper also presents a BlueJ extension which enables students to autonomously evaluate their adherence to international coding standards and the use of Javadoc.

Finally, our counter-motto is "Modest expectations give modest results". We always try to challenge students by giving assignments they love working on and that can make them proud of their achievements. Our ultimate reward is the proud sparkle in the eye of students, e.g. presenting their own Java

program with animations, threads and MVC after just 6 credit points of programming.

Keywords: multidisciplinary ICT education, research integration, Java, BlueJ

MSC: 68-00, 68-01, 68N15

1. Context

Europe is a diverse continent where, despite the European Union, countries can still decide autonomously on a lot of issues, reflecting the diverse history of each of the participating countries and the fact that Europe started as a economic union. On the other side of the spectrum we have language – a very noticeable difference – and education, where countries have their own legislation, habits and history, with an origin often long before the advent of the European Union. For example in Belgium and France it is still mostly based on the Napoleon system, but many other systems exist. [1, 2]

In the 80's and beginning of the 90's Belgium had a ternary system:

- HOBU - KT: an acronym for higher education outside university - short type, where the addition of short type indicated a 3 year program (in the 70's sometimes also 2 years for some degrees)
- HOBU - LT: again outside university but of the long type: typically 4 year programs. These programs are comparable to university degrees in a number of countries. It was written in law that these programs are of "academic level". However, because they were not physically part of a university, there were often not recognised as such outside Belgium.
- University: is of course university as we know it.

In an effort to align the structure of higher education all over Europe and to improve the international recognition of the different degrees, the Bologna declaration of 1991 [3, 4, 5] imposed a binary division between bachelor and master degrees.

Belgium did this by moving the second bullet (HOBU - LT) to the third bullet (university) as these degrees were already of academic level, and by rebranding the first bullet to professional bachelor where the addition of "professional" reflects the fact that the degrees are directly targeted at a profession, where students of the other bachelors (of bullet 2 and 3) are supposed to continue with their masters at their academic institution. Hence the labelling "academic".

However, despite the fact that it easy to explain, the process to perform was far from trivial, for a number of reasons. The only reason relevant for this paper is the fact that this implied a sort of upgrade for the curriculum, mostly in the form of integrating research into the curriculum. We took the maximalist approach and didn't stop by adding some research results but actually reformed the entire ICT-trajectory of the curriculum with a distinct focus on software engineering.

We hope that the ideas and critical design decisions presented in this paper may be of interest to anyone involved in similar processes such as country wide educational reforms, technical universities integrating into (research) universities or individual institutions seeking to advance in IT or to integrate more research into their programs.

1.1. Joint Faculty of Engineering Technology

Figure 1 shows the 7 campuses where our programme for BSc and MSc in Industrial Sciences is offered[1] After a broad, multi-disciplinary common base in the first 3 semesters the students have to choose their specialisation. where Electronics/ICT is only one of 9 options. This may make the comparison with other degrees a bit more difficult because ICT is typically either a main component of the programme or only a 'tool' to achieve something in another course – in the best case for programming or scripting, but often only by configuration or employing skills in the particular tool.

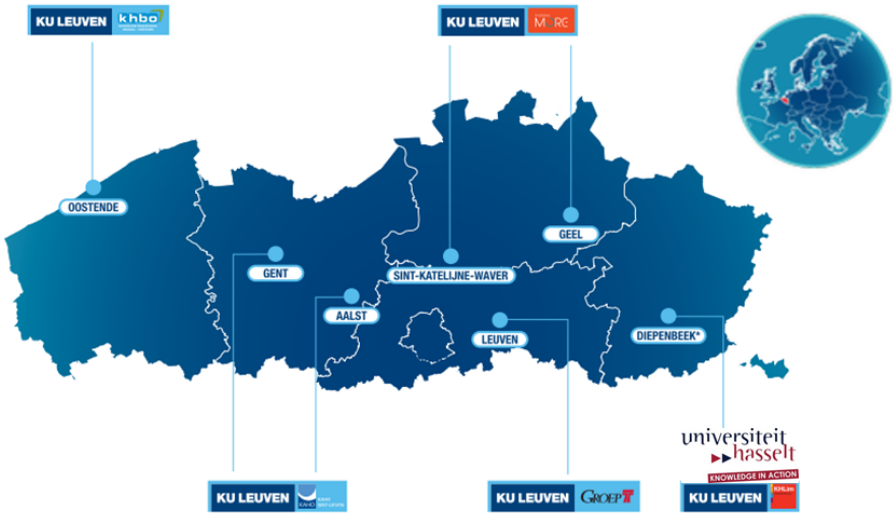

Figure 1: Geographical overview of campuses

In our curriculum it is neither: 6 of the 90 credit points in the common base have been allotted to ICT. This is little compared to specialised ICT degrees but we believe (and will show) that it is enough to give a thorough understanding of the design techniques used in developing software and insight in the complexity of making and maintaining software.

[1] Actually, this paper is specific for the programme in Diepenbeek/Hasselt only, where it is taught in the Joint Faculty of Engineering Technology of KU Leuven and UHasselt, with some deviations from the general program.

However, the fact that the first 3 semester are common for all 9 options, puts some stress on the 'freedom to operate' in (at least) 3 ways:

- there is always friction between students (and colleagues alike) how may complain that <fill in the blank> is not useful for their education and that they'd rather learn netiquette or how to make nice graphs in Excel on the one hand and collegues from the major who expect you to go as deep as possible in preparation for their major

- being attractive as in 'attracting more students for your major' is never an explicit goal, but always implicitly present

- courses are scattered and divided in small units. A typical ICT-course, especially in the first 3-4 semesters, is typically 3 credits, worth for up to 90 hours of student effort. So courses are expected to consist of small but self contained packages with little room for diversion and side stories. However, we try to provide 'cliff hangers' to the next course and/or our major.

2. The real challenge

Characteristic	Interpretation	Trend	Explanation
Raw material	Capabilities of incoming students	equal	Students are not better than 10 years ago. They may have more skills in *using computers*, but not in programming.
Final Product	Learning outcome	major increase	More concepts must be understood to properly design software
Processing time	Time spent by students	equal & limited	Actual time is fixed at max. 27h per credit point
Cost	Time spent by tutors	decrease	More students and more courses should be handled.
Customer Satisfaction	Evaluation by peers and students	equal or increase	Students have become more assertive and their evaluation/perception plays an important role in the evaluation of the tutors

Table 1: Demands for an updated ICT programme

3. Vision on education

Reflecting our origin as a practically oriented, yet open-ended education, we prefer competences over pure theoretical knowledge, e.g. instead of giving an overview of all design patterns with a written exam about the patterns, we'd rather have them study a few patterns in depth including designing and building an application.

This small example already covers out three main motto's: focus, the analogy with a drivers license and putting the stakes high, which are omnipresent throughout all our courses.

3.1. Focus

A striking feature of our programme is that an ICT course typically contains 3 or 4 credit points, corresponding to an expected student effort of between 80 and 105 hours, which consists of following and preparing lectures, executing tasks, studying for and doing exams. This is relatively small compared to most courses at other institutions. Partially this is a consequence of our multi-disciplinary programme, but it is also an explicit choice: by having only relatively small courses, both tutors and students must keep focus. One must carefully design courses with only the most relevant topics: there is little time for long introductions and sideways. The actual learning outcome must be clearly identified. Students benefit from this: one the hand it gives them more easily the impression that the course is manageable, resulting in a higher motivation. On the other hand because we focus on a limited set of design methodologies per course and let the students explore them thoroughly, they become well acquainted with them. This is the typical motto of less is more. They may know less topics but understand them more in depth. We believe that picking up other methodologies then becomes more easy because they have had good experiences applying well known and generally approved ones, and are open to learning others.

However, despite the fact that we focus, we make sure that every course has open ends. There are always features mentioned but at most partially explored. The projects assigned to the students have an open scope: for example the features requested are described more and more vaguely as students progress in the curriculum. The good students are encouraged to go further (and often do so).

3.2. Analogy with a drivers license

Using anecdotes to explain something is a well known rhetorical technique. For our ICT courses we use the drivers license. Despite the fact that some have to try harder, practically everybody has a drivers license. We persuade our students that the same applies for our courses. While for some students the course may seem trivial, others have to struggle more, some even a lot, but perseverance helps. Practicing is the only solution. Just like no one even considers driving all night long just before the drivers license test, studying all night long before a software engineering exam is of no use.

3.3. Putting the stakes high

Although we have focus, we are not modest in our expectations. We carefully respect the counter-motto: "Modest expectations give modest results".The end result is always something students cannot reach (too) easily. After going through a phase of frustration they reach a level of exaltation and deliver a product of which they are genuinely proud.

We expect commitment and show commitment by giving extensive guidance, quick responses to emails and thinking along with their design decisions. But we expect results as well. This is also a facet of putting the stakes high at both sides.

By putting the stakes high and rewarding them properly we combine the typical external stimulus (the credit cannot be earnt without spending enough time) with intrinsic motivation: students loving to work on the subject and becoming proud of their deliverable, also giving great satisfaction to the tutors.

4. CS1 and CS2 courses

The CS1 and CS2 courses, each having 3 credit points, are for most students the only software engineering component in their programme. The final learning outcome is hands on experience with one of the most fundamental architectural patterns in software engineering: model-view-controller. This gives them a good understanding of the complexity involved in developing software and insights in how applying architectural patterns helps in solving problems.

In CS1 the focus is on object oriented programming. In the continuous struggle between objects first versus algorithms first we fulfil our Belgian role of making compromise by first having one credit point of 'algorithms first' in VBA in the comfortable environment of Excel, before switching to our main focus of objects first in Java. The focus in VBA is not on the features of Excel, but on functional decomposition, variables, parameters and control structures (if, for, while). A short assignment helped in increasing the initially very low points.

The Java part really stresses object orientation. We proceed from composition to arraylists and introduce inheritance, citing the Liskov substitution principle (one of the rare cases where we drop a name to stress the importance of the concept) to arrive at lists containing different types of objects. Javadoc is another focus of attention as it shows the fact that classes are to be used by other people, and as it is a concept that is applicable in other fields engineering where processes and connections must be well documented. We use BlueJ, the pedagogical programming environment created by the university of Kent [6].

The knowledge of Java and object orientation is expanded in CS2 and accumulates in students building a self chosen graphical application in Java (most select a mini-game, but this is not compulsory) using the model view architectural pattern. Extra techniques introduced include inheritance through interface, threads, double buffering. Study time measurement has shown that the average load is 31 hours per credit point, where the target is between 25 and 30 hours per credit point,

indicating that the programme is ambitious, but reachable. The overall score of the student evaluation of the course was very high as well, with a top rating of 92.

5. BlueJ Extension for self evaluation

The previous sections already introduced the tension between giving a lot of feedback to students to help them in reaching a higher level on the one hand and the inclination to cost efficiency on the other hand. One approach to solve this conflict is to give more power to the students. The more they can evaluate their programs autonomously, the more effective the time spent on feedback.

Luckily, two important evaluation criteria, compliance to international coding standards (camelCase, getters & setters, ...) [6] and the use of Javadoc can be semi-automatically evaluated. For this purpose we have developed a BlueJ-plugin Self evaluation, which checks

- for camel case (in names exceeding a certain length)
- whether instance variables have a getter and setter method
- the presence of javadoc, including @param and @return
- the absence of public instance variables
- common mistakes such as = instead of == and in String comparison == instead of .equals()

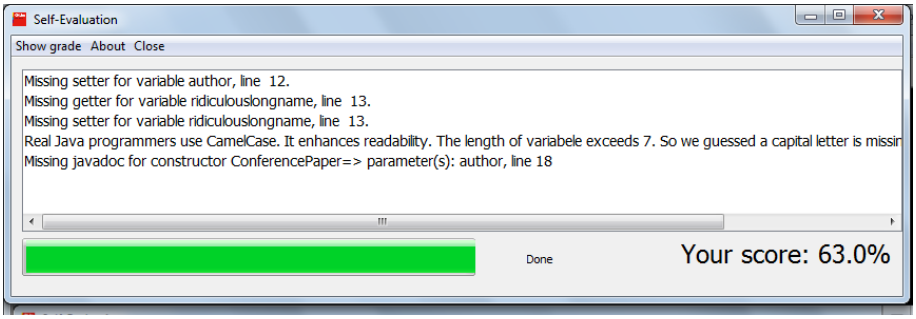

Figure 2: Evaluation of a Java class

The result of evaluation is a percentage where students should try to get 100

Because each of the 'errors' can be intentional, programmers can add the @Intentional-annotation to assist the extension in calculating the appropriate score.

6. General suggestions

From the lessons learnt we would like to make some suggestions to anyone changing their curriculum. Although revolution may be suitable in some situations, we find it undesirable, perhaps even infeasible to upgrade an entire curriculum at once. Different, consecutive courses have to be aligned and it is better to stabilise the first courses before proceeding with the next courses at the risk of having to backtrack.

Therefore we suggest to proceed stepwise. We have obtained very good results by working both at the head (CS1 and CS2 courses) and the tail (master courses). By setting a solid base in the introductory courses, and not beginning too soft, students are getting acquainted with high standards. Even when follow up courses do not (yet) meet the same standards, students are used to work hard for ICT courses and will continue to do so.

It may be a good idea to work on the master courses in parallel with the beginners courses. Students are a bit more mature and can in most cases cope with some additional complexity, certainly when the context of integrating into the university (in our case) is explained to the students. However the change should not be too harsh. As explained in the following sections, we gradually integrated our research in the master course.

Afterwards, when both head and tail have been more or less stabilised, the effort can shift from the beginning courses to the intermediate courses. In our programme we introduced courses on Database-programming and Cloud computing & service oriented architectures, but this depends on the profile of the degree.

7. Integrating research

The possibilities to integrate your own research into the curriculum depend heavily on the subject of the research, the institute and the actual programme for the degree. Our approach is particularly interesting because the research topic: using Lava [ref] to generate constrained cryptographic hardware by applying design space exploration [7, 8] is not directly suitable for our type of students, both in style and in content.

However, paying respect to tradition, integration succeeded well by following a step by step approach in the master course “capita selecta”. In this course we introduced an explicit open end: a joker/trump/bonus theme, where students can pick a theme and kind of double their points on this theme (if they perform will in the extra question and/or assignment). This was an excellent opportunity to let the eager students do limited experiments with the research topic.

The actual steps performed were

- Introduce functional programming as a topic (necessary to be able to understand Lava)
- Encourage (better) students to take Lava as a joker

- Let a student 'massage' the subject in his master thesis
- Introduce Lava as a topic, and do some more 'massaging'
- Make a full course dedicated to the topic

Both the experiments by a broader group and the massaging by a smaller group were vital in preparation for the full course offered to every student. It also helped us in formulating our research topic more precisely, especially for newcomers to the subject.

8. Conclusion

This paper presented a number of design decisions in reforming the ICT-component of a multi-disciplinary master degree in Engineering Technology. Essentials keys in the success of the new program are focus through relative small but ambitious courses with a well defined learning outcome, emphasis on competences and project work through assignments and a wide spread belief in putting the stakes high.

References

[1] Dearing ROn, editor, Setting the context of higher education in Europe Retrieved from `http://www.leeds.ac.uk/educol/ncihe/r11_065.htm` at May 8, 2014 (1997)

[2] Neave, Guy, The Bologna declaration: Some of the historic dilemmas posed by the reconstruction of the community in Europe's systems of higher education., Educational Policy 17.1 (2003), 141–164.

[3] Schwarz, Stefanie, and Don F. Westerheijden, eds. Accreditation and evaluation in the European higher education area. Dordrecht: Kluwer Academic (2004).

[4] Declaration, Bologna. The European higher education area. Joint declaration of the European Ministers of Education 19 (1999).

[5] van der Wende, Marijk C. The Bologna Declaration: Enhancing the transparency and competitiveness of European higher education. Higher education in Europe 25.3 (2000), 305–310.

[6] Kölling, Michael, et al., The BlueJ system and its pedagogy., Computer Science Education 13.4 (2003), 249–268.

[7] Wolfs, D., Aerts, K., Moelans, J., Mentens, N., Design automation for cryptographic hardware using functional languages, In Proceedings of the 32nd WIC Symposium on Information Theory in the Benelux, (2011), 194–201.

[8] Wolfs, Davy; Aerts, Kris; Mentens, Nele, Design space exploration for automatically generated cryptographic hardware using functional languages, In: Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on. IEEE, (2012), 671–674.

[9] WILLIAMS, Laurie, et al, In support of pair programming in the introductory computer science course, Computer Science Education, (2002), 12.3: 197–212.

[10] KAY, Judy, et al., Problem-based learning for foundation computer science courses. Computer Science Education, (2000), 10.2: 109–128.

[11] Fincher, Sally, and Marian Petre, eds, Computer science education research, CRC Press, (2004), 239 pages.