Design of a (yet another?) DevOps course

Alfredo Capozucca, Nicolas Guelfi, and Benoît Ries

University of Luxembourg, Faculty of Science, Technology and Communication,
Maison du Nombre, 6, Avenue de la Fonte, L-4364 Esch-sur-Alzette, Luxembourg
{alfredo.capozucca,nicolas.guelfi,benoit.ries}@uni.lu

Abstract. DevOps have received marginal attention inside the higher education level curricula despite of its boom in the industrial sector. This paper presents the design of an academic master-level course aimed at DevOps. The proposed design is based on earlier experiences in teaching DevOps-related topics. The specification of the course design is provided using the SWEBOK Guide and Bloom's taxonomy to enhance the quality of the course design specification, and ease its assessment once delivered.

Keywords: Software Engineering · Education · Course design.

1 Introduction

A recent study about emerging jobs in the U.S. [1] has found out not only that tech-focused jobs are leading the trend, but also that people holding one of the top emerging jobs, five years ago they were working as software engineers. This provides evidence about the key role played by the software engineering area regarding opportunities in the U.S. labour market.

Actually, this evidence becomes even more important when considering that some of today's top emerging jobs did not exist five years ago. Therefore, teaching software engineering at the higher education level is a must to form people with a relevant set of skills that would allow them to chase the most exciting job opportunities (which today, might not yet exist).

In the large spectrum covered by the software engineering field, agile methods have gained particular attention due to their widespread use on the industrial sector [2]. Moreover, special emphasis is currently being given to DevOps [1] initiatives as a means to ease the achievement of certain agile's principles, in particular those related to iterative development, testing and team collaboration.

Since there is not official standard definition for DevOps[3], there exist many of them out there [4]. However, it can be stated with certain level of certainty that the aim of DevOps is to facilitate the interaction between Development and Operations to shrink the time since a modification is made by a developer until it makes into production without sacrificing quality.

Contrary to agile methods, which have gained their own place inside the higher education level curricula, DevOps-related subjects have received marginal

[1] Abbreviation for the interactions between Dev (Development) and Ops (Operations).

attention [5]. Thus, there is a clear need to enhance computer science curricula with content oriented to DevOps such that graduates can be better prepared to tackle the industrial sector needs.

The aim of this paper is to present the design of a master-level [2] course aimed at DevOps. The design of this course is based on earlier experiences in teaching topics associated with DevOps, but from a different viewpoint. This design is described providing not only the course's objectives, expected learning outcomes, and grading system, but also how the scheduled activities allow students to reach the expected learning outcomes. The expected cognitive level to be reached by students for each learning outcome is also indicated in the course design description by means of the well-known Bloom's taxonomy. Moreover, it is also indicated the parts of the *Guide to the Software Engineering Body of Knowledge Guide (SWEBOK)* [6] that are covered by each of the course's activities. This allows the reader to have a vision of the coverage offered by the course regarding such standard. This way of providing the description of a course may be also considered as a contribution brought by the paper.

The reader can find information about the SWEBOK and Bloom's taxonomy in Section 2, where it is also explained the course's context. Section 3 contains information about the former version of the course and its origins, whereas Section 4 reports the teaching experiences for such a version. The description of the new course design can be found in Section 5. The paper closes with some discussions about this new proposed design (Section 6).

2 Context and background

2.1 The MiCS

Started in September 2010, the *Master in Informatics and Computer Science (MiCS)* [7] is a 2 years full-time programme (120 ECTS [3]) offered at the University of Luxembourg [4] (UL).

The MiCS's objectives are to introduce students to state-of-the-art computing knowledge in modern and relevant fields, as well as laying the groundwork for either working in high-level industry-oriented environment or continuing PhD studies.

The MiCS courses, which are fully taught in English, are organised in four semesters (S1, S2, S3 and S4). In S1 courses are common to all students as the aim is twofold: to provide a solid foundation on computer science and act as orientation for the courses to pursue in the next semesters. It is in S2 and S3 when students select courses based on available *profiles*.

Each profile is a set of *required* and *elective* courses related to a particular field. A student accomplishes a profile once he has validated all *required* courses. Notice that this distinction between required and elective courses is what allows

[2] In this paper "master-level" and "graduate" terms are used interchangeably.
[3] European Credit Transfer and Accumulation System.
[4] Created in 2003 and characterised by its multilingual and intercultural environment.

students to have the chance of following multiple profiles, if desired. It is necessary (but not sufficient) to accomplish at least one profile to obtain the degree. Thus, the way the programme is organised lets students decide their own path through the degree, which is in line with the student-centred education principle promoted by the Bologna process.

Today, there are five available profiles: Adaptive Computing, Communication Systems, Information Security, Intelligent Systems, and Reliable Software Systems. The list of profiles is regularly assessed according to student, professor or industry needs. The course described in this paper belongs to the *Reliable Software Systems* profile and it is taught in S3.

The official MiCS programme description [5] states that, regardless the chosen profile, students who have validated (at least one) profile and collected the 120 ECTS would be able to (sic): *master general and specific topics in computer science, build bridges between several computer science subjects via different profiles, demonstrate a broad understanding of both fundamental and specialised areas of information technologies, stay abreast of the fast technological changes in the rapidly evolving IT sector, work effectively in multinational teams being exposed to the culture diversity of the very international master studies, tackle complex technical problem in IT by productively using a wide range of tools.* These learning outcomes should be key points to be considered when designing a MiCS's course.

2.2 SWEBOK Guide and Bloom's levels

Today, there is not (in the scientific community) common agreement about the definition of DevOps [4] nor acknowledged standard that can be used as fundamental building block for clarifying what it means and covers [3].

Despite of this lack of common agreement in the community, it is already possible to glimpse with an important level of confidence that contributions brought by DevOps drop into the software engineering field. Therefore, it makes perfect sense to rely on the *Guide to the Software Engineering Body of Knowledge Guide (SWEBOK)* [6] as reference to present the intended knowledge to be covered by a course that addresses software engineering concerns. After all, it was one of the objectives for which it was conceived: *"to provide a foundation for curriculum development and for individual certification and licensing material"*. Therefore, the SWEBOK Guide's content, which is a characterisation of the generally accepted [6] software engineering knowledge, can be used as reference when designing software engineering-related curricula.

The characterisation provided by the SWEBOK Guide is organised in 15 knowledge areas (KAs), representing each of them a chapter of the Guide. This structure is aimed to scope and clarify the place of software engineering regarding

[5] See *Learning outcomes* at [7]

[6] *"Means the knowledge and practices described are applicable to most projects most of the time, and there is consensus about their value and usefulness" [6]*

other disciplines like Mathematics and Computer Science. Each KA is decomposed in topics (TPs), sometimes also named subareas. Since the aim of TPs is to ease the way readers may find references to the actual body of knowledge, they are also broken down in sub-topics (STPs). Each STP provides a short description including one or more references. In total, the SWEBOK Guide has 100 TPs and 395 STPs. For course design purposes, a coverage course description with respect to the SWEBOK Guide could fairly being done relying on TPs, only. This is exactly the SWEBOK Guide usage intention for the purposes of this paper.

There are others well known curriculum guidelines which could be used to assess the knowledge coverage of a course (or even an entire programme). The closest alternative to the SWEBOK Guide would be the *Curriculum Guidelines for Graduate Degree Programs in Software Engineering* [8] as the others target undergraduate programmes [9, 10]. However, one of the main advantages of the SWEBOK Guide is its periodic updating and revision.

While the SWEBOK Guide (or any other standard curriculum guidelines) can be used to clearly indicate the knowledge elements to be covered by a particular course, it is still necessary to specify the expected minimum level of attainment for these targeted knowledge elements. A widely used classification system for such a purpose is the Bloom's taxonomy [11] [7] corresponding to the *cognitive* domain [8]. This domain, also known as knowledge-based, is concerned with the acquired knowledge and how such a knowledge it is acquired by a learner.

In particular, the Bloom's taxonomy for the cognitive domain was used in the SWEBOK Guide version 2004 (Appendix D) [13] to determine the levels of learning a software engineering graduated with four years of experience should have on each topic. Thus, the widely usage of Bloom's taxonomy when organising levels of acquired knowledge along with its use in the Guide (version 2004) justify the use of such taxonomy in this paper.

For recalling purposes and to make the paper self-contained, a short description of the Bloom's taxonomy (adapted from [13]) for the cognitive domain is presented in Table 1.

It is worth mentioning that the levels listed and described in Table 1 are hierarchical: reaching learning objectives at a higher level depends on having attained knowledge at the lower levels. For example, a learner aimed at attaining a level L3 will need first to attain levels L1 and L2, in that order.

In this paper, the SWEBOK Guide and the Bloom's taxonomy are used to indicate the KAs/TPs covered by new proposed course's along with the expected level of attainment for each course's learning outcome, respectively.

2.3 Existing DevOps courses

As mentioned in the introduction, currently DevOps-related courses at the higher education level are the exception rather the rule. At the moment of writing this

[7] The work has been revised in [12].
[8] The other domains are *affective* and *psychomotor*.

Table 1. Bloom's Taxonomy for Cognitive domain.

Taxonomy Level	Description of Level
(L1) Knowledge	Recall data.
(L2) Comprehension	Basic understanding without knowing its full implications.
(L3) Application	Use a concept learned in the classroom in a new concrete situation.
(L4) Analysis	Structure information such that its organisational structure may be understood.
(L5) Synthesis	Rely on previous knowledge to produce new knowledge.
(L6) Evaluation	Make judgments about ideas using concrete and valid evidence.

paper, only four courses have been found matching the following criterion [9]: course targeting graduate students, delivered by an academic institution in the context of a programme oriented to either software engineering, computer science or informatics.

The first of these found courses corresponds to the one delivered by Len Bass at Carnegie Mellon University (USA) since 2016 [14]. Its title is *DevOps: Modern Deployment*, it lasts one semester and counts for 5 ECTS. The course covers both theoretical and practical aspects: beside regular lectures, students have to complete assignments. The course is focused on the implementation of DevOps principles from a software engineering viewpoint, only (i.e. soft-skills are not considered). The text book of reference is *DevOps: A Software Architects Perspective* [15]. Major topics covered by the course are: DevOps overview, virtualisation (virtual machines and containers), deployment pipeline (continuous integration, continuous delivery), microservice-based architecture, the cloud as platform, basics on security related to networking, and monitoring.

It is at North Carolina State University (USA) where other of the found courses is being offered. It is delivered by Christopher Parnin since (spring) 2015 [16]. The course's title is *DevOps: Modern Software Engineering Practices*, it lasts one semester, and counts for 5 ECTS. The course combines lectures with in-class workshops. Students work in teams to accomplish a project aimed at building a continuous delivery pipeline from scratch. This project is delivered in several milestones. In-class workshops are meant to ease the achievement of each project's milestone. Main topics covered in the course are virtualisation (virtual machines, provisioning, and infrastructure as code), continuous delivery (configuration, build, test, and deploy management), monitoring and analysis. This course is enclosed in graduate programme with professional orientation. That may explain large number of tools covered through the intensive technical in-class workshops.

[9] Paper's authors are pretty much sure that more courses would match this criteria by the time the paper sees the light of day.

The participation at the DevOps18 workshop [17] allowed knowing two other courses addressing DevOps-related topics: one delivered at Polytech Nice-Sophia (France) by Sebastien Mosser et al. [18], and other at DePaul University (USA) by Christopher Jones [19].

The course at Polytech Nice-Sophia, named *Introduction to Software Architecture & DevOps*, is available to students since 2015. It is offered as an optional full semester course that counts for 5 ECTS. Students work in groups over a project where they exercise the topics presented during lectures interleaved with in-class practical work. DevOps-related topics covered by the course are continuous integration, testing frameworks, and containers. The project also requires students to develop. That means, they face with real interactions problems during the execution of the project. This allows students to learn soft-skills to deal with such as problems.

It has been reported that despite of being offered as an optional course, it is close to its full capacity since it was started. This is because students are very aware of the advantages of having DevOps-related skills when applying for a job.

The course delivered at DePaul University, named *Continuous Delivery and Devops*, is a full semester course being offered since (spring) 2015. The course forms part of a programme designed to let students attend the courses while working. It counts for 5 ECTS [10] and the main topics covered are: virtualisation, cloud technologies, and deployment pipeline (i.e. continuous integration, building, deploy, and testing), and configuration management. Beside the technical topics, the course also covers non-technical aspects related to organisational transformation and economics of DevOps, team organisation, collaboration, and software development practices. The course uses as book of reference *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* [20].

3 The course

3.1 Origins

In early 2012 the paper's authors started a research project in the domain of software engineering. More precisely, the project's goal was to provide a software engineering methodology oriented to students aimed at learning software engineering. Such methodology, was supposed to come along a tool aimed at providing direct support to ease the implementation of such methodology. As result of this research project, the methodology *Messir* and the tool *Excalibur* [21] saw the light of the day by middle 2014 [11].

The software environment required to manage the *Messir* requirements elicitation process while developing prototypes and beta versions of *Excalibur* was

[10] This information could not be officially confirmed by the instructor at the writing of the article.

[11] Newer versions of the tool were released after this year, as both the methodology and the tool are under continuous improvement.

evolving proportionally to the visibility and maturity of the expected outcomes. Thus, the initial software environment made of a simple IDE, some console applications and intensive use of emails turned into a much more complex environment. Figure 1 shows the software environment produced at the moment of releasing the first official version of *Excalibur*. It is worth mentioning that such software environment is still under use and maintenance to handle the delivery of every new *Excalibur* release.

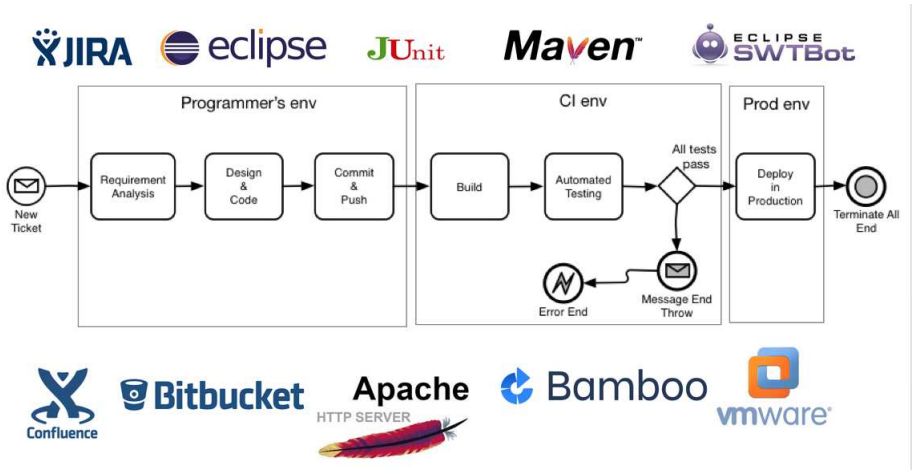

Fig. 1. The *Excalibur* deployment pipeline.

It was in early 2015 when the idea came: teach to students of the MiCS the problems faced, decisions made and lessons learnt during the process that led to the *Excalibur* software development environment. The final objective behind such idea was that at the end of the course students should be capable to specify, design and better implement the necessary means for supporting software engineering when either joining to an ongoing project, or starting one from scratch. The idea became concrete in September of the same year with the implementation of the course entitled *Software Engineering Environments*.

3.2 Initial design

The request to the MiCS programme's director to deliver such a course ended up with the assignment of a weekly 1.5 hs course in the second year's winter semester (i.e. S3), counting for 4 ECTS and a duration of 14 weeks. It was part of the initial request to make the course part of the *Reliable Software Systems* profile. The course's ECTS, place into the programme, and schedule were set based on the MiCS's curriculum. They are not supposed to change unless the profile is modified, or a restructuring of the MiCS programme takes place (something that has never happened, so far).

The course was designed as a series of lectures and practical sessions. At the beginning of the course, 2 regular lectures were used to (1) recall the fundamentals of software engineering (e.g. definition, phases of software development life cycle, etc.), (2) give a panorama about the categories of tools aimed at supporting the life cycle of a software development project [12], and (3) describe the product quality model introduced in the ISO/IEC 25010:2011 Standard [22] along with its use when assessing both the software under development and the software engineering environment that supports its development. Last, but not least (4), both the *Excalibur* tool and its associated software development environment (see Figure 1) were used to show the role played by each tool regarding the phase of the software development life cycle and the targeted quality model attributes.

Once presented the theoretical framework, each student was assigned with an individual project. The project was aimed at enhancing the *Excalibur* software engineering environment regarding one (or more) quality attribute. Thus, each student had to start analysing the current status of the given software engineering environment from the targeted quality attribute, and then perform a market analysis regarding the projects goals to choose some tools that would allow such as goals to be reached. Finally, a proof-of-concept implementation using the selected tools had to be made.

An example of a given project was to explore the use of containers as a mechanism to ease the maintenance of the different required environments (development, testing, production), while enhancing the performance compared to virtual machines.

Based on the course's ECTS, the load for an average student was stipulated in 8 weekly hours. From the third week, and until the end of the course, each student had a 1 hour individual practical work session to present to his project supervisor [13] the advances achieved with respect to the objectives set in the previous week, and the impediments faced (if any). Support to solve the faced impediments was given by the project supervisor during the practical work session (office hours were also organised in case of more time was required to deal with the impediment). After 3 practical work sessions, each student gave a talk (knows as checkpoint) to the whole class presenting the current status of his project along with a plan of activities to achieve the remaining project's objectives. That made a total of 3 delivered talks for each student.

A report evaluating the work done in the project from a qualitative viewpoint had to be also delivered along with the project's technical artifacts. This report counted for 25% of the final grade, whereas the technical artifacts counted for 50%. The 3 interleaved talks counted for 12,5% of the final grade. A final wrap-

[12] *Tools* for supporting any single task of the software development project life cycle, *workbenches* which combine in an integrated way two or more tools to cover a subpart of the software development project life cycle, and *environments* which combine tools and workbenches in order to cover the full software development project life cycle.

[13] One of the teaching staff members.

up project presentation, delivered for each student during the exam session, was the last component of the final grade (12,5%). While the report and the project's technical artifacts were only evaluated for the project supervisor, the talks and the final presentation were graded by all teaching staff members [14].

At the end of this course students were expected to attain the following learning outcomes: define the requirements of the software engineering environment required for a particular software development project (LO1); analyse and classify tools based on certain quality attributes (LO2); use, integrate and/or improve existing development tools (LO3); write a report of scientific and technical quality (LO4); and handle (i.e. plan, coordinate, and report activities) a project (LO5).

4 Facts, feedback and reflections

Three editions of the course were delivered. These editions were delivered by the same teaching staff composed of one professor and two teaching assistants. The profile of the students remained the same on each edition of the course: 70% got their undergraduate (i.e. bachelor) degree one year ago (20% were graduated at the UL, whereas 80% came from central and east Europe), very few students had industrial experience, and most of them were not used to doing projects. In general, they had good technical abilities, but poor scientific skills.

The first edition (i.e. academic year 2015-2016) resulted in 4 students passing the course, 2 failures and 1 drop out. Students' feedback collected through general discussions organised at the end of checkpoint sessions and individual interviews made during the project follow-up sessions confirmed agreement about the following facts: "hard time to get into the project's subject", "very hard work to get the project done", "high effort vs. course ECTS", and "unbalance between practical and theoretical work".

The student's claims were acknowledged by the teaching staff based on the many office hours given to students to help them to move the projects ahead. As reflection of the first edition, the teaching staff concluded that a more detailed initial project description would be provided to each student, and in particular a clear description of the technical-related objectives to be met. It was also added as requirement to provide management-related information (e.g. track weekly working hours for each working package and report them at each checkpoint). Both measures were aimed at decreasing the risk of getting students lost and/or working overtime on non-relevant tasks. Last, but not least, the teaching staff reviewed the material of the initial theoretical lectures aimed at clarifying how concerns like product under development, software engineering environment, tools, and quality attributes were related to each other, and particularly, their role on the student's projects. Obviously, the teaching staff had to invest a non-negligible time in the preparation of the second edition of the course, in particular in the project descriptions to achieve a equal level of complexity among them (very challenging task).

[14] Average of the given grades.

The second edition of the course (i.e. academic year 2016-2017) resulted in 5 students passing the course, 1 failure and 2 drops out. The student's feedback again gave as result a high agreement about the claims "very hard work to get the project done", and "high effort vs. course ECTS". Despite some punctual facts as overtime work and request for office hours (of some students) by the end of the semester, not real evidence was found by the teaching staff supporting these two claims. Extra inquires revealed that students' claims were based on comparing the requested working load of the course with respect to other programme's course with the same ECTS. Thus, the fact the course was actually requesting the maximum allowed budget for each given ECTS (i.e. 30 hours/semester for each given ECTS), plus the execution of a project as part of the requirements to pass it, were (from the student viewpoint) valid concerns when choosing and assessing the course. However, for the teaching staff, both the requested working load and project-based approach were (and still are) non-negotiable items in the design and execution of the course.

The same feedback revealed that much less effort was required by students to get into the project and understand its objectives. Moreover, a better alignment between practice and theory was achieved by the results presented by students during their checkpoints and in the final report. Therefore, the investment made by the teaching staff paid off. However, the time spent by the teaching staff on the course was over the average mainly due to: tough task of defining equally complex projects [15], and their respective supervision.

The third (and last) edition (i.e. academic year 2017-2018) was a game changer for the course's life cycle. The course started with 6 students, but, after four weeks, only 2 students remained coming to the weekly project follow-up meetings. Through informal discussions with the "survivor" students it was confirmed that students quit the course because of its (expected) workload and evaluation mechanism (continuous project-based assessment). This resulted not only in a 4-project description effort discarded, but also in a questioning process about what to do with the course.

It must be mentioned that for one of the two remaining students, the course was elective as he was formally registered in a different profile. This is an important fact to highlight as it shows that (fortunately) there are still motivated students that decide to attend a course based on the expected learning outcomes rather its exigencies. Therefore, based on the willingness to contribute in the development of such kind of students and to continue supporting the MiCS's mission, it was decided to keep delivering the course, but only after redesign it.

4.1 Objectives for the new version

The reason for redesigning the course was not only to (1) make its content more DevOps-oriented, but also to (2) achieve an organisation and execution that were more independent of the number students. While the first point was motivated

[15] The number of projects was directly proportional to the number of students registered to the course.

by the need to bring DevOps into the classroom as a first-class subject to better prepare students to modern software industry, the second one's was to minimise the impact of drops out in the course execution, as faced in its last edition. Notice that achieving a course organisation and execution that is (up to certain level) independent of the number of participants helps not only when the numbers decreases, but also increases.

It has to be recognised that the fact of redesigning the course allowed advertising it as a sort of "new course" covering a trending topic as DevOps. This, may help having more registered students, but definitively not to avoid their later drop out. Thus, the new version of the course had to be designed to favour interactivity and student engagement.

The next session describes the new design of the course aimed at attaining the objectives earlier mentioned while coping with the challenges imposed by the context: students (very often) not motivated, with weak analytical and technical skills, and not used to doing project-based courses that require regular continuous work.

5 New version

5.1 Activities & Organisation

Here it is explained what are the activities to be done by teachers and students during the course such that the objectives can be reached. Before presenting these activities along with their respective descriptions, it is worth explaining that most of the types of such as activities are taken from the course's initial design. This is because project-based teaching, blended with traditional lectures and tutoring sessions has already proved to be successful as knowledge transfer mechanism.

Project presentation: activity performed by the teaching staff at the inaugural lecture of the course. The project plays a central role in the course as it is the chosen pedagogical vehicle to let student construct knowledge and skills based on the tasks required to be performed by themselves in order to reach the project's objectives. The success of this pedagogical approach (clearly confirmed in the course's previous editions as due reported) depends on the precision and clarity of the project's objectives description. Thus, a clear presentation relying on the *Excalibur* case study is given to let students understand what is a deployment pipeline as this is the aim of the project: implement a deployment pipeline based on open source technologies that work on a unix-based OS. Any other conditions that may apply over the project (like that the product to pass through the pipeline has to be a web app) are also clearly stated. It is emphasised that the intention is to achieve a proof-of-concept deployment pipeline, so aspects like performance or reliability will not be evaluated. That, however, does not impede students to take into consideration such concerns when developing the project.

It is not part of the project the development of the product to demonstrate the working of the pipeline. This means that each group has to pay special attention when choosing the product: it has to help achieving the project's goal rather than adding complexity. The selected product would make each project unique with respect to each other, regarding the technical solution to be provided. However, the overall project's objectives and conditions are the same for each group, regardless the chosen product (i.e. a common project for all the groups). The project is done in groups, and students are free to decide about the group's composition.

Lecture: activity performed by the teaching staff once the project has been presented. Lectures are aimed at teaching the relevant concepts associated to DevOps used in the project. However, it is not the interest of the course's lectures to teach how a particular tool works, but the concepts associated to the use of such a tool, in particular highlighting the requirements that such tool aims to tackle. A total of six lectures are delivered along the semester. The first three weeks of the course are mainly aimed at lectures. The remaining ones are separated two/threes weeks each other as other project-related activities start being interleaved. The initial condensed number of lectures are aimed at presenting the DevOps theoretical background that scopes the project. They will later leave place to sessions oriented to management and follow-up of the project. The topics covered by the lectures are aligned with the project's objectives. Thus, after introduced DevOps to present its definition(s), principles, practices, its role on the software engineering life cycle, and the different roles and responsibilities (first lecture), the deployment pipeline concept (architecture, environments, tools and selection criterion) is presented (second lecture). It is also part of this lecture to introduce the notion of quality [16] and how such notion applies both to the pipeline and the product(s) that will go through it. The remaining lectures focus on the pipelines concerns, so configuration management (third lecture), build management (fourth lecture), test management (fifth), and deploy and release management (sixth lecture) are the covered topics. The project's objectives (and then, covered topics) were chosen based on the recommendations given in [5, 17], teaching staff's experience and context's constraints (i.e. course duration and assigned workload). The books given as main references to the students (and required to consult) are [15] and [20]. Other extra relevant references (but not required during the course) are [23] and [24].

Short product presentation: activity assigned to each group, and performed by one (or more) member of the group as an informal presentation. This activity is given as assignment for the second week. Each group has to present a list of possible products to be used with the deployment pipeline. The presentation's form is free, and it is followed by a open discussion. This should help each

[16] It is planned to keep using the ISO/IEC 25010:2011 Standard [22] to introduce the notion of product quality model.

group to choose the best product. The main aim of this activity, beside an early selection of the product, is to foster students interaction and engagement with the project. It is a non-graded activity.

Project follow-up session: activity performed by students and the teach staff in joint way. Each group is supervised by a member of the teaching staff (aka project supervisor). A member of the group chairs the session. The chair opens the session handling a (free) document that reports the time spent: coding, designing, using a particular tool, learning about a particular tool, and coordinating with members of the group. These times [17] have be detailed for each group member. The aim of requesting students to track these times is twofold: detect students' overtimes/downtime; and assess actual project workload (very useful for improving future editions of the course). The session continues with the report of work done, and what to do for the next session. The last part of the session is used to resolve the encountered impediments, if any. There are five follow-up sessions scheduled such that there is always one before a checkpoint. The chairing of the session is a rotating post among group's students. The supervisor grades each session based on the quality of the information provided and formulation of the encounter impediments.

Checkpoint: activity assigned to each student's group, which consists in presenting the advances of the group's project. This presentation consists in a 10-15 minutes talk where one single member of the group reports the work done by the group. There are three checkpoints, so each group's student has to deliver (at least) one presentation [18]. The first checkpoint takes place in the fifth week. There are four weeks between each consecutive checkpoint. In the first checkpoint the group is aimed at presenting the product to be used to demonstrate the functioning of the deployment pipeline. A first design of such pipeline, including candidate tools and quality attributes to be addressed must also be included. In the second checkpoint the group should present how virtualisation and provisioning tools are used to setup the different environments. It is also in this checkpoint that the presence and correct functioning of a continuous integration server (i.e. commits in a version control system end up in launching a build plan). The final checkpoint is devoted to present the final version of the deployment pipeline architecture, selected tools, and quality attributes addressed. A demo of a feedback loop [19] provided by the deployment pipeline is a must in the final checkpoint. Each checkpoint is graded by the teaching staff. The project deliverables (i.e. scripts, source code, test cases, readme file, ...) have to be submitted in the morning of the last checkpoint's day.

Report writing: activity assigned to each student's group and expected to be performed in a collaborative manner by all members of the group. The report has

[17] Tracking times taken from [14].
[18] While the conditions would allow it, groups will be made of up to 3 students.
[19] Commit, build, automated test cases execution, deploy (if all test cases have passed).

to describe the objectives of the project executed by the group, requirements, assumptions, and constraints. Special emphasis has to be given not only in the description of the proposed solution, but also in the justification of the choices led to such solution. It is expected that students write the report in a scientific manner (e.g. precise description of the problem, its context, found evidence, ...), but still providing relevant technical information about the developed solution. The report must contain between 3000 and 4000 words, and have a lesson learnt section. It has to be submitted two weeks after the official end of course.

Report reviewing: activity performed both for each teaching staff member and students. This activity consists in reviewing the report to check the accuracy of its content from the technical and scientific viewpoint according to the given report's objectives. While each teaching staff member has to review every single report, a student is expected to review only one non-authored report. A report is reviewed for every teaching member and two students, providing each of them a grade (that is differently weighted). The report's grade goes to each of its authors. Note that this activity introduces peer-assessment. Depending on how good (or bad) the student performs his review he may gain (loss) extra points to his individual grade. More details are provided below in the grading section. Each student has one week to perform the review of the report.

5.2 Grading

Students are evaluated based on the activities they perform. Except for the assessment of the project report review (where the student is evaluated individually), the other evaluations are group-based: i.e. the same grade is given to every group member. Below, it is detailed how the final grade of a particular student is computed:

- Project deliverables: 50 %
- Report: 25 % (includes peer reviewing)
- Checkpoints: 12.5 %
- Project management: 12.5 % (the capability to report the tracking indicated times, along with the working plan (done/to-do/impediments)).

The peer-reviewing grading system deserves an explanation in itself. The assessment of the report, which counts for 25%, is made by the teaching staff and two non-authored students. Below it is shown how this 25% is decomposed depending on the person's profile:

- Average(Teaching staff): 40%
- Student 1: 30 %
- Student 2: 30 %

The peer-reviewing activity made by a student is "paid" in points for his grade. Depending on the quality of the performed review, he may either win or loss points. Thus, a teaching staff member will assess the student's review to decide how many points he gets (from -30% to 30% of the report's grade).

5.3 Learning outcomes

The execution of the activities described in the previous section would let students attain the following learning outcomes: design and implement a deployment pipeline for a particular software development project (LO1); classify tools based on certain quality attributes (LO2); use, and integrate existing tools (LO3); write a report of scientific and technical quality (LO4); and plan, coordinate, and report activities in a multi-participant project (LO5). It is not surprising that several learning outcomes are the same as in course's previous edition since some type of activities were kept in the new course design.

Table 2 shows how each of the activities performed along the course contributes to the achievement of the claimed learning outcomes. Moreover, the same table describes the covered SWEBOK knowledge by each activity, allowing to get an overall idea of the coverage provided by the course with respect to such standard. Table 3 complements the mapping activity-learning outcomes with information about the cognitive level reached by each student passing the course with regard each of the claimed learning outcomes.

6 Discussion

Making the course oriented to DevOps was not a random or marketing choice. Former editions of the course have already covered some subjects related to DevOps like virtualisation, continuous integration and automated testing. The new version not only better organises these subjects, but also introduces others to show how they could be efficiently integrated to achieve a pipeline that covers continuous integration, building, testing, and deployment. Thus, the previous experiences on teaching and working (i.e. *Excalibur* case study) with these subjects provides certain guaranties about the mastering level of the subjects to be taught. Moreover, the topics covered by the course justifies the claimed MiCS's learning outcome that students once have completed the programme would *"stay abreast of the fast technological changes in the rapidly evolving IT sector"* and be able to *"tackle complex technical problem in IT by productively using a wide range of tools"*

DevOps is not only about advanced technical aspects related to the software engineering, but also about culture and organisation [14]. The project-based pedagogical methodology allows to achieve a holistic approach that encloses the three dimensions. This teaching approach, already used in the previous editions of the course, is reinforced in the new edition by requesting students to work in groups. This decision expects to enhance interaction and recreate social environments where soft skills can be developed. Due to the multicultural environment that characterises the MiCs, team work guarantees the achievement of the claimed programme's learning outcome: *"students would be able to work effectively in multinational teams being exposed to the culture diversity"*. Notice, that people with such skill are very valuable in any professional sector.

Table 2. Mapping between activities, SWEBOK knowledge areas/topics, and addressed learning outcomes.

Activity	**SWEBOK coverage**	**Addressed learning outcome**
Introduction to DevOps (Lecture)	KA1: TP1 KA8: TP1, TP2 KA11: TP2, TP3	LO1,LO3
Deployment pipeline (Lecture)	KA2: TP3 KA3: TP1 KA4: TP1, TP2 KA6: TP1 KA10: TP1	LO1, LO2, LO3
Configuration Management (Lecture)	KA6: TP1, TP2, TP3	LO1, LO3
Build Management (Lecture)	KA6: TP6	LO1, LO3
Test Management (Lecture)	KA4	LO1, LO3
Deploy and Release Management (Lecture)	KA6: TP6	LO1, LO3
Short product presentation (Talk)	KA1: TP4 KA2: TP3 KA3: TP4 KA4: TP1, TP2, TP6	LO2, LO5
Project follow-up session (Meeting)	KA1: TP5, TP6 KA11: STP1.9, TP2, TP3	LO1, LO2, LO3, LO5
Deployment pipeline implementation (out of class work)	KA2: TP3 KA3: TP3 KA6 KA10 KA15: TP1, TP4	LO1, LO2, LO3, LO5
Checkpoint (Talk)	KA7: TP2, TP3 KA11: TP2, TP3	LO1, LO2, LO3, LO5
Report writing (out of class work)	KA11: TP2, TP3	LO4, LO5
Report reviewing (out of class work)	KA11: TP3	LO4

Table 3. Bloom's level reached for each learning outcome.

Learning outcome	**Bloom's Level**
LO1	Application (L3)
LO2	Analysis (L4)
LO3	Application (L3)
LO4	Synthesis (L5)
LO5	Application (L3)

Assigning the same project to every group let the teaching staff avoid the challenge task of producing different projects but with equal complexity. This choice also helps to ensure consistency when evaluating groups' work.

Yet another advantage of working on a common project is related to the organisation and execution of the course as it is not any more dependent on the number of registered students (what was the case in former editions of the course). Based on earlier experience, a teaching staff composed of two people [20] can handle a class of up to 12 students without quality lost.

A novelty introduced in the new design is peer-assessment. This was decided to foster the academic dimension of the master programme: reviewing articles is part of the duties of a researcher. Moreover, it is believed that such activity may help enhancing student's motivation and engagement with the course.

The use of the SWEBOK and Bloom's taxonomy not only bring clarity over the specification of a course, but also, they can be used later to assess the execution of the course. Based on the executed activities and student's performance (grades, feedback) it can be judged whether the SWEBOK's TPs were covered and the learning outcomes attained as expected, respectively.

It is expected that, despite of the context-specific constraints, the design of the course presented in the paper could be taken as reference by other instructors when facing with the challenging task of teaching DevOps.

References

1. LinkedIn Economic Graph Team: LinkedIn's 2017 U.S. Emerging Jobs Report (December 2017) Available at `https://economicgraph.linkedin.com/research/LinkedIns-2017-US-Emerging-Jobs-Report`.
2. VersionOne: 11th annual state of agile report (2017) Available at `https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2`.
3. Standard, N.I.: Devops - standard for building reliable and secure systems including application build, package and deployment (2016) Available at `https://standards.ieee.org/develop/project/2675.html`.
4. Jabbari, R., bin Ali, N., Petersen, K., Tanveer, B.: What is devops?: A systematic mapping study on definitions and practices. In: Proceedings of the Scientific Workshop Proceedings of XP2016. XP '16 Workshops, New York, NY, USA, ACM (2016) 12:1–12:11
5. DevOps Educator Workshop: First devops educators workshop (November 2016) Available at `https://github.com/devopseducator/2016workshop`.
6. Society, I.C., Bourque, P., Fairley, R.E.: Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0. 3rd edn. IEEE Computer Society Press, Los Alamitos, CA, USA (2014)
7. MiCS: Master in information and computer sciences (2010) Available at `https://mics.uni.lu`.
8. Pyster, A and others: Graduate software engineering 2009 (gswe2009) curriculum guidelines for graduate degree programs in software engineering. Stevens Institute of Technology (2009)

[20] Teaching staff composition to deliver the new version of the course.

9. Joint Task Force on Computing Curricula, A.f.C.M.A., Society, I.C.: Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. ACM, New York, NY, USA (2013) 999133.
10. on Computing Curricula, T.J.T.F.: Curriculum guidelines for undergraduate degree programs in software engineering. Technical report, New York, NY, USA (2015)
11. Bloom, B.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Mackay (1956)
12. Anderson, L., Krathwohl, D., Bloom, B.: A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives. Longman (2001)
13. Society, I.C.: Guide to the software engineering body of knowledge 2004 version. SWEBOK 2004 Guide to the Software Engineering Body of Knowledge (2004)
14. Bass, L.: Devops: Modern deployment (2017) Available at `http://mse.isri.cmu.edu/software-engineering/Courses/17-611-DevOps-Modern-Deployment.html`.
15. Bass, L., Weber, I.M., Zhu, L.: DevOps : a software architect's perspective. Addison-Wesley Professional New York (2015)
16. Parnin, C.J.: Devops: Modern software engineering practices (August 2017) Available at `https://wolfware.ncsu.edu/courses/details/?sis_id=SIS:2018:1:1:CSC:519:001`.
17. DEVOPS18: First international workshop on software engineering for continuous development and new paradigms of software production and deployment (2018) Available at `https://www.laser-foundation.org/devops/2018/`.
18. Sébastien Mosser, Anne-Marie Pinna-D, P.C.G.M.: Introduction to software architecture and devops (2017) Available at `https://github.com/mosser/isa-devops`.
19. Jones, C.: Continuous delivery and devops (2017) Available at `https://www.cdm.depaul.edu/academics/pages/classinfo.aspx?Term=20182&ClassNbr=21100&fid=258484`.
20. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. 1st edn. Addison-Wesley Professional (2010)
21. Guelfi, N., Capozucca, A., Ries, B.: Website of the Messir Method and the Excalibur Environment (2014) Available at `https://messir.uni.lu`.
22. ISO/IEC: ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. (2011) ISO/IEC 13211-1.
23. Davis, J., Daniels, K.: Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale. 1st edn. O'Reilly Media, Inc. (2016)
24. Httermann, M.: DevOps for Developers. 1st edn. Apress, Berkely, CA, USA (2012)