International Journal of Engineering Education Vol. 34, No. 4, pp. 1400–1410, 2018
Printed in Great Britain

0949-149X/91 $3.00+0.00

A Computer Engineering Curriculum Model for Teaching Software Development to Bridge the Gap between Academia and Industry*

MARIA-JESÚS MARCO-GALINDO[1], JOSEP M. MARCO-SIMÓ[1] and MARC FUERTES-ALPISTE[2]
[1] Universitat Oberta de Catalunya, Dept. of Computer Science, Multimedia, and Telecommunications.
[2] Universitat Oberta de Catalunya, eLearn Center, Rambla Poblenou, 156. 08018 Barcelona, Spain.
E-mail:{mmarcog, jmarco, mfuertesal}@uoc.edu

Software development has been traditionally taught in universities starting with programming, then analysis and design phases of software engineering, and ending up with software management processes. However, this bottom-up approach (from the specific to the general software view) is precisely the opposite of the typical accepted software development processes and life cycles found in the professional field. With the aim to bridge this gap between academia and industry, a multidisciplinary team of eleven lecturers of the Universitat Oberta de Catalunya participated in a long-term research study with the aim to propose a new engineering curriculum model in the scope of software development. This paper reports on this study and demonstrates the correct adaptation of the proposed curriculum model to the design of the Bachelor in Computing Engineering within the European Higher Education Area whilst revealing in general a positive impact in the actual implementation of the innovative software development curriculum in our university in terms of academic performance and satisfaction.

Keywords: software development; curriculum model; computing engineering; higher education; industry; bachelor in computing engineering (BCE); European Higher Education Area (EHEA); Universitat Oberta de Catalunya (UOC)

1. Introduction

In the last decades, the software development (SD) landscape has dramatically changed. Human activity is software dependent and hence software production is continually evolving [1]. Knowledge on developing quality and effective software is constantly expanding and this is putting SD curricula under pressure to fit the industry needs [2]. This situation is creating new challenges in SD education to bridge the gap between what industry demands and the competences and skills of Computing Engineering graduates [1].

Traditionally, the teaching of aspects involved in the SD scope has followed a bottom-up approach, i.e., starting with programming (P), following the aspects of analysis and design of software engineering (SE)—often in parallel to database (DB) design—and ending with SD management tasks (M). As can be seen in Fig. 1, in Universitat Oberta de Catalunya (UOC), we followed this type of bottom-up curriculum model, typically found in SD curricula in Computer Science programs. The origin of this bottom-up approach can be found in the historical evolution of the SD process itself. However, currently this bottom-up approach is precisely the opposite of the typical accepted SD processes and life cycles found in the industrial arena [3, 4].

In other Computer Science scopes, teaching had also followed this bottom-up approach. For instance, the computer networks' teaching pathway started and was focused on the lower layers of the OSI model, even though the main professional interest lies in the upper layers. Nowadays, many proposals have been made to abandon the traditional bottom-up approach by taking the opposite direction based on a top-down vision (i.e. they start with the upper layers of the OSI model). This is being adopted in many university programs though often implies a change of some contents [5].

In line with the above-mentioned, current discussions in the SD scope deal with: (a) the extent to which a computer engineer should be trained in programming [6], (b) how to adapt the computer curricula to the European Higher Education Area (EHEA), and (c) what are the differential competencies and skills of a computer engineer compared to other professionals in the sector, thus fostering a reflection about how to articulate and organize teaching in the SD scope while trying to determine which obsolete aspects could be discarded and what new training needs should be included. The ultimate goal of these efforts is to set out a new curriculum model in the SD scope following a top-down approach (from the general to the specific software view), accordingly with the actual software development in the professional field.

To this end, this paper reports on the results of a long-term research study conducted by a team of

* Accepted 12 March 2018.

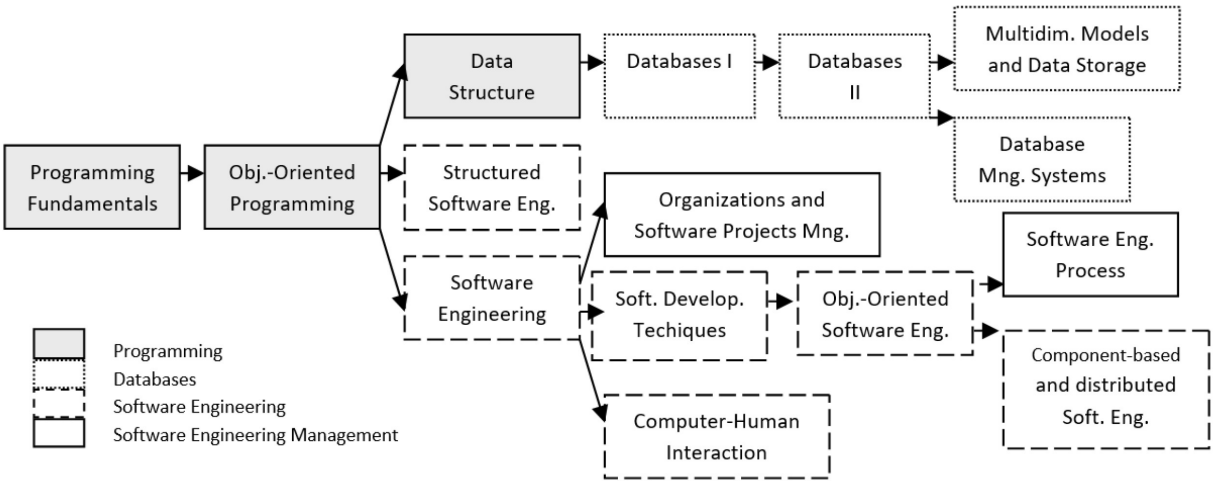

Fig. 1. Traditional bottom-up UOC's curriculum model of SD before the European Higher Education Area (EHEA) period.

eleven professors from the UOC that come from the four knowledge areas involved in the SD scope we take into account in this research (P, DB, SE and M). In 2007, with the aim to perform and submit a constructive criticism, this teaching team participated in joint discussion sessions in which they rethought the usual topics of each knowledge area while attempting to formulate a new proposal that followed the top-down approach. In 2010, the creation of the new EHEA degree programs allowed the implementation of this proposal in the new degree in Computing Engineering. Since then and throughout 6 consecutive academic years, data was collected, analyzed and discussed in terms of the impact of the new engineering curricula in the teaching of software development.

The paper is organized as follows. Section 2 presents previous work related to our study. Section 3 describes the research methodology and describes the proposal of a new curriculum model of SD at university level. Within this methodology, Section 4 shows the results of this research by a demonstration that our SD model worked well when applied to a degree design adapted to the EHEA as well as the evaluation of the actual implementation of this model in our university. We conclude the paper in Section 5 by summarizing the main ideas of this paper and outlining the next directions of our research.

2. Related work

Many authors claim that students perceive SD education as a challenging field for the high level of abstraction it implies and because it is not very much connected with real objects [7]. In some cases, students find it hard to create a clear mental model of a program execution [8] due to a lack of knowledge of initial programming strategies [9] and procedural programming skills [10]. To overcome these barriers, there is a strong need to link the SD curriculum with real life competences and skills [11].

Some authors have proposed alternatives to address these challenges and particularly in the programming subjects, which are related to the requirements of logical thinking and abstract concepts. One example is the use of problem-based learning approaches to increase students' motivation and involvement [7]. For instance, problem-based learning has been used for teaching Scrum-based software development methods [12] and Object-oriented programming [13], as well as in combination with game-based learning to teach Computational Logics [14], database management, programming, and data structure [15]. Another example is the design of meaningful learning activities through the use of microworlds [16]. However, these proposals do not solve the problem of the traditional bottom-up approach used in programming subjects.

The curriculum model we propose in this paper has points of connection with the contributions of Meyer for the teaching of SD, who back in 1993 [17] assumed the principles of inverted curricula or progressive opening of black boxes [18] applied to the field of electrical engineering education. In this case, instead of adopting a top-down vision, Meyer promoted an outside vision which is based on introducing the student first as a software consumer-user who moves towards a producer-developer role. His proposals were based on giving total access to the exclusive and unambiguous object-oriented use, with design by contract as a strategy and supported by a powerful library environment. Regarding low-level concepts and skills, students take care of all the aspects required by the SD, ranging from a broad approach to more specific details [19]. However, it seems that these proposals

have been mainly limited to the initial programming subjects [20] even though Meyer already presented some interesting guidelines for a complete curriculum design [21]. Regardless of its actual final scope, Meyer's reasons for the need to deeply rethink the curriculum organization of SD teaching are very close to our views.

3. Method

The context of this research study is found in the higher education field and particularly in the Computing Engineering degrees. The study was conducted by a multidisciplinary team of eleven lecturers of the UOC who proposed an innovative engineering curriculum model to bridge the gap between academia and industry in the scope of software development. Both the human factor and the complexity of the study made appropriate to consider an interpretative perspective of the research [22], motivated by the need to deeply understand the observed reality rather than control it [23, 24].

From this interpretative perspective, we selected a research methodology that supports the creation and evaluation of IT solutions applicable to real organizational needs [25]. To this end, we considered the Design Science Research Methodology (DSRM) [26], which proposes a process with several activities in a nominal sequence, including (i) the identification of the problem and motivation, (ii) definition of the objectives for a solution, (iii) the design and development (of the artifact, such as a model, system, etc.) that solve the observed problem, (iv) the demonstration, and (v) the evaluation of the artifact to measure to what extend the results obtained are in line with the research objectives.

The sequence of these five activities fit quite well the achievement of the research goals of our study. Following the DSRM methodology, next we describe the first three activities while the results of the last two activities will be shown and discussed in Section 4.

3.1 Identification of the problem and motivation

The main purpose of this study is to conduct research in the context of Computing Engineering degrees within the EHEA in order to investigate if it is possible to implement the teaching of SD from a top-down approach, starting with the SE analysis and design phases before undertaking the programming phase. This purpose turned into the research question: can the SD teaching get close to the widely accepted SD life cycles found in the professional field?

Following an interpretative view of the problem and motivated by the research question posed, we first performed a literature review (see Section 2) in order to understand the different views of the problem found in the literature, which motivated the research process to be carried out. Then, we searched for existing engineering curriculum proposals in our geographical context of higher education, namely the Universitat Politècnica de Catalunya, Universidad Politécnica de Madrid, Universidad Politécnica de Valencia and Universidad Carlos III de Madrid, though without any result in line with our study was found. We continued our search in the international higher education context and found some representative SE curriculum [27], which proposed to start with SE. However, from a deeper analysis we observed that this curriculum did not only include programming, but actually started with programing.

Overall, we could not find sound approaches similar to our study except for some theoretical proposals presented in Spanish conferences on Education. However, these were addressed from a different perspective or tackled the problem only partially. For instance, they identified the disconnection between the academic areas of programming and software engineering, resulting in knowledge gaps and poor engagement when learning new software design techniques, since the students claimed these techniques to be irrelevant when facing programming tasks. This view is in line with the purpose of our study.

Apart from these approaches, we detected a number of criticisms about the traditional SD curriculum organization from informal discussions and comments in both the UOC and in other universities within our geographical context. We summarize them next:

1. Traditionally, students do not have a complete vision of the SD scope until they arrive to the late stages of their studies. Hence it is necessary to provide a global and top-down vision of the SD from the very beginning. A proposed solution was the creation of a SD reference framework that students could learn from at the beginning and during their studies.
2. Several reasons made inappropriate to drastically reverse the SD curriculum from a bottom-up to a top-down approach. Among them, there was a tight curriculum dependency between the involved knowledge areas in SD and other areas of the same degree, the expectations to rapidly train programmers to meet the needs of the job market, the wrong perceptions of the actual student's skills, and the own doubts of the teaching teams about the feasibility of reconsidering the entire curriculum model for SD. A less drastic approach was proposed by introdu-

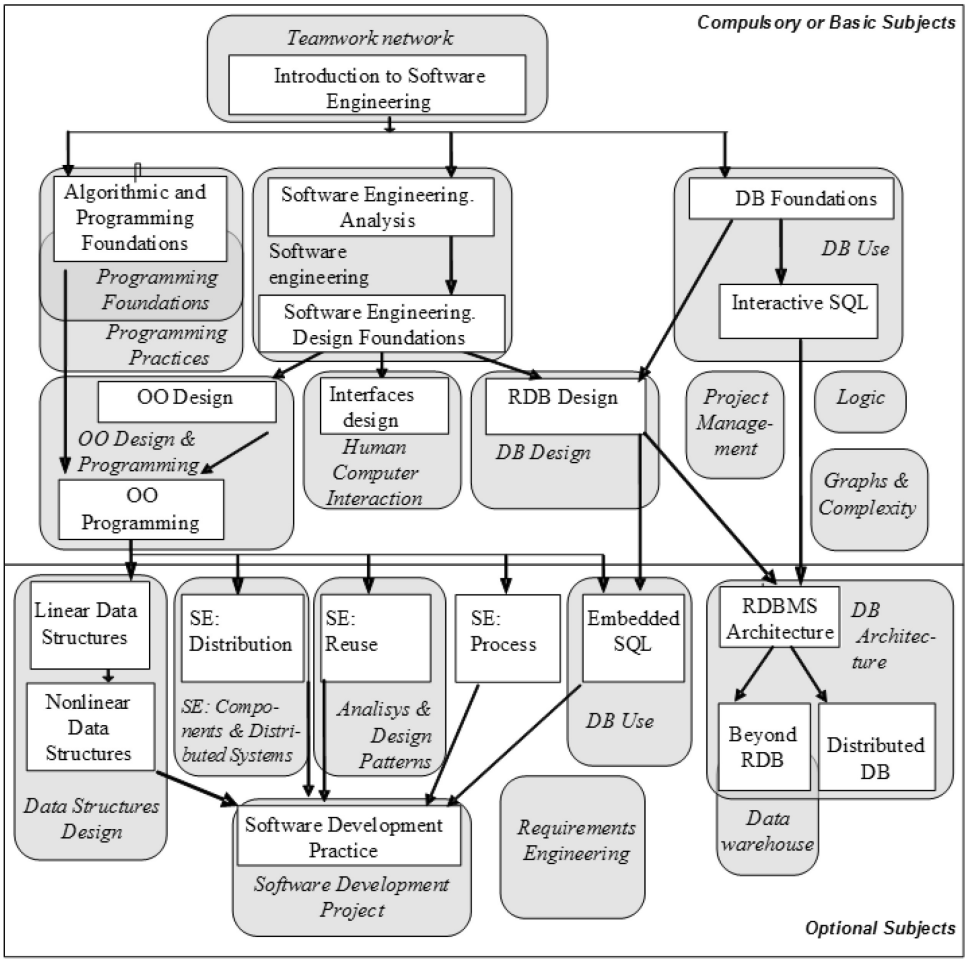

Fig. 2. Model proposal of 2007 (white cells, units of learning content) vs. 2009 (shaded cells, BCE subjects).

cing SE and P in parallel rather than starting with SE before P in the first place. This would allow for reinforcing and insisting on common aspects between SE and P whilst minimizing content overlapping.

3. In order to approach the academic view to that of the industry, it was proposed to reduce the time devoted to those learning contents that are less relevant for the SD profession as well as rationalize the rest of contents by avoiding redundancies and introducing the relevant SD aspects.
4. The Computing Engineering degrees include several knowledge areas (computer networks, operating systems, etc.) different from those that strictly constitute the SD scope, which can also contribute to improve and enhance SD teaching. However, the inclusion of further knowledge areas in our study would require reconsidering the learning contents of these areas as well as overall a complex strategy to be operative.

Finally, we reflected in our approach the reasons to prevent attempts to propose new SD curriculum models while criticizing the feasibility of our proposal.

3.2 Define the objectives for a solution

The above reported discussions and reflections by our participating teaching team set out a number of requirements to be met when building a new curriculum model for the effective SD teaching based on a top-down approach. Next, we formulate these requirements in terms of research objectives:

- O1: Provide a global vision of the SD scope at the beginning of the studies.
- O2: Introduce SE and P in parallel.
- O3: Optimize the SD learning contents.
- O4: Other knowledge areas will contribute to the SD curriculum.

3.3 Design and development

From our interpretative approach to the problem and objectives set out previously, we started in 2007 the design of the new Bachelor in Computing Engineering (BCE) degree at the UOC. The same

Table 1. Learning goals and skills to be achieved for each primitive unit of learning content.

Units of content	Skills
Introduction to Software Engineering	• Phases/Stages of the Software Development Life Cycle (SDLC) • Software programming paradigms (structured, object oriented) • Basics of algorithmic (control structures, elementary types, in/out) • Programming process (coding, debugging, testing)
Algorithmic and Programming Foundations	• Data structures: Array, record • Direct access, sequential search • Code efficiency
Software Engineering. Analysis	• Object Oriented features • Unified Process • Requirement elicitation and analysis (UML and OCL) • Functional testing • Persistent data (UML. Data modelling)
Software Engineering. Design Foundations	• Use case design • Architectural design • UML
Database Foundations	• Management information principles and historical evolution • Data Base Management Systems functionalities • Relational databases
Object Oriented Design	• Adjustment to a programming language
Object Oriented Programming	• Object Oriented programming motivation • Object Oriented Code. Java tools. • Unit testing, integration testing • Interfaces
Interfaces Design	• User interface design
Relational Database Design	• Programming Logic and design • Standardization • Physical design (indexes, tuning)
Interactive SQL	• Database creation and table, queries and DB changes (insertion, delete, update) • Views, triggers, stored procedures, transactions, authorizations and roles
Embedded SQL	• JDBC and SQL-J
Linear Data Structures	• Introduction to ADT • Study of efficiency • Sequences: stack, queue and list • Priority queue • ADT libraries in Java
Nonlinear Data Structures	• Recursive programs • Three, functions and sets, relations and graphs • Design of complex ADT • ADT libraries in Java
Relational Database Management Systems Architecture	• View management • Security Management • Query optimization • Concurrency control and recovery in Data Base Systems
Beyond Relational Databases	• DB OO and DB Object-relational • Data warehouse (multidimensional, FIC, OLAP) • Semi structured data and XML
Distributed Databases	• Distributed database access • Distributed architectures BDMS
SE: Distribution	• Specification (RM-ODP) • Distributed architectures • components engineering (UML association) • Distributed design OO (UML association)
SE: Reuse	• Principles of reuse • Patterns and components • Other Reuse techniques
SE: Process	• Quality • Software configuration management • Maintenance • Metrics
Software Development Practice	• A practical software development case

teaching team of eleven lecturers also participated in the design of the new subjects of this degree, considering all the discussions and reflections conducted until then.

Figure 2 shows the initial SD curriculum model proposal of 2007 (white cells with primitive units of learning content) in contrast to the resulting SD subjects of the BCE whose design was eventually completed in 2009. The BCE subjects are depicted as containers of the units of learning content in order to represent how the BCE design covered our 2007 model proposal as to learning contents. In addition, the learning goals and skills for each unit of learning content are summarized in Table 1.

4. Results and discussion

Following the DSRM methodology (see previous section) we continue in this section with the rest of the activities related to report on the results of the research conducted so far.

4.1 Demonstration

In this activity we discuss on and demonstrate that our SD model proposed in 2007 (made of primitive units of learning content) worked well when applied to the BCE design within the EHEA in 2009. The research objectives formulated in Section 3.2 will drive the following discussion.

O1: Provide a global vision of the SD scope at the beginning of the studies. The primitive unit of learning content ''Introduction to Software Development'' became the targeted reference framework and introduction where the SD scope was globally presented. In the final BCE design, this unit was included in the subject ''Teamwork Network'' (TN), which is being compulsory in the first academic semester. This subject introduces first the different knowledge areas of the BCE, and in particular the areas of SE, DB, P and M. Then, it provides a solid reference framework for students to learn and have a global vision of the SD scope at the very beginning of their BCE studies.

O2: Introduce SE and P in parallel. The primitive unit of content ''Algorithmic and Programming Foundations'', which introduced P was split into two subjects in the BCE design: First, ''Programming Foundations'' introduced algorithmics and provided a first learning contact with an actual programming language. Second, ''Programming Practices'' tackled the skills of programming techniques (e.g., use of an integrated development environment and initial hands-on experiences in the code-compile-test programming cycle). This separation of concerns into two subjects (12 academic credits in all) allowed for responding some criticisms towards the traditional SD curricula (e.g., lack of time for students to tackle the full learning of P while getting mature knowledge as to algorithmics and programming mechanisms). The resulting curriculum map showed the parallelization of the first subjects of P and SE, which led to a more natural connection between both areas, thus eventually dealing with another strong criticism towards the traditional SD curricula (see Section 3.1).

O3: Optimize the SD learning contents. Following our 2007 proposal, the BCE design discarded or reduced to the minimum certain outdated topics in the professional praxis, such as a) SE based on structured analysis and design in favor of SE based on Object Oriented (OO) principles, and b) old-fashioned pre-relational DB aspects. Similarly, the BCE design eliminated certain overlapping contents as pointed out in our 2007 proposal: a) the design of the data models which was repeated in both SE and DB, and b) the OO principles and UML notation included in both SE and P. From the start of the new BCE degree, these contents are being explained only once from a general perspective and then applied to either SE or P accordingly.

Eventually, the resulting BCE design presented a curriculum arrangement closer to the actual SD logics of the professional field. In this sense, parallel to the introduction of SE and P, the subjects Requirements Engineering and Project Management are introduced. These two subjects are not exclusively related with SD, but they introduce the agile methodologies that are essential today in the SD professional field. Moreover, after the introduction of SE, the aspects of the interface design were encompassed in the subject ''Human Computer Interaction'' (HCI) being well integrated in terms of SD process, thus allowing for responding further criticisms of the traditional curriculum organization. However, HCI contents are cross-curricula as they are applied further to the SD scope, such as the device interface design, thus complicating their full integration.

O4: Other knowledge areas will contribute to the SD teaching. This objective was not fully realized since in spite of setting up a steering committee to centralize and make the main decisions of the BCE design, the lecturers of all the involved knowledge areas participated in the BCE design from different separated sub-committees, thus hindering the communication between the SD and the rest of scopes. A representative issue is the specific arrangement of the conceptual subject ''Logic'' (L) in parallel with ''Programming Practices'' and ''Software Engineering'' instead of preceding them as pre-required conceptual knowledge. Moreover, following L, there is ''Graphs and Complexity'', which is another conceptual-purposed subject that coincides in time with the application-purposed subject ''OO Design

and Programming'' instead of preceding it. Finally, certain units of content, such as ''SE: Process'' (see Fig. 2) were not eventually covered in the BCE design. Overall, however, the BCE design conveniently responded to the different reasons to prevent attempts to propose a new SD curriculum model as well as to criticisms about the feasibility of our proposal (see Section 3).

4.2 Evaluation

In this final activity of the DSRM methodology, we present a quantitative and qualitative evaluation of our SD curriculum model proposed in Section 3.3 in terms of the impact of the actual implementation of the BCE design since the introduction of the BCE degree in 2010.

It is out of the scope of this paper to present and analyze the full implementation of each and every BCE subject in detail as well as to describe the level of adaptation of the subjects according to the BCE design. Instead, from the results obtained in the first semesters after introducing the new BCE degree, certain details of these subjects whose implementation has been significantly deviated from the BCE design will be revealed. In the next section (Conclusions) these deviations will be duly informed.

4.2.1 Quantitative evaluation

Table 2 shows the comparative results in terms of teaching impact in academic performance and satisfaction of the different subjects related to the SD scope between the BCE degree within the EHEA (implementing our SD curriculum model) and the equivalent pre-EHEA Computer Technical Engineering (CTE) degree (without implementing our SD curriculum model). Academic performance refers to the percentage of students who pass a subject out of those enrolled in the course. Satisfaction refers to the percentage of each response to a Likert-scale questionnaire (''very satisfied'', ''satisfied'', ''neither'', ''dissatisfied'', ''very dissatisfied'') out of the total number of responses obtained from students who usually assess the subject at the end of course. The data was collected in the UOC during the period between February 2010 (introduction of the BCE within the EHEA) and February 2017, and throughout 14 consecutive academic semesters. The statistical mean for Performance and Satisfaction presented from pre-EHEA corresponds to all the historical data available before February 2010. Although diminishing, the CTE degree was still running in parallel with the BCE until its total extinction in 2016. Next, we show the most relevant results of the comparative data analysis by using basic descriptive statistics (mean).

The introductory subjects of P, ''Programming Foundations'' (PF) and ''Programming Practices'' (PP), have the lowest performance of the BCE though in line with the equivalent subjects of the pre-EHEA degree for (about 35%). Indeed, the first-degree subjects enrolled by newly admitted students at the university usually result in high drop-out rates [3]. On the other hand, PF's satisfaction in both EHEA and pre-EHEA is as high as 74% though PP's shows a negative impact in both performance (45.85% from 53.63%) and above all satisfaction (52.80% from 70.55%). This decrease in both indicators actually responds to many difficulties informed by the teaching staff so as to design effective learning activities that tackle the most complex contents of these subjects (e.g., recursivity, complexity and algorithmic efficiency) while reporting high levels of knowledge immatureness of the newly admitted students. It is worth noticing that

Table 2. Summary of the results of academic performance and satisfaction for the SD subjects within the EHEA (implementing our model) and the equivalent SD subjects within the pre-EHEA (without our model)

SD Subjects of BCE (EHEA)	EHEA Performance (mean) %	pre-EHEA Performance (mean) %	EHEA Satisfaction (mean) %	pre-EHEA Satisfaction (mean) %
Teamwork Network (TN)	72.21	65.90	76.65	74.19
Programming Foundations (PF)	36.30	34.62	73.70	74.40
Programming Practices (PP)	45.86	53.63	52.80	70.55
Software Engineering (SE)	65.79	69.10	78.97	64.36
OO Design & Programming (OODP)	65.09	53.63	73.51	70.55
Human Computer Interaction (HCI)	71.24	64.20	65.82	49.24
DB Use (DBU)	59.07	69.73	78.75	81.88
DB Design (DBD)	62.43	75.90	79.73	87.50
DB Architecture (DBA)	82.26	69.75	79.58	74.93
Data Structures Design (DSD)	71.40	35.47	72.69	58.16
SE Component & Distributed Systems (SECDB)	72.42	77.42	79.21	51.93
Analysis and Design Patterns (ADP)	76.75	77.42	81.38	51.93
Software Development Project (SDP)	93. 06	82.03	N/A	61.93
Requirements Engineering (RE)	85.59	N/A	84.67	N/A
Project Management (PM)	72.25	64.29	60.03	64.35
Logic (L)	49.24	36.36	78.05	72.45
Graphs & Complexity (GC)	54.39	51.14	66.48	46.05

these complex contents were neither included in our model proposal of 2007 nor in the BCE design of 2009, but in the BCE implementation these contents had to be eventually incorporated into these subjects as they were the only compulsory subjects in the P area.

The following subject in the P area is "Object-Oriented Design and Programming" (OODP), which has an acceptable level of performance and satisfaction (65.09% and 73.51% respectively) and higher than the pre-EHEA degree (53.63% and 70.55%, respectively). This positive impact chiefly in performance can be justified because students are in later stages of their studies as well as this subject's demands are in line with the training received and skills acquired in previous subjects of P and SE. Similarly, the next subject in P, "Data Structures Design" (DSD), obtained a high level of performance (71.60%), doubling the same indicator of the equivalent subject in the pre-EHEA degree. In terms of satisfaction, the improvement is also significant (72.69% vs. 58.16%). This highly positive impact responds to the same criteria as OODP, since the previous subjects in P sufficed to face the challenging contents of this subject, such as the design of data structures, the development of algorithms and the programming issues related to them. Furthermore, DSD represents a quantum leap in terms of knowledge level and focus with respect to previous subjects in P (OODP and PP), thus conferring even more relevance to this result.

Regarding the SE area, as above mentioned the subject "Software Engineering" (SE) can be studied in parallel with the subjects of the P area during the first academic semester of the EHEA degree. The results of performance and satisfaction are very acceptable (65.79% and 78.97%, respectively) and similar to the equivalent subject in the pre-EHEA degree with a notable positive impact in satisfaction. Therefore, we can conclude that anticipating SE in the EHEA degree has not been counterproductive from the SD teaching view. Similarly, the other subjects in the SE area, "Software Engineering of Components and Distributed Systems" (SECDS) and "Analysis and Design Patterns" (ADP) show good results on performance and above all satisfaction (about 80%). These results are similar to the equivalent subjects of the pre-EHEA in terms of performance while showing a significant positive impact in satisfaction (30 percentage points of increase). This is interpreted as students in the EHEA feel comfortable and competent with their previous knowledge on SE when they study these advanced subjects.

In line with the SE knowledge area, the subject in the DB area "DB Use" (DBU) also introduces the databases in parallel with the subjects of P and SE areas. However, the results are not so good (59.07% of performance and 78.75% of satisfaction) showing a negative impact compared to the equivalent subjects of the pre-EHEA degree, though not significant. The following subject in the DB area "DB Design" (DBD) points out similar trends as for performance and satisfaction (62.43% and 79.73%, respectively) showing again a negative impact of both indicators with respect to the equivalent subjects of pre-EHEA (75.90% and 87.50%, respectively). On the other hand, the following and last subject in this area, "DB Architecture" (DBA) shows a change of trend with good results and a positive impact of both satisfaction (79.58%) and above all performance (82.26%) with respect to the equivalent subjects of the pre-EHEA degree. We can conclude that considering the DB area as a whole, the change of orientation of the subjects was performed successfully, though partially, since the positive effects are not noticed until late stages of this area.

4.2.2 Qualitative evaluation

In addition to the previous quantitative evaluation, we also conducted a qualitative analysis of the actual implementation of our curriculum model in the BCE degree. To this end, we collected the evaluation reports that the teaching staff perform and submit at the end of the academic semester, and throughout 14 consecutive semesters. We complemented these reports with personal interviews with the teaching staff driven by the objectives of our study.

Next, we summarize the main results with respect to the objectives formulated in Section 3.2 (see also Table 3).

O1: Provide a global vision of the SD scope at the beginning of the studies. As mentioned in Section 4.1, providing a global vision of the SD scope was included in the subject TN, which was compulsory during the first academic semesters after introducing the BCE, thus the reading and study of this specific content was mandatory. However, given that the main goal of TN was focused on acquiring cross-curricula skills (not the global vision of the SD scope), this subject has been evolving in a way that this specific content has been gradually set aside and its mandatory condition forgotten by the teaching staff in charge of the subject. Even though the BCE steering committee can force teaching staff to execute certain actions, it is more productive to seek a natural agreement by leaving the teaching staff the eventual decision to whether keep this content as mandatory. It is worth noticing that the TN teaching staff has a non-technical profile with a strong background in social sciences and in education in particular.

Table 3. Overall summary of the results regarding the general question and the research objectives

Objectives	Results
O1. Provide a global vision of the SD scope at the beginning of the studies	Global vision of SD introduced as compulsory content into the BCE first-semester "Teamwork Network" subject.
O2. Face SE at the beginning of the BCE degree	SE successfully aligned with P at the beginning of BCE. Programming becomes crucial to face other teaching scopes.
O2. Keep overall academic performance and satisfaction	Academic indicators equal or better than the pre-EHEA learning of software development.
O2. Achieve enough knowledge level of programming to face other teaching scopes.	Updated and well-built programming knowledge by separation of concerns (algorithmics and programming) as well as learning extended up to 2 subjects and 12 credits. Perceived need for more practice and application of programming knowledge.
O3. Eliminate learning content overlapping and repetitions	Overlapping significantly reduced after a rigorous revision of learning contents.
O4. Overcome the limited communication and collaboration with teachers from other teaching scopes close to SD	Found strong complexity of communication and collaboration among the teaching staff from different scopes and knowledge. Detected tight time frames to implement the subjects of BCE degree.

O2: Introduce SE and P in parallel. As already discussed in the quantitative evaluation, the introduction of SE and P subjects in parallel was managed without too many curricula issues. On one hand, the gradual completion of the SE area throughout the involved subjects was perceived as natural and sufficient by the students. On the other hand, P subjects were perceived as problematic by the students, in particular PP and DSD subjects. The reasons argued by the teaching staff were actually out of scope of our proposed curriculum model as many new students usually enroll directly in these two subjects without having studied the prerequired PF subject. The bottom line is that many newly admitted students at the UOC have some subjects validated from previous studies. In the particular case of P subjects, many students have the initial and fundamental PF subject validated from either vocational formation courses studied in the context of high school or previous Computing-related degrees studied long ago, thus their programming skills are quite low. Indeed, PP and DSD require good knowledge of algorithmics and strong skills in programming languages and techniques. Therefore, the study of PF becomes fundamental so as to have these skills updated and well-built knowledge. On top of that, the key learning contents used in the P area have turned out to be more formal than applicable, thus somehow hindering to put them into practice by the students.

O3 & O4: Optimize the SD learning contents & Other knowledge areas will contribute to the SD teaching. In general, the perception of the teaching staff on these two aspects is in line with the discussions reported in Section 4.1 related to these same objectives. On the one hand, some learning content overlapping and repetitions were eliminated while reordering the contents satisfactorily. On the other hand, a smooth interaction was not managed among the subjects involved in different scopes from SD (e.g., L, GC, RE, PM, etc.) but with connections with SD. As mentioned, the reason behind this issue is the complexity of communication and collaboration among the teaching staff of each scope as well as the tight time frames to implement the subjects in time to start the new BCE degree as scheduled, thus making uneasy the coordination among the teaching teams in charge of these subjects.

We conclude this long-term research work by stating that there is neither positive nor negative limited response to our main research question of whether the SD teaching can get close to the widely accepted life cycles found in the professional field. On one hand, the achieved knowledge level of P was reported not to be enough for students to get introduced to other teaching scopes, such as computer networks, operating systems and distributed systems, which require better programming skills than those planned in the SD scope with 12 academic credits only and with many students with P subjects validated from previous studies coursed long ago, thus having quite low programming skills. Moreover, we could not find the appropriate mechanisms to overcome the limited communication and collaboration with teachers from other teaching scopes close to SD, thus not receiving contributions from them. On the other hand, a global vision of the SD was successfully introduced in the first academic semester as one of the objectives of our model. Moreover, by parallelizing P and SE, students successfully faced SE in the first courses at

the same time of P, thus keeping P at the beginning of the curriculum as a crucial tool to face other teaching scopes with programming requirements. Finally, learning content overlapping and repetitions were eliminated. Overall, in terms of academic performance and satisfaction the actual implementation of our SD curriculum model in the BCE degree within the EHEA achieved acceptable results in line or better on average than the pre-EHEA teaching of SD.

5. Conclusions and future work

In this paper, we have presented an independent free-context engineering curriculum model for teaching SD based on primitive units of learning and the corresponding skills to acquire. This model was proposed as early as in 2007 by a teaching team of the UOC with the aim to reverse the traditional academic bottom-up approach so as to get closer to the accepted SD life cycles of the professional field based on the opposite approach (top-down). For evaluation purposes, our curriculum model was, first, applied to the BCE design within the EHEA in 2009, which demonstrated to fit well for the purpose of reorienting the knowledge areas forming the SD scope. Then, we implicitly evaluated our model by comparing the performance and satisfaction levels achieved by the SD subjects based on our model implemented in the new BCE degree (EHEA) to the equivalent SD subjects existing in the previous degrees (pre-EHEA). The comparative results reveal that most of our curriculum model had a positive impact in the teaching and learning at the UOC. However, a few subjects were reported to have a negative impact, sometimes due to certain unexpected issues or reasons, which are out of scope of our model.

To sum up, although turning upside-down the teaching of the SD scope was not completely managed, our proposed curriculum model fostered the participating teaching team to face a challenging exercise to step outside the comfort zone and to rethink many topics and stereotypes about the teaching of the SD, which were profoundly accepted in academia settings after many years of teaching SD from the same perspective and approach. Eventually, we managed to get fairly close to a real implementation of the SD inversion in the way we imagined when we proposed our model far back in 2007.

As future work we plan to conduct a qualitative study with BCE graduated students (or in their last academic semester) who have fully experienced the implementation of our model in the SD teaching in order to evaluate whether they have acquired the expected global degree skills, beyond those particular skills of each SD subject. To this end, we will design a practical use case for these students to identify and assess whether their skills match the global BCE skills. Moreover, we plan to complement this study with the analysis of the final theses carried out in the SD scope over the last years within both the EHEA and pre-EHEA so as to evaluate whether the quality of these theses has significantly increased within EHEA while identifying specific improved aspects as well as areas for improvement.

Acknowledgements. The authors acknowledge the financial support from the Faculty of Computer Science, Multimedia and Telecomunications of Universitat Oberta de Catalunya. The authors are also deeply grateful to Professor Santi Caballé of the same Faculty for his extremely productive, constructive, motivating and helpful dedication and guidance in the design of the final version of this paper.

References

1. J. Liebenberg, M. Huisman and E. Mentz, The relevance of software development education for students, *IEEE Transactions on Education*, **58**(4), 2015, pp. 242–248.
2. P. Dolog, L. Thomsen and B. Thomsen, Assessing Problem-Based Learning in a Software Engineering Curriculum Using Bloom's Taxonomy and the IEEE Software Engineering Body of Knowledge, *ACM Transactions on Computing Education (TOCE)*, **16**(3), 2016, p. 9.
3. R. S. Pressman, *Software engineering: a practitioner's approach*, Mc Graw Hill, New York, 2001.
4. I. Sommerville, *Software Engineering*, Pearson Education, Harlow, 2001.
5. J. F. Kurose and K. W. Ross. *Computer networking: a top-down approach*, Pearson Education Ltd, 2012
6. N. McBride, The Death of Computing, 2007, *Retrieved from http://www.bcs.org/server.php?show=ConWebDoc.9662* (as of June 2017).
7. L. De la Fuente, A. Pardo and C. Delgado, Addressing drop-out and sustained effort issues with large practical groups using an automated delivery and assessment system, *Computers & Education*, **61**, 2013, pp. 33–42.
8. I. Milne and G. Rowe, Difficulties in Learning and Teaching Programming - Views of Students and Tutors, *Education and Information Technologies*, **7**(1), 2002, pp. 55–66.
9. A. Robins, J. Rountree and N. Rountree, Learning and teaching programming: a review and discussion, *Computer Science Education*, **13**(2), 2003, pp. 137–172.
10. G. White and M. Sivitanides, A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics, *Journal of Information Systems Education*, **13**(1), 2002, pp. 59–66.
11. A. Alarifi, M. Zarour, N. Alomar, Z. Alsahaikh and A. Mansour, SECDEP: Software engineering curricula development and evaluation process using SWEBOK, *Information and Software Technology*, **74**, 2016, pp. 114–126.
12. N. H. El-Khalili, Teaching Agile software engineering using problem-based learning, *International Journal of Information and Communication Technology Education*, **9**(3), 2013, pp. 1–12.
13. A. J. Olier, A. A. Gómez and M. F. Caro, Design and Implementation of a Teaching Tool for Introduction to object-oriented programming, *IEEE Latin America Transactions*, **15**(1), 2017, pp. 97–102.
14. F. J. Gallego-Durán, C. Villagrá-Arnedo, F. Llorens-Largo, R. Molina Carmona, PLMan: A game-based learning activity for teaching logic thinking and programming, *International Journal of Engineering Education*, **33**(2), 2017, pp. 807–815.
15. Y.-H. Hung, R.-I. Chang and C.-F. Lin, Developing Computer Science Learning System with Hybrid Instructional

Method, *International Journal of Engineering Education*, **32**(2), 2016, pp. 995–1006.
16. F. Djelil, A. Albouy-Kissi, B. Albouy-Kissi, E. Sánchez and J. M. Lavest, Microworlds for learning object-oriented programming: considerations from research to practice, *Journal of Interactive Learning Research*, **27**(3), 2016, pp. 247–266.
17. B. Meyer, Towards an object-oriented curriculum, *Journal of Object-Oriented Programming*, **6**(2), 1993, pp. 76–81.
18. B. Cohen, The Inverted Curriculum, Report, *National Economic Development Council*, London. 1991.
19. B. Meyer, The outside-In Method of Teaching Introductory Programming in M. Broy and A. V. Zamulin (eds), Perspectives of System Informatics, *Lecture Notes in Computer Science*, vol 2890, Springer, Berlin, Heidelberg. 2003.
20. M. Pedroni and B. Meyer, The inverted curriculum in practice, *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pp. 481.
21. B. Meyer, Software Engineering in the Academia, *IEEE Computer*, **43**(5), 2001, pp. 28–35.
22. W. J. Orlikowski and J. J. Baroudi, Studying information technology in organizations: Research approaches and assumptions, *Information Systems Research*, 2, 1991, pp. 1–28.
23. J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*, Sage, 2013.
24. L. Cohen, L. Manion and K. Morrison, *Research Methods in Education*, Routledge, 2007.
25. V. K. Vaishnavi and B. Kuechler, Design research in information systems, *MIS Quarterly*, **28**(1), 2004, pp. 75–105.
26. K. Peffers, T. Tuunanen, M. A. Rothenberger and S. Chatterjee, A design science research methodology for information systems research, *Journal of Management Information Systems*, **24**, 2007, pp. 45–77.
27. The Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery (ACM/IEEE-SE). *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, ACM, New York, 2004.

Maria-Jesús Marco-Galindo is an Associate Professor at the Universitat Oberta de Catalunya (UOC) since 1999. She had also been part-time Assistant Professor in the Universitat Politècnica de Catalunya (UPC) two years before. She holds a PhD in Education and ICT (eLearning) at the UOC, a MSc in Information Society at the UOC and an MBA at the UPC and she also received her BS degree in Computer Science at the UPC. From 2001 to 2007 she has been Academic Director of Computer Engineering (CTE) degree at the UOC. In addition to her research in the field of IT curriculum design and didactics of programming learning, she is also focused on the study of the teaching professional competencies and soft-skills in ICT curriculums, with special interest in written ICT professional communications.

Josep M. Marco-Simó is an Associate Professor at the Universitat Oberta de Catalunya (UOC) since 2001. He holds a PhD and a MSc in Information Society (UOC) and he received his BS degree in Computer Science at the Universitat Politècnica de Catalunya (UPC). From 2003 to 2015 he has been Academic Director of Computer Technical Engineering (CTE) degree at the UOC. In addition to his research in the field of IT curriculum design, he is also focused in IT management field with special interest in the study of the provision processes of IT.

Marc Fuertes-Alpiste is a researcher at the eLearn Center of the Universitat Oberta de Catalunya (UOC). His research interests are mainly devoted to the study of teaching and learning processes with digital technologies. He has a Bachelor in Educational Science and received his PhD in Educational Multimedia from the Universitat de Barcelona in 2011. He has authored papers in conferences, book chapters and articles in indexed journals at both national and international level.