UDC 004.41, DOI: 10.2298/CSIS0901001B

Development of a Modern Curriculum in Software Engineering at Master Level across Countries

Klaus Bothe[1], Zoran Budimac[2], Rebeca Cortazar[3], Mirjana Ivanović[2], and Hussein Zedan[4]

[1]Dept. of Informatics, Humboldt University Berlin
Unter den Linden 6, 10099 Berlin, Germany
bothe@informatik.hu-berlin.de
[2]Dept. of Mathematics and Informatics, University of Novi Sad
Trg. Dositeja Obradovića 4, 21000 Novi Sad, Serbia
{zjb, mira}@dim.uns.ac.rs
[3]Dept. of Software Engineering, University of Deusto
Apdo. 1, 48080 Bilbao, Spain
cortazar@eside.deusto.es
[4]Software Technology Research Laboratory, De Montfort University
Hawthron Building, LE1 9BH Leicester, UK
zedan@dmu.ac.uk

Abstract: A strong need for new approaches and new curricula in different disciplines in European education area still exists. It is especially the case in the field of software engineering which has traditionally been underdeveloped in some areas. The curriculum presented in this paper is oriented towards undergraduate students of informatics and engineering. The proposed approach takes into account integration trends in European educational area and requirements of the labour market. The aim of this paper is to discuss the body of knowledge that should be provided by a modern curriculum in software engineering at a master level. Also the techniques used in development and implementation of such curriculum at different universities will be described. The presented ideas are based on the experience gained in the 3 year TEMPUS[1] project "Joint MSc Curriculum in Software Engineering", which established joint master studies in software engineering. Over a three-year interval, the project managed to define a new and joint curriculum, create teaching materials and deliver the curriculum in two institutions.

Key words: Curriculum development, Software engineering education, Teaching methodologies.

---

[1] TEMPUS - Tempus is one of a number of European Community programmes designed to help the process of social and economic reform and/or development in the EU partner countries. The Tempus Programme focuses on the development of the higher education systems in these countries through cooperation with institutions from the EU Member States.

1. Introduction

Given software systems' pervasiveness in everyday life, the need for professionals who can build on modern software engineering foundations is critical. Meeting this need requires a focus on software engineering education at both bachelor and master levels [30]. Unfortunately, West Balkan countries represent rather small region for education of software engineers at the bachelor level, but there is a strong need for software engineers in software companies. Therefore the most acceptable solution is to introduce master studies in software engineering (later on: SE) at local universities.

However, development of master curriculum in software engineering is not an easy task, especially concerning methodological and pedagogical issues. Over the years the teaching of software engineering has changed only slightly. Many practitioners believe that universities are not doing a good job and many academics argue that industry does not use the latest, best technology [23]. Consequently, both industry and academia have recognized the need for adjustments in software engineering education to effectively train the future generations of software engineers. The challenge in designing a curriculum for software engineering studies is to find a way to combine formal with practical learning, technical with non-technical skills [5], [8], [12], [17]. To do this, simulation of a real-world environment at the university is needed.

The aforementioned situation in West Balkan countries and experiences gathered during a longlasting DAAD project[2] influenced the design of Joint Master Curriculum in SE (JMCSE) [4], [15], [18]. The paper outlines the philosophy, main characteristics and some of the principles and experiences of designing and implementing such a curriculum. The project successfully fulfilled all of its goals. Over a three-year period the consortium managed to define new and joint curriculum in a relatively new educational field such as software engineering; to have it adopted in three institutions (Novi Sad, Serbia; Skopje, FYR Macedonia; and Leicester, UK); and to deliver it in two countries (Novi Sad, Serbia; Skopje, FYR Macedonia).

The rest of paper is organized as follows. Section 2 describes the state in education in software engineering which influenced the creation of our curriculum. In section 3 the main goals and activities of the project are outlined. Section 4 explains in more details the development of teaching materials. Discussion on how we are trying the decrease the level of 'spoon feeding' is given in section 5. Section 6 delivers our experiences from the first two years of running the studies, while section 7 concludes the paper.

---

2 Project - "Software Engineering: Computer Science Education and Research Cooperation", support of DAAD and auspices of Stability Pact for South Eastern Europe Sponsored by Germany.

2. Current State in SE Education and Curricula Development

More and more employees in software development industry are learning nowadays how to do (some) programming aided by tools for the mass market. This creates high competition and could eventually force the real professionals to stand out. Therefore it is very important to be informed about this situation in the industry at the university level of education and to follow the real industry needs and expectations [11], [16], [22], [24], [29].

In the last several years foreign software companies have entered the West Balkan region. Soon it turned out that they needed a lot more knowledgeable and skilled professionals then they could find. Also "older" graduates had a problem of coping with the newest trends. On the other hand, universities in the region started to accept the changes coming from the newest European educational trends. This pushed universities to cooperate closely in defining and implementing common or joint curricula in different ICT domains.

Some of the most developed countries have started to require licensing [26] for software engineers. Accreditation standards also serve as a source of requirements for validating software engineering curricula design [9]. These trends have important influences and consequences for universities and push them towards making essential restructuring in their curricula. The aim of a top educational curriculum is to train people who will belong to the top tier. While teaching only the use of tools that are fashionable at a certain point of time may bring you short-term popularity among students, doing so is not necessarily the best service you can give to future professionals. What really matters is teaching them to think critically, which will accompany them throughout their careers and help them grow in this ever-changing field [7], [20], [21], [24]. The University curricula must look beyond tools to the fundamental concepts that remain for a longer period of time. Curriculum recommendations proposed by ACM [1] and IEEE [28], contain plenty recommendations and arguments on how to teach individual courses as well as on how to structure curricula depending on the size of the department, number of faculty members, and orientation of the faculty. These recommendations can always be taken as the starting point when (re)constructing a curriculum.

In order to develop a modern curriculum in software engineering [27] it is also important to have in mind the following harmonizing elements [24]:

- *principles:* lasting concepts that underlie the whole field;
- *practices:* problem-solving techniques that good professionals apply regularly;
- *applications:* areas in which the principles and practices find their best expression;
- *tools:* state-of-the-art products that facilitate the application of principles and practices;

- *mathematics:* the formal basis that makes it possible to understand everything else.

On the other hand, students require not only excellence in technical expertise but also social competence [14]. Typically, software engineers are leading or are involved in team projects, which are often distributed, mobile, and in which members have diverse skills, different backgrounds, and may speak different languages.

Any team of teachers devoted to the curriculum development in software engineering has to have in mind the above-mentioned elements and principles as a starting point in thinking of courses, technologies/tools, methodology and pedagogy.

3. Joint Master Curriculum Development and Content

From September 2004 until August 2007 we were involved in the process of development and implementation of "Joint MSc Curriculum in Software Engineering" (JMCSE) under Tempus grant CD-JEP-18035-2003. Three institutions from Serbia, one from Macedonia, three from the European Union (EU), and two individual experts (from Bulgaria and Romania) were members of the project consortium: Humboldt University, Berlin, Germany (as Grant-holder); Deusto University, Bilbao, Spain; De Montfort University, Leicester, UK; University of Novi Sad (as Co-ordinator), University of Belgrade, University of Niš, all from Serbia; and University 'Sts. Cyril and Methodius', Skopje from FYR Macedonia.

Up to the beginning of the project, studies leading to MSc degree were organized quite differently in the West Balkan region:

- students mostly worked individually with consultations with lecturers and without regular lectures/exercises;
- the list of courses was often too broad and too theoretically-oriented;
- most studies led to a general degree of MSc in informatics without particular specialization.

Most students completing an MSc degree were those wishing to pursue an academic career and were rarely employed in the industry. Because of that, significant changes in university education in informatics (and software engineering) in the region were necessary and urgent. There were four major goals of the project:

1. to create a *new* master curriculum in software engineering according to above-mentioned principles, Bologna declaration, current practice in EU, and local industry needs;
2. to make it *joint* for as many participants as possible;
3. to produce teaching and learning materials for as many new courses as possible;
4. to start with lectures as soon as possible.

Of several possible definitions of a *joint* curriculum, the strongest one has been chosen – all participating institutions should adopt the same curriculum, thus enabling sharing facilities, lecturers and students. The weakest definition would be: adjusting existing curricula and sending students to other participating institutions for a semester or two. Adopting such a weak definition would, however, highly influence the first goal of the project – to create a *new* curriculum according to high standards and needs.

The consortium members from the EU [18] helped in fulfilling these goals with their immense expertise in all phases of definition and implementation of the curriculum. The EU consortium members were respectable institutions, highly experienced in software engineering research and education and already adopting many of the principles that the new curriculum should attain.

The created curriculum has been independently reviewed by the leading European professional organization 'European Software Institute' (ESI, Bilbao, Spain). The review was positive and confirmed its importance and good quality for education of software engineering professionals.

3.1. Project Activities and Basic Decisions

The JMCSE was designed to meet the needs of the beneficiary universities and did not pretend to meet the requirements of all educational systems in the European educational area. It has been created for all B.Sc. in general informatics (with background in both science and technical faculties) and also with the aim to support student mobility between West Balkan universities participating in the project.

The first basic principle behind our Joint Curriculum was that it should represent a significant step to a European-wide employment. The curriculum should ensure that graduates are prepared for positions in the whole Europe with the knowledge and skills required for future competitiveness.

Second basic principle of the Joint Curriculum was that it should include a solid body of fundamental knowledge and should not aim to teach everything that the graduates would need later and that it should provide a basis for lifelong learning [16]. Of course, it does not mean that training in practical issues should be ignored.

Third basic principle was that all courses including theoretical ones should be SE-oriented or have a significant software engineering flavour.

The basic principles of JMCSE have been agreed and accepted by the project consortium in October 2005, after a year of consultations, discussions, and analyses. Most of the efforts in this phase of the project were devoted to the second project goal – to create the *joint* curriculum, taking care of institutions' traditions, backgrounds (technical vs. science), basic Bologna decisions for the first two cycles (3+2 or 4+1); and even traces of vanity.

The resulting principles enabled the creation of the firm-enough joint curriculum, while leaving enough flexibility for every institution to adjust it to its own needs and tradition. By the end of the project three institutions (Novi Sad, Serbia; Skopje, FYR Macedonia; Leicester, UK) adopted it officially. The first

two institutions adopted it fully as suggested by the project's documents, while Leicester adopted its subset (but according to rules for a joint curriculum). Belgrade and Niš (both from Serbia) adopted it partly at the moment, due to ongoing reforms at the universities.

It has to be emphasized that this has been a great achievement because three institutions from three countries (with different educational systems, rules, procedures, and higher education laws) adopted the curriculum. The achievement is even greater because the process took place in uncertain legal and procedural conditions and concurrently with deep and general university reforms in Serbia and FYR Macedonia.

The forthcoming sections will in more detail discuss the principles and documents of the accepted joint curriculum.

3.2. Structure of the Joint Curriculum

The curriculum JMCSE has been designed for a wide range of graduate students (general computer science, business informatics, practical informatics, engineering, economics, even mathematics) that have different pre-knowledge in the software engineering field. Also, it has been designed for "older" graduates, who have a problem of coping with the newest trends in their jobs, and would like to continue their education. As the general guidelines for those who wish to attend the JMCSE, the expected competencies for every student were provided. These guidelines can be also used by the institution to decide whether or not an induction is necessary.

- Fundamental knowledge in basic fields of mathematics;
- Ability to think logically, formulate of prerequisites, and derive conclusions in a formal or formalized way;
- Ability to understand and formulate problems, and model them to enable analysis and solving;
- Programming skills in at least procedural and object-oriented paradigms;
- Understanding of all phases in the software development cycle: requirements, analysis, design, implementation, testing, maintenance;
- Practical skills in using programming environments, database management systems (DBMS), and computer-aided software engineering (CASE) tools;
- Understanding of current trends in the development of informatics
- Ability to adapt to new circumstances, i.e. ability to learn new models, techniques and technologies as they emerge and appreciate the necessity of such continuing professional development;
- Appreciation of basic ethical and social responsibilities.

As noted, the project consortium recognized the various challenges in developing the joint curriculum due to the different environments, rules and regulations that govern education at different universities and countries. Therefore, instead of defining a fixed curriculum, we defined a curriculum template that could be implemented in several different ways by the

participating institutions, maintaining the overall goals and principles. Universities which implemented the curriculum shared:

- Goals, structure, and the course list of the curriculum,
- Principles for adopting the curriculum,
- Quality assurance and control mechanism,
- Pool of fully implemented core courses according to:
    - Course templates and
    - Principles for the development of courses.

The value of master studies was adopted to be 90 ECTS (European credit transfer system) credits. one ECTS credit is worth 20 hours of total student's workload and there are 21 - 24 contact hours per week. The studies are organized into three semesters (1 semester = 15 weeks). The first two semesters consist of lectures, while the third one is devoted to the final project/thesis. General program of studies is as follows:

- **Induction Layer:** A couple of introductory courses will be offered before the 1st official semester for students without sufficient pre-knowledge in the domain of programming techniques, software and other similar subjects necessary to follow core and elective courses (i.e. for those who graduated in economics, technical domains, employees who would like to continue education).

- **First semester:** Core courses (30 ECTS)

- **Second semester**: Elective courses (30 ECTS)

- **Third semester:** Final project (30 ECTS)

3.3 Principles for Instantiation of the Curriculum by the Institution

All courses are one-semester long and equally weighted. The total number of courses is 8 or 10 (4+4 or 5+5) and it depends on the particular beneficiary institution rules, study models and local university polices. At least 8 must be taken from the pool of existing courses (see section 4.2). The remaining two courses (if at all existing) can be defined freely by the institution. The value of each course is 6 ECTS (in case of 5 courses per semester) or 7.5 ECTS (in case of 4 courses per semester) [19]. In the latter case additional homework must be given to students in order to reach the value of 7.5 ECTS credits per every course. The number of contact hours for each course is 4 per week.

Induction layer. The courses given here can be implemented directly but can also serve as guidelines to institutions how to map them into corresponding undergraduate courses.

***Core courses*.** Every institution can choose 4 (or 5) courses from the list of core courses that will be obligatory for the students in the 1st semester. The institution can change its choice each year and have to direct students to the core courses of other institutions belonging to the JMCSE.

***Optional courses*.** In the 2nd semester students can choose 4 (or 5) courses from the following set of courses:

The rest of core courses

Optional courses from the home institution and from the other institutions belonging to the JMCSE. Institutions can organize the optional courses into strands.

The Institution does not have to offer all available optional courses each year. The Institution can include new course(s) into the set of optional courses, providing that they pass through the same quality procedures as the originally existing courses. The consortium of institutions implementing the joint studies should agree on the aims and learning outcomes of the new courses. Each institution implementing the curriculum will select appropriate technologies and products to be used for implementation and exemplification purposes, according to suggestions given in course templates.

These master studies enable students to work as professionals in development of large software or software-intensive systems. Apart from that, the joint master curriculum and appropriate delivered courses and subjects, during education, provide students with an additional qualification in different software engineering domains and the means to continue towards a PhD degree and scientific research.

4. Curriculum Courses

As suggested in [6] any curriculum can be seen as a system. The components of this system are courses, labs, coursework, final project, etc. Each component has its requirement specification. For example, a course description specifies outcomes, content, teaching methodologies, considerations and other course requirements. Therefore any curriculum is a collection of specifications.

A “hot problem” of this approach is the implementation, i. e. “staffing the curriculum”. None of the beneficiary universities has been able to implement such a curriculum in a high-quality manner independently (due to shortage of absolutely adequate experts and other resources). The best solution for applying this curriculum at different universities was to accumulate experts’ resources of the partners and implement the curriculum jointly using students’ and teachers’ mobility like in [20].

4.1. Competences and Learning Outcomes

We expect that a student following this curriculum will acquire some general competences and reach specific learning outcomes [19].

General competencies include:

- Ability to work in an interdisciplinary team
- Capacity for critical analysis and synthesis
- Capacity for applying knowledge in practice
- Capacity for generating new ideas (creativity)
- Capacity to learn
- Decision-making
- Knowledge of a second language
- Research skills

Specific learning outcomes include:

- Show mastery and critical thinking of the software engineering knowledge and skills and professional issues necessary to begin practice/research as a software engineer.
- Work as individual or as part of a team to develop and deliver high quality software artefacts, being able to analyze their level of quality.
- Identify, analyse, and reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems and organizations.
- Analyse, design and document appropriate solutions in more than one application domain using software engineering approaches that integrate ethical, social, legal and economic concerns
- Demonstrate an understanding of and critically analyze and apply current theories, models and techniques that provide a basis for problem identification and analysis, software design, development, implementation, verification and documentation.
- Demonstrate an appreciation and understanding of the importance of negotiation, effective work habits, leadership and good communication with stakeholders in a typical, industry-strength software environment.
- Learn new theories, models, techniques and technologies as they emerge and appreciate the necessity of such continuing professional development.

Each course has a multi-layer structure and implements three levels of knowledge and outcomes. The outcomes of courses are defined in terms of abilities: foundations, core, and advanced. Any course is specified in terms of prerequisite abilities, developed abilities, and trained abilities. The set of

trained abilities may be empty for some courses. So, any course adds new value to the students in two ways: firstly, the course develops some new abilities; secondly, it may train some previously gained abilities, improve them and thus increase the level of professionalism of the students (see Table 1).

Table 1. Classification of knowledge and abilities

No	Level of knowledge	Abilities	Level of professionalism
1	Foundations	Be able to describe basic concepts.	Be aware
2	Core	Be able to explain basic concepts and methods.	Understand
		Be able to apply general theoretical knowledge to solve model problems.	Be able to participate in student projects
3	Advanced	Be able to apply career-oriented theoretical knowledge and skills to solve real-life problems.	Be professional
		Be able to predict consequences and impacts of professional decisions.	Be expert
		Be able to propose innovative solutions of real- life problems.	Be pioneer

4.2. Course Study Pack

Having in mind all the above mentioned recommendations, other curriculum developments and different requirements for our curriculum we proposed the following courses [19].

Induction layer courses: Introduction to software engineering; Principles of programming, coding and testing; Project management; System modelling and design.

Core courses: Research methods; Requirements engineering; Architecture, design, and patterns; Software testing; System integration; Information system development process.

Optional courses: Software evolution; Component-based development; Formal methods engineering; Software engineering for critical systems; Privacy, ethics, and social responsibilities; Applied system thinking; Business modelling; E-business; Business process re-engineering; Service quality management; Software engineering for database systems; Advanced topics in software engineering.

Final project. Detailed project descriptions are made available to students at the beginning of the curriculum but they start working on them in 3rd semester. At the beginning of semester a condensed course on selected topics in project management can be delivered to students. Some suggested projects are: Electronic Patient Records, Electronic purse, Flight Control Systems, and E-voting System.

Every fully developed course is described by study pack that contains the following items [15], [18]:

- A detailed course template that refines the course template and also contains: a) requirements for the lecturer (job description) for each course and b) precise rules for examination and coursework;
- Supporting literature for lecturers;
- Presentation material, preferably slides in PPT format;
- Lecture notes, separate or attached to slides (preferably the latter), explaining the way in which slide contents can be delivered to students;
- Material for theoretical exercises (assignments, rules, solutions…);
- Material for practical exercises (assignments, rules, solutions, technologies, methodologies, tools…);
- Supporting literature for students (the reference list and/or actual reading material).

4.3. Course Template

Having defined the structure of the curriculum and the respective course lists, each of the courses had to be described by a corresponding requirements specification which we called a course template. A course template defines the: aims, learning outcomes, syllabus, prerequisite, and recommended assessment of that course.

As an example, figure 1 provides the course template of the optional course “Formal Methods Engineering” [19].

AIMS:

Formal methods are those with a firm basis in mathematics. They are often used in the specification and design of critical systems where failure can cause catastrophic effects such as death, damage to the environment, loss of money, etc. However, the use of these methods in large scale design and development is still not as wide-spread as originally thought. What is needed are mechanisms to engineer these methods so that they can be used in industry and on large scale systems.

This course is intended to provide the student with a comprehensive understanding and critical evaluation of formal methods and to give a detailed account of a particular technique that is based on automata theory and their industry-strength support tool (“Statemate”).

LEARNING OUTCOMES:
▪ Upon successful completion of this course, the student will be able to: ▪ critically evaluate the basis for the need of trustworthiness in large scale computer systems; ▪ critically evaluate fundamentals of formal methods; ▪ appreciate the essential issues of using formal techniques in the whole system lifecycle and in particular in requirement engineering and architecture design; ▪ critically evaluate various types of large scale system from transformational to hybrid systems; ▪ critically evaluate the role of tools and methods for engineering the formal methods.
SYLLABUS CONTENT: Large scale systems. Taxonomy of formal methods. Transformation vs. reactive vs. Hybrid systems. Automata theory. State-based development methods. State chart and activity chart. Statemate semantics and development. Real-time aspects in (e.g.) Statemate. Case studies.
PREREQUISITES: None
RECOMMENDED ASSESMENT: Coursework and unseen paper

Fig. 1. Course template example

Definition of course templates was a joint activity of the whole consortium. Developing teaching materials was considered as an individual activity, monitored by the consortium. Each course has been assigned to a single course developer (in some cases, two staff members have been involved). Developers of teaching materials had to implement everything outlined in the course templates. During that activity, several visits of beneficiary partners' staff took place to the advisory partners (UK, Germany, Spain) providing teacher retraining to develop expertise.

5. Teaching Methodology

5.1. Traditional Challenges of Teaching Software Engineering

Teaching software engineering has never been easy and no consensus has emerged about what is the best way to do it [8], [9], [25], [27]. New or experienced engineering educators may have a sincere desire to enhance student learning but are not sure which approach to take. Indeed, one may

choose from a wide array of well promoted learning enhancing pedagogies, such as active learning, cooperative learning, and problem-based learning, but, in practice, a basic question of how and which teaching method to choose still remains unclear [13].

The main reason is that the complexity of software engineering comes from the complexity of problems and it is impossible to construct real-world complexity in a classroom. Unfortunately, in software engineering if you peel away complexity, you are left with unrealistic, inappropriate problems. As software engineering is a multi-faceted discipline, there are many tradeoffs that a teacher must make, thus limiting the experience of the student. Some of the common tradeoffs are [23]:

- Practice versus theory.
- Development versus management.
- Product versus process.
- Formal versus empirical.

The above mentioned skills of a software engineer are of technical nature, but non-technical skills that are also essential to the success of the software engineer include: communication with other participants in software development process and the ability to work in a team. Working in a team (essential for a software engineer) [2], [3] requires making room for others. Most of universities do little in education in general, and in software engineering courses in particular, to teach teamwork.

Common pedagogies to have the most relevance for engineering education are [13]: traditional pedagogies; active/engagement pedagogies, and mixed methods. Three most important *traditional pedagogies* are: subject-based learning, cookbook laboratories, and group work.

However, curriculum developers and corresponding teachers have to bear in mind some additional factors to evaluate risks and benefits of a particular pedagogy for a course they intend to teach. These have been identified as factors relating to students, instructor, course, and institution. In our case:

Students' factors. Most of our students that are now studying according to the newly developed curriculum, had some experience in different software development companies and most of them are used to work in different-sized teams. Also most of them were highly motivated to experience new methodological and pedagogical approaches and willingly and actively participated in class discussions, exchange of experiences from their everyday work, and teamwork projects.

***Instructor factors*.** All teachers and assistants joined TEMPUS [18] project with high motivation to prepare new teaching materials for assigned courses. They very much appreciated help, guidance and supervision of European colleagues during retraining process. Also they all agreed to innovate their style of teaching including: combination of subject-based and project-based teaching, organizing theoretical exercises as active-discussion sessions, and organizing practical exercises in form of different-sized projects (mostly for teamwork manner).

Course factors involve: learning objectives, future implications, and pedagogical resources, all of which has been outlined in the course templates.

***Institutional factors*.** Beneficiary institutions with great satisfaction accepted the idea to introduce a modern master curriculum in software engineering. Under the project funds, all universities obtained new equipment. Most of the staff have tenure status and were able to devote important part of their working hours for preparation of course material and being re-trained.

5.2. Our Methodological Approach

A successful software engineer must possess a wide range of skills and talents [23]. We tried to adopt new methodological approach when delivering particular courses and appropriate subjects. Faced with different methodological, pedagogical and even ethical possibilities in defining the pedagogical model for courses in our curriculum we had to resolve the usual regional particularity in university education.

We moved course developers to prepare during the course a group project that would include all important aspects of software development and also to realise it emphasizing teamwork. We would like to make connection to local industry and software companies in order to define projects (even reproducing some existing projects, or working on a mirror project) running over several years, with each new generation of students taking over the result of the preceding one and developing it further.

Project-based teaching has been gaining interest in the last few years in different areas. Reasons mentioned for the adoption of a project-based approach are that it engages the student and therefore increases motivation, and that in certain fields learning by doing is the most effective way. Apart from highly practical approach we have adopted for different courses in our curriculum we have tried to employ other methodologies in order to make students more active and motivated. In other words, we are trying to avoid 'spoon-feeding' as much as possible, in the following ways:

a) Teamwork, especially for larger students' projects.

For a variety of courses students have to accomplish some larger software projects. Usually they are divided into teams [3] according to their own choice. This approach has several advantages [2]. The first is simplicity from the managerial point of view. Second is that opportunity for a student to sign up for the team of her/his choice creates a tendency to base the choice on personal relationships. Thus, the time needed for adjustments and adaptation of team members is drastically shortened. Third, efficiency of teams created in this manner tends to be rather high. It also has a major disadvantage that in real-life situations they will not be in such positions and will have to work with different team members. However, there are more advantages than disadvantages and we can be satisfied with adopted pedagogies.

b) Active-learning during exercises but also during lectures.

Teachers and assistants are trying to make active conversation with students by:

a. asking them to answer some questions related to the subject;
b. presenting them smaller problems (similar to the one previously taught or exercised) and pushing them to make a discussion in order to define main steps of a solution (most active are employed students with significant experience in work on real-life projects);
c. asking them to quote examples from real-life, similar to the one presented during class, to discus similarities and differences, and deeply analyze them.

c) Presentation of the real-life situations and problems in order to provoke discussions between the teachers (or assistants) and students.

It is similar to the previous method but it is related to "medium-size" problems (based on home-work afterwards). Analyzing, discussion and leading to solution is happens during 2-3 classes and students are supposed to finish work at home (individually or in a team). After they produce a solution, special sessions are organized. It is also interesting to mention that sessions where solutions were discussed, analyzed, and criticized, always provoked arguing and strong exchange of opinions. This produced very creative classes. Simulating reality, a high degree of freedom should be given to students in their solutions, discussions, and opinions. Yet, since students usually have little or no experience with such kind of work (projects and teamwork), some level of monitoring, guidance and supervision is needed to ensure advancements and successful results

d) Preparation of seminar papers.

Usually students are willing to participate in such kind of activities. These activities usually include several different forms.

a. Reading several scientific papers on the subject and presenting main ideas, making critical analysis and giving some concluding remarks.
b. Collecting different sources for some methodology, technology, or tool and then presenting it, analysing, discussing applicability (in case of a tool, applying it on different real-life examples).
c. Making report on a part of a project an employed student is working on in his/her company.

6. Experiences

By the third year of the project we had adopted the JMCSE in Novi Sad (Serbia) and in Skopje (FYR Macedonia), while adoption in Leicester (UK) was underway. Having most of the project goals already fulfilled we started with the lectures in October 2006 in Novi Sad. Lectures in Skopje started a semester later, in February 2007. Students from Niš also joined two courses for which they had equivalents in their current (i.e. old) curriculum.

During the first year of implementation, students were offered 11 courses, out of 18 possible. Of those 5 were core courses (out of 6) and 6 were optional (out of 12). Fourteen guest lecturers delivered eight courses at three beneficiary universities (Novi Sad, Niš, Skopje) and the funds for 115 students' mobility were used to support attendance of students to four courses outside of their home institutions. Three courses were lectured by the local staff of the University of Novi Sad, while eight courses have been lectured jointly by the guest lecturers and local lectures (prospective lecturers in the future). In the latter case the typical scenario was that the guest lecturer gave majority of the lectures, while the local lecturer took care of coursework and examinations.

From a global perspective, the results of students' mobility were that they experienced the technical challenges and the social and cultural diversity of global development: the courses with accompanying projects were carried out in an integrated way across multiple universities, students bringing the technical and cultural experience gained at one university to another one.

The idea is that in the future, beneficiary universities will realize the innovative concept of *shared courses*, delivered jointly by two to four institutions, resulting in both students' and teachers' mobility from different universities constituting a common team.

Communication, collaboration, and coordination are three main challenges in global software engineering. The experience (based on the project results, the inquiries and student interviews) has indicated certain difficulties in coordination and exercising (in situations when we have a guest teacher and a local assistant) of distributed projects, especially in the early phases.

In spite of problems in long-lasting communication between students and guest teachers, and other organizational problems, students acquired knowledge, competences and experiences in different software engineering domains and topics, which implies the ability to cope with complexity of understanding, designing and implementing such systems in the global marketplace. This comprises techniques from software engineering disciplines, including requirements engineering, software processes, software architecture and design, system analysis, testing, verification and validation.

The students were also being educated as global citizens. With specialized modules in research methodologies and professional ethics being part of the curriculum we expect to create an impact on the quality of our masters and societal relevance of their education [20]. By applying similar way of curriculum implementation in the future, students will be aware of, and trained to work with diversity (e.g. cultural, social, and economical), know how to communicate in a global network and a global team, interpret diversity and exploit it in their professional and personal lives. Early signs show that we have satisfied both teachers and students.

By the end of the project we had 16 (out of 18) developed study packs – not all of them were in its full form, however all of them contained lectures with lecture notes. After the project, Leicester (UK) also adopted the JMCSE and started with the lectures. At the same time, in Novi Sad and Skopje began the second year of studies. We are now offering all 6 core courses and

7 optional ones, planning an exchange of 7 teachers and (only) 6 students. The main reasons for a (much) lower students' mobility are following: the lack of funds, a lot of employed students which have no time for mobility, and as well the lack of suitable infrastructure that would take care of students' visits in Skopje and Novi Sad in a routine way.

7. Conclusion

The whole project went through a sequence of phases starting at the end of 2004. These phases covered:

- Definition of the curriculum goals; analysis of the situation at beneficiary universities; analysis of the special requirements of the local software industry;
- Based on that, definition of the structure and the contents of the curriculum;
- Development of teaching materials;
- Delivery of the curriculum.

During this process, the consortium had to cope with several challenges.

National educational environments. During the whole development time we were confronted with ustable educational environments in Serbia and FYR Macedonia. University reforms were without definite and clear decisions, thus, even the length of the proposed curriculum was subject to insecurity.

Differences between beneficiary institutions. The curriculum has been developed for four different faculties with different traditions in education and research. Correspondingly, their ideas of the curriculum contents were different from each other. One of the issues was the role of theoretical foundation in master studies which had been traditionally underestimated by engineering faculties.

Flexibility. Because of previous issues the consortium defined a curriculum framework, rather than to prescribe a fixed curriculum for all institutions. Based on the pool of teaching materials developed for the proposed course list, each institution has the freedom to select appropriate ones, according to the common rules.

Resolving all of these difficulties and defining a framework for joint curriculum took a year of project activities.

Teaching materials development. The development of teaching materials from scratch is a time-consuming activity. Although there were existing teaching materials of EU partners, only some of them were sufficient to serve the needs of the joint studies. Thus, a larger amount of new materials had to be produced. Main emphasis had to be placed on reusability: teaching materials had to be enriched with lecture notes to enable their application by a lecturer that has not developed the materials. Lecture notes consist of

teaching tips (methodological information) as well as additional technical information.

During this phase, an elaborate quality assurance mechanism for the study packs was developed by the project consortium. That document was necessary in order to establish the uniform quality of developed study packs in distributed environment and in the situation when the producer of study pack is not necessary the teacher of the corresponding course.

Delivery of the curriculum. In this phase we faced mostly organizational problems: a) most of the students were already employed and b) guest lecturers were not available all the time. Therefore it was not easy to find suitable intersections of their free time for organizing lectures and exercises. Students' mobility also proved to be a very demanding task. Due to the lack of official university/faculty services to support students' visits, all activities had to be performed by the teachers themselves.

Despite the mentioned difficulties, the project consortium managed to fulfil all the given goals – from creation of *new* and *joint* curriculum to its delivery in three institutions (two of them even during the project course).

The main experiences gained so far can be summarized as follows:

Common principles of quality assurance are crucial in such a multi-lateral project: curriculum validation by a validation panel including academics and industrialists; quality assurance of teaching materials; common principles of students' selection and students' assessment.

It is rather advantageous that experts of certain special fields are being responsible to work out the teaching materials in that field. In that way, high quality of the technical contents could be guaranteed.

The involvement of all partner institutions in the curriculum development process is necessary for the success of the project, in particular for the acceptance of the project results.

8. References

1. ACM/IEEE, Computing Curricula 2005, [Online]. Available: http://www.acm.org/education/curricula.html, (2005)
2. Belikova, M., Navrat, P.: Experiences with Designing a Team Project Module for Teaching Teamwork to Students, Journal of Computing and Information Technology, Vol. 13, No. 1, 1 – 10. (2004)
3. Budimac, Z., Putnik, Z., Ivanović, M., Bothe, K., Schuetzler, K.: Conducting a Joint Course on Software Engineering Based on Students Teamwork, Informatics in Education Journal, Vol. 7, No. 1, 17-30. (2008)
4. Bothe, K., Budimac, Z., Cortazar, R., Zedan, H.: Developing a joint software engineering master's curriculum across countries: report on a multi-national educational project, e-journal ITALICS, Vol. 6, No. 3. (2007)
5. Bourque, P., Dupuis, R. (ed.): Guide to the Software Engineering Body of Knowledge, IEEE CS Press (2004)
6. Caplinskas, A.: Curricula engineering: application of systems engineering methods to the development of university curricula, Information Technology and Control, Vol. 22, No. 1, 10-21. (2002)

7. Caplinskas, A., Vasilecas, O.: MOCURIS - modern curriculum in information systems:at master level. In Proceedings of the 10th European Conference on Information Systems (ECIS2002), Gdansk, Poland, V.1, 184-193. (2002)
8. Ford, G.A., Gibbs, N.E.: A Master of Software Engineering Curriculum – Recommendations from the Software Engineering Institute, IEEE Computer, Vol. 22, No. 9, 59-70. (1989)
9. Frezza, S. T., Tang, M., Brinkman, B. J.: Creating an Accreditable Software Engineering Bachelor's Program, IEEE Software, Vol. 23, No. 6, 27-35. (2006)
10. Gorgone, J.T., Gray, P. (ed.*):* MSIS 2000. Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems, Communications of the Association for Information Systems, Vol. 3, No. 1, pages 61. (2000)
11. Henderson, P.B.: Software engineering education (SEEd), ACM SIGSOFT Software Engineering Notes, Vol. 28 No. 4. 3-5. (2003)
12. Hislop, G.W., Ellis, H.J.C. et al.: Graduate Software Engineering Education: Adapting for the BSSE?, In Proceedings of the 17th Conference on Software Engineering Education and Training (CSEET'04). 1 pp. (2004)
13. Huang, M., Malicky, D., Lord, S.: Choosing an Optimal Pedagogy: A Design Approach, Proceedings of 36th ASEE/IEEE Frontiers in Education Conference, October 28 – 31, San Diego, CA, T2C-1-T2C-6. (2006)
14. Inverardi, P., Jazayeri, M.: Software Engineering Education in the Modern Age, Software Education and Training Sessions at the International Conference on Software Engineering, ICSE 2005, St. Louis, MO, USA, May 15-21, Revised Lectures, Springer. (2006)
15. Ivanovic, M., Budimac, Z.: Software Engineering Studies - a Step to Virtual University, In Proceedings of Second Balkan Conference in Informatics, Ohrid, Macedonia, November 17-19, (invited lecture), 13-21. (2005)
16. Ivanović, M., Budimac, Z., Dudan, Z.: Lifelong Learning for Small and Medium Companies in Serbia, In Proceedings of Informatics Education Europe II, Thesalloniki 29-30 November 2007, 60-67. (2007)
17. Joanne, M. A., Richard, J. LeBlanc, Jr. Et al. : Reflections on Software Engineering 2004, the ACM/IEEE-CS Guidelines for Undergraduate Programs in Software Engineering, P. Inverardi and M. Jazayeri (Eds.): ICSE 2005 Education Track, Lecture Notes in Computer Science, Vol. 4309, Springer-Verlag, Berlin Heidelberg New York, 11–27. (2006)
18. Joint MSc Curriculum in Software Engineering, project supported by Tempus under the grant no. CD-JEP-18035-2003., [Online]. Available: http://perun.im.ns.ac.yu/msc-se/
19. Joint MSc curriculum in software engineering, TEMPUS Project CD_JEP-18035-2003, Curriculum specification, 3rd release, Version: November 1, pages 40. (2006)
20. Muccini, H., Beus-Dukic, Lj., Crnkovic, I., Punnekkat, S., Vliet, H. V.: Towards a European Master Programme on Global Software Engineering, In Proceedings of Software 20th Conference on Engineering Education & Training - CSEET '07, 3-5 July 2007, 184 – 194. (2007)
21. Lethbridge, T., LeBlanc, R.J. Jr, et all.: SE2004: Recommendations for Undergraduate Software Engineering Curricula, IEEE Software, Vol. 23, No. 6, 19-25. (2006)
22. Lutz, M. J., Bagert, D.: Software Engineering Curriculum Development, IEEE Software, Vol. 23, No. 6, 16-18. (2006)
23. Mehdi, J.: The Education of a Software Engineer, In Proceedings of 19th IEEE International Conference Automated Software Engineering Conference, Linz, Austria, September 22-24, 2004, xviii-xxvii. (2004)

24. Meyer, B.: Software Engineering in the Academy, IEEE Computer, Vol. 34, No. 5, 28 – 35. (2001)
25. Mingins, C., Miller, J., Dick M., Postema, M.: How We Teach Software Engineering, Journal Object-Oriented Programming, Vol. 9, No. 11, 64-75. (1999)
26. Parnas, D.L.: Software Engineering Programmes Are Not Computer Science Programmes, CRL Report 361, Communication Research Laboratory, McMaster Univ., Apr. 1998; IEEE Software, Vol. 16 , No. 6, 19-30. (1999)
27. Show, M.: Software engineering education: a roadmap, In Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000, 371 – 380. (2000)
28. Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering: A Volume of the Computing Curricula Series, tech. report, IEEE CS-ACM Joint Task Force on Computing Curricula, 2004, [Online]. Available: http://sites.computer.org/ccse.
29. Thompson, J. B., Edwards, H. M.: Third International Summit on Software Engineering Education (SSEE III) Bridging the University/Industry Gap, ICSE'06, May 20-28, 2006, Shanghai, China, 1011-1012. (2006)
30. Varol, C., Bayrak, C.: Applied Software Engineering Education, In Proceedings of ITHET 6th Annual International Conference, July 7 – 9, 2005, Juan Dolio, Dominican Republic, T3C-25-T3C-29. (2005)

K. Bothe is a professor at the Institute of Informatics, Humboldt University Berlin. His research interests cover: compiler construction, software engineering, software testing methodology and tools, programming languages, e-learning, and logic programming. During the last years, he was the grantholder of an international DAAD project and of an EU Tempus project concerned with the development of a Joint Master's curriculum in software engineering and the construction of teaching material repositories.

Zoran Budimac is a professor at Faculty of Science, Department of Mathematics and Informatics, University of Novi Sad. He graduated in 1983 (informatics), received master's degree (computer science) in 1991 and doctor's degree (computer science) in 1994. His research interests include: mobile agents, e-learning, software engineering, case-based reasoning, implementation of programming languages. He has been project leader for several international and several national projects. He has published over 170 scientific papers in proceedings of international conferences and journals, has written more than 12 university textbooks in different fields of informatics. He is head of computer science chair.

Rebeca Cortázar is an associate professor in the Software Engineering Department, Faculty of Engineering, at the University of Deusto in Bilbao (Spain). Her interests include educational issues in Software Engineering, Software Measurement and Effort Estimation, Object Orientation and System Integration Technologies. She is currently lecturing on Object Oriented Design & Patterns and Integration Technologies. Additionally, she is an active participant in the promotion of the European Space of Higher

Education, known as the Bologna process, at her institution. Since October 2008, she holds the position of vicedean (International Relations and External Cooperation) at the Faculty of Engineering. At the present time, she is involved in several EU Tempus project proposals and participates in IBIM, an EU-USA ATLANTIS project. Dr. Cortazar received her Ph.D. in Computer Science from the University of Deusto in Bilbao (1999).

Mirjana Ivanović is a professor at Faculty of Sciences, Department of Mathematics and Informatics, University of Novi Sad. She graduated in 1983 (informatics), received master's degree (discrete mathematics and programming) in 1988 and doctor's degree (computer science) in 1992. Her research interests include: multi-agent systems, e-learning and web-based learning, data mining, case-based reasoning, programming languages and tools. She actively participates in more than 10 international and several national projects. She has published over 180 scientific papers in proceedings of international conferences and journals, has written more than 10 university textbooks in the field of informatics and ICT.

Professor **Hussein Zedan** is the Technical Director of the Software Technology Research Laboratory (STRL) at De Montfort University, UK. His research interests include formal methods, verification, semantics, critical systems, re-engineering, computer security, CBD and IS development.

Received: January 13, 2009; Accepted: February 27, 2009.