Teaching Software Engineering and Software Project Management: An Integrated and Practical Approach

Gabriele Bavota[1], Andrea De Lucia[1], Fausto Fasano[2], Rocco Oliveto[2], Carlo Zottoli[1]
[1]*School of Science, University of Salerno, 84084 Fisciano (SA), Italy*
[2]*STAT Department, University of Molise, 86090 Pesche (IS), Italy*
gbavota@unisa.it, adelucia@unisa.it, fausto.fasano@unimol.it, rocco.oliveto@unimol.it, eomeredoras@hotmail.com

***Abstract*—We present a practical approach for teaching two different courses of Software Engineering (SE) and Software Project Management (SPM) in an integrated way. The two courses are taught in the same semester, thus allowing to build mixed project teams composed of five-eight Bachelor's students (with development roles) and one or two Master's students (with management roles). The main goal of our approach is to simulate a real-life development scenario giving to the students the possibility to deal with issues arising from typical project situations, such as working in a team, organising the division of work, and coping with time pressure and strict deadlines.**

I. Introduction

One of the main challenges when teaching software engineering within an undergraduate course is providing the students with meaningful experiences they will find useful when they enter the labour market [8]. A typical example is represented by a project where students have the possibility to experience team working and understand in the practice the concepts dealt with in the course (see e.g., [10], [19], [31]). In addition to development methods, such an experience should also focus on management.

While the need of emphasising both software development and project management in undergraduate computer science curricula is long dated [2], in general a first software engineering course is more oriented to teach concepts related to software process models, requirements elicitation and analysis, software design, and software testing, while concepts related to software project management and software quality are only marginally addressed.

As an effect of these choices, it is generally difficult to assign project management and coordination roles to students of such a course, unless they have a natural attitude towards coordination. Nevertheless, in most cases the project is led by one of the students [22]. In some cases, management roles are assigned by the lecturer after identifying the attitudes of the students through some interviews [15], or based on the role preferences expressed by the students [14]. As imposed leaders can get the opposition of natural leaders in the project group, often the leaders are elected within the groups [12], [35], or the coordination roles are assigned in turn to all the students for a short period [11], [16], [17].

Each of the approaches above has some risks, due to the lack of knowledge about software project management and the lack of a senior status of the students taking management roles. In addition, students do not effectively experience the complexity of management and the different levels of responsibility in a project. For this reason, in some cases the class is viewed as a software company managed by the instructor, where the team negotiates with the instructor the selection of the project, timelines, and deliverable products [5]. In other cases a real client is involved and students relate directly to it [15] even outside of the traditional academic environment [19].

On the other hand, when the focus of the course is on software project management, showing in practice the complexity of a realistic project is a problem [26]. Building project teams within the same course might be not effective, because students would be required to work more with technical roles, than with management roles that should be assigned in turn to the different students involved in a project. Finding the right balance between technical and management issues can be difficult in this case [8]. As a result, the practical aspects tend to be restricted to a simulation of a real project [4], [26] or to an unrelated last year software engineering capstone project [21], and most of the effort is dedicated to the development of a software project management plan [27] or to address homework assignments and exercises [28].

In the academic year 2003/2004, the second author of this paper was asked to teach in the same semester a mandatory Software Engineering (SE) course for Bachelor's students in Computer Science and an elective Software Project Management (SPM) course for Master's students in Computer Science[1]. This was considered as a unique chance to build mixed project teams composed of Bachelor's students with development roles and Master's students with management roles. Such a project organisation would have solved the two main problems discussed above: (i) providing the Bachelor's students with an imposed senior leadership, thus allowing to understand the different levels

[1]Besides the lecturer of these two courses, the list of authors include two past teaching assistants of the SE and SPM courses and two past students of the SPM course.

of responsibility in a project and (ii) providing the Master's students with the main resource they required, i.e., people to coordinate, guide, coach, and mentor. In addition, within the projects Bachelor's students could have the possibility to learn more on project and quality management concepts through the Master's students.

This first experience was very successful, so it was replicated in the following years by improving and better structuring the project organisation and infrastructure. In this paper we present a seven years experience of teaching these two Bachelor and Master courses in an integrated and practical manner through the common project. This experience involved 69 Master's students and 313 Bachelor's students organised in 45 projects. We also analyse in detail the data of the projects developed in the last 5 years as we have full and coherent data for them. Finally, we analyse the preliminary results of a recent survey we conducted to get the feedbacks from these students on the impact that these courses and the project experience have had on their academic as well as professional life.

The paper is organised as follows. Sections II and III present the organisation of the two courses and of the project, respectively. The data collected from the analysis of the projects is presented in Section IV, while Section V presents the preliminary results of the survey. Section VI concludes the paper.

II. Organisation of the Courses

This section describes the organisation of the SE and of the SPM courses.

A. Software Engineering Course

The SE course is a mandatory course of the Bachelor program in Computer Science. The course aims to contribute to the students' professional profile required to operate in the software industry. It provides an overview on theory, models, techniques and technologies that characterise the development and the entire life-cycle of a software system, with particular reference to Object-Oriented (OO) development. Covered topics are software life-cycles and software development processes, the Unified Modelling Language (UML) [33], requirements elicitation and analysis, system (high-level) and object (low-level) design, implementation, and testing (with particular emphasis on black-box testing techniques). Some basic management concepts are also covered, including project organisation and communication, software configuration management, and design rationale management. The main reference book for this course [7] is integrated with further material provided by the lecturer such as the Software Engineering Body of Knowledge (SWEBOK) [24], standards, and research articles.

After successful completion of the course, students should be able to understand and be fluent in the use of software engineering terminology, communicate with other software engineers and stakeholders in a software project, take on technical roles (e.g., developer, system analyst, or tester) in a software development organisation, and be able to document all phases of the software development process.

Some prerequisites are required to the students attending this course. They should have at least good knowledge of algorithms and data structures, OO programming, and database systems (design and implementation). Knowledge of networking and web technologies is not a pre-requisite but it is highly recommended.

The course is scheduled on 24 lessons of 2 hours each and 12 laboratory lessons of 3 hours each. Some of the laboratory lessons are dedicated to train the students on the use of software development technologies, including (i) ADAMS, an advanced artefact management system developed at the University of Salerno [23]; (ii) Subversion[2], a version control system; (iii) Rational Software Modeler[3], a UML-based visual modelling and design tool developed by IBM; (iv) Eclipse[4], an integrated development environment; (v) JUnit[5] and PHPUnit[6], two code-driven testing frameworks to support unit testing; (vi) Selenium[7], a tool to create robust, browser-based regression testing automation. The remaining lab lessons are devoted to the discussion of technical issues concerned with the software systems developed by the different project teams.

B. Software Project Management Course

The SPM course is an *elective* course of the Master program in Computer Science. The main goal of the SPM course is to introduce the main concepts and techniques of software project management and quality management and train project leaders through the experience of best practices.

The course is composed of two modules. The first module focuses on software project management and aims at introducing the students with organisational and economical aspects of software engineering, including project planning and monitoring, risk management, cost estimation, and people management. The second module focuses on software quality management and provides background information about product and process quality, with particular emphasis on software metrics, software quality models, quality assurance, quality planning, quality control, and process improvement. The course is taught in an iterative way: first basic concepts are introduced with reference to general software engineering books (e.g., [7], [34]), then more details on the different topics are introduced using specific books (e.g., [3], [6], [13], [18], [36]) and further material provided of

[2]http://subversion.tigris.org/
[3]http://www-01.ibm.com/software/awdtools/modeler/swmodeler/
[4]http://www.eclipse.org/
[5]http://www.junit.org/
[6]http://phpunit.sourceforge.net/
[7]http://seleniumhq.org/

the lecturer, such as the PMI Project Management Body of Knowledge (PMBOK) [32], standards, and research articles.

After successful completion of the course, students will be able to organise the work of a team for the development of software systems within schedule and budget constraints. Students will be also able to plan and control the quality of both products and processes within a software project.

Some prerequisites are required to the students attending this course. The students should have good knowledge of software development processes, requirement elicitations and analysis, software design, and testing. They should be also familiar with UML. Last but not least, the students should have knowledge of component-based software engineering in order to (i) promote the reuse of software components and (ii) suggest solutions based on design patterns and frameworks.

The course is scheduled on 24 lessons of 2 hours each. There are no laboratory lessons. However, some lessons are devoted to discuss and share with the other Master's students particular management issues raised within the different projects.

III. Project Organisation

The two courses of SE and SPM are taught in the same semester, thus allowing to build mixed project teams composed of Bachelor's students with development roles and Master's students with management roles. The project organisation is the core of the proposed approach that allows teaching SE and SPM in an integrated and practical way. The main goal of our approach is to simulate a real-life development scenario. Besides allowing Bachelor's students to understand the dynamics of team working and Master's students to experience the project and quality management techniques presented during the course, a side effect goal is that key project and quality management concepts (that are not part of the SE course) are transferred from Master's students to Bachelor's students through the project meetings, activities, and documents. The projects have duration of four months and span different phases, including a Definition phase (three weeks), a Development phase (three months), and an Acceptance phase (one week).

A. Project Definition and Start-up

During the Definition phase Master's students attending the SPM course submit project proposals to develop software systems with a three tier architecture. A project proposal has to solve a real business problem, should include an analysis of competing systems, and should be stimulating and appealing for the developers. The project proposals are evaluated and can be rejected (in this case students need to re-formulate it). When the proposal is accepted, the lecturer "funds" the projects by providing the Master's student with a team of Bachelor's students. We prefer having the Master's students develop the project proposals instead of having proper customers do that. This because we would like to give Master's students a free rein to express their own creativity and feel the project as their project, rather than an imposed task to make as an exercise. Project ownership should also make Master's students able to better motivate Bachelor's students during project activities.

In the meantime, the Bachelor's students apply for this type of mixed and coordinated project. It is worth noting that this type of project is not mandatory: students who are not motivated can decide to work on smaller projects in teams of two or three people without the coordination of Master's students and without tight deadlines (it is even difficult to call this a "project"). Bachelor's students applying for the coordinated project can also express preferences to work with other Bachelor's students and these preferences are taken into account when building the teams.

The ratio between the number of Bachelor's students applying for the coordinated project and the number of Master's students is variable. As a rule of thumb, each team should include at least five Bachelor's students and a Master's student should not coordinate more than eight students. In fact, with a higher number of team members, the coordination effort required would be too high (and unbalanced with respect to the work load of the course), while a number of team members lower than 5 would not well simulate the typical dynamics of team working.

Normally, teams are composed of five-eight Bachelor's students and are randomly assigned to one or two (depending on the availability) Master's students. This case simulates a shortage of developers and Master's students have to work with the available people.

In case the ratio is higher than eight, a selection of Bachelor's students is needed: groups of more than eight students are composed (again preferences expressed by the students are taken into account) and randomly assigned to each Master's student. The Master's student is required to interview the Bachelor's students (under the supervision of the lecturer or a teaching assistant) and select and hire up to eight students to build his/her project team. Master's students can also interviews Bachelor's students of other groups who have not been selected for other projects. At the end, Bachelor's students who have been not selected will be required to work on a small project without coordination of Master's students. For Bachelor's students this represents a first (simulated) experience of being interviewed to get a job. Failures are very important to teach them what might also happen in the real life.

B. Development and Acceptance phase

Once the teams are composed, the project Definition phase ends and the Development phase starts with a kick-off meeting. During this meeting Master's students present the problem statement and the project management plan for the software system to be developed. Usually, the project

Table I
DOCUMENTS TO BE PRODUCED DURING THE PROJECT.

Document Name	Students in Charge	
	Master	Bachelor
Project Proposal and Problem Statement	X	
Software Project Management Plan	X	
Quality Plan	X	
Meetings Agenda	X	
Meetings Minutes	X	
Requirements Analysis Document		X
System Design Document		X
Database Design Document		X
Object Design Document		X
Test Plan		X
Test Case Specification		X
Test Execution and Incident Report		X
Test Summary Report		X
User's Manual		X
Software Project Management Report	X	
Quality Report	X	

managers also conduct an icebreaker game, i.e., a group-interaction exercise intended to start the process of forming the group of students into a team [20].

The goal for the Master's students is to (i) get a preliminary idea on the skills and attitudes of the team members; and (ii) attempt to identify whether a Bachelor's student is more task or interaction oriented and elicit potential technical leaderships and conflicts in the team.

Each project has to be completed within schedule and budget constraints. Concerning the schedule constraints, the lecturer establishes the deadlines for the three phases of the project. As for the budget constraints, project managers can employ team members for no more than 80 hours each. This means that Master's students have to collect data about the effort spent by each team member for each completed task. Several meetings are conducted during the project (generally once or twice per week) to report on progresses and/or completion of tasks, to assign new action items, and to discuss and/or make decisions about some emerging issues.

An incremental software development process model is adopted in each project. Students perform a complete requirements elicitation and analysis and high-level design of the software system to be developed (resulting in a Requirements Analysis Document and System Design Document, respectively) and then proceed with an incremental development of the subsystems. The goal is to release at least one increment by the end of the three months of the Development phase (deadline for closing the projects). For each developed subsystem, the Bachelor's students have to produce (besides the implementation) a System and Integration Test Plan, including functional test case specifications (typically derived using category partition [30]) as well as test case specifications derived by non functional requirements. An Object Design Document (mainly focusing on the specifications of the module interfaces) and test execution documents are also produced for each subsystem. Unit tests are developed (for example using JUnit), but are not documented. Finally, a Database Design Document is also produced. Table I lists the documents to be produced.

The Master's students are responsible for coordinating the project, defining the project schedule, managing project risks, organising project meetings, collecting process metrics, and allocating human resources to tasks. They have also to manage the quality of the project processes and artefacts, by defining process and product standards, collecting product and process metrics, and organising checklist-based artefact reviews for quality control before the project milestones. Finally, Master's students are required to evaluate the contribution of the Bachelor's students within the project on a four levels ordinal scale from sufficient to excellent. Student contributions can also be evaluated as insufficient: in this case, an individual extra work on the project is required to pass the exam.

Master's students develop a Software Project Management Plan before the kick-off meeting and a Quality Plan early in the project Development phase. They have also to monthly report the lecturer about the project status. Besides producing a project management report (including detailed activity re-planning for the next month) and a quality report, a formal presentation is made at the end of each month to discuss about project risks and other issues.

At the end of the Development phase, students have one week (Acceptance phase) to submit the printed final version of the documents produced and prepare the final presentation. The Acceptance phase ends with the presentation of the project, where Bachelor's students focus on the technical aspects, while Master's students discuss managerial issues and present the evaluations of the Bachelor's students.

C. Students' Evaluation

The evaluation of Bachelor's students is based both on on an individual examination on the theoretical part of the course and on their contribution to the project. The contribution of Bachelor's students to the project is defined by the evaluations given by by the Master's students weighted by the evaluation given to the quality of the project. The teaching assistants evaluate the quality of the documentation produced by Bachelor's students through an objective evaluation (based on checklists) and a subjective evaluation (their rate the overall quality of the documentation from A to D).

The evaluation of Master's students is based on the quality of the management documents produced, on the coverage of the techniques presented during the lectures, and on the way they have conducted the project and managed the team. The final presentation and the answers to the questions of the lecturer and the teaching assistants also help to understand how much the Master's students have learnt from the course.

In this type of projects Master's students have a high authority status (an imposed leadership), so they are also evaluated on the way they use it. We expect that their seniority status results in respect from Bachelor's students, but this respect can be mined by several issues, including for example lack of technical skills, lack of actions, lack of commitment, anxiety and excessive pressure on the team members. Master's students should also identify technical leaderships and positive energies within the project and try to drive them towards the project success, should dominate and resolve conflicts, and should share most decisions with the team members. For example, the evaluations of the team members made by the Master's students should be accepted and not questioned by the Bachelor's students. It is a responsibility of the Master's students to clearly define and advertise the criteria for evaluating the team members. Also, it is highly recommended that Master's students conduct and make public their evaluations through all the phases of the project, so that team members are aware of how they are being evaluated, can discuss with project managers potential problems, and improve their contribution to the project. This mitigates the risk that the Bachelor's students will not accept the final evaluations.

As a further example, Master's students are given the authority to fire team members, in case they show scarce interest in the project, do not attend the meetings, or repeatedly miss the deadlines for assigned tasks, without a valid reason. These behaviours constitute a risk for the project and Master's students are allowed to fire Bachelor's students acting in this way. However, in this case Master's students have to overcome the risk of opposition deriving from group loyalty, so they should share with the other team members such a decision, by explaining that the success of the project is more important than the individual goals.

IV. Analysis of the Projects

The first experience with this type of course and project organisation was made in 2004, with 5 Master's students and 36 Bachelor's students organised in 5 projects. The experience was replicated in 2005 with 34 Master's students and 108 Bachelor's students organised in 17 projects. The second year has to be considered as an outlier. The high number of Bachelor's students was due to two different undergraduate software engineering classes, while the high number of Master's students was due to the fact that half of them were from the SPM course and half of them were recruited from a Master's course on Advanced Software Engineering (ASE) where quality management concepts were taught in that year. In this way, students of the SPM course assumed the role of project managers in the projects, while students of the ASE course assumed the role of quality managers. Unfortunately, due to some backup problems with the ADAMS system [23] (used as project repository) we do not have access anymore to the project documents for the first two years (in particular management plans and reports that have been used as main source for the analysis). Thus, we report an analysis of the project data collected from 2006 to 2010. It is worth noting that, the characteristics of the project developed in 2004 and 2005 are in line with the characteristics of the projects developed from 2006 to 2010 [23]. Since the academic year 2010/2011, the second author of this paper does not teach the SPM course anymore. However, based on the experience of the previous seven years, the new lecturer still maintains the same project organisation, but using a different project repository and infrastructure.

Table II
Students involved in the projects from 2006 to 2010.

Year	# Projects	Students			Avg. # of Bachelor Students per Team
		Master	Bachelor	Total	
2006	4	5	30	35	8
2007	5	6	45	51	9
2008	3	3	24	27	8
2009	6	6	42	48	7
2010	5	10	28	38	6
Total	23	30	169	199	7

From 2006 to 2010, 199 students (30 Master's and 169 Bachelor's students) were organised in 23 teams. ADAMS was used as main repository and project infrastructure. Master's students generally customised the project communication infrastructure with other tools, such as Google Code[8], Microsoft Project[9], and several tools to analyse system quality, e.g., Metrics[10], chum[11] and Klocwork Insight[12]. Raw data and some exemplar projects are available on-line[13].

A. Resources Involved and Artefact Produced

Table II reports the number of projects and students involved for each year. We had generally a limited number of Master's students and a good number of Bachelor's students except in 2010 where we had a higher number of Master's students and a limited number of Bachelor's students. From 2006 to 2009, due to the more limited number of Master's students, we assigned to each team only one Master's student that was in charge of both project and quality management. However, there are two cases (one in 2006 and one in 2007) where we had the projects were coordinated by two Master's students who shared project and quality management roles. For example, in 2007 two Master's students decided to merge their teams in one team composed of 14 Bachelor's students (this is the reason why in 2007 the average number of Bachelor's students in a team is 9). In 2010, since we had a higher number of Master's students but a low number of Bachelor's students, we assigned two Master's students to each team.

Figure 1 shows the word cloud extracted from the project abstracts. It gives an idea of the topics of the these projects.

[8]http://www.code.google.com/
[9]http://www.microsoft.com/project/
[10]http://www.metrics.sourceforge.net/
[11]http://www.spinellis.gr/sw/ckjm/
[12]http://www.klocwork.com/
[13]http://www.sesa.dmi.unisa.it/reports/teachingSE

Figure 1. Word cloud extracted from the project abstracts.

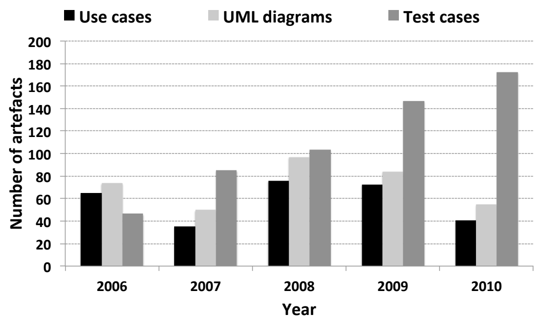

Figure 2. Average number of produced artefacts.

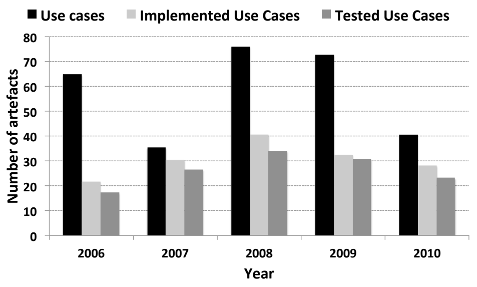

Figure 3. Average number of produced use cases.

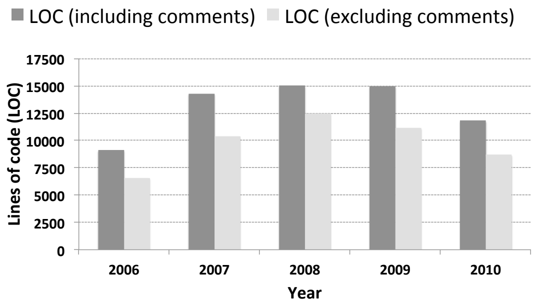

Figure 4. Average number of LOC.

The "hot topic" is represented by systems to support and manage daily activities, such as medical doctor's office, travel agency, tourist guides, and secondary schools as well as university. In addition, most of the developed projects have a web-based user interface. In the other cases, they are developed in Java with Graphical User Interface and a distributed architecture.

Turning to the size of the developed projects, Figure 2 reports the average numbers of artefacts (grouped by types) for each year. As we can see, the developed projects are not trivial. The average number of use cases is about 60 (ranging from 20 to 139). Since the goal of the project was to release at least one increment by the end of the semester, students did not implement all the documented use cases. On average, students implemented 50% of the documented use cases. In addition, due the short time available, students prioritised the implemented functionalities and, in some cases, they decided to test the ones with highest priority. On average, students tested 90% of the implemented functionalities (Figure 3).

Figure 2 also shows the average number of UML diagrams produced by students. These diagrams include use case diagrams, high-level and low-level sequence diagrams, package diagrams, component diagrams, and deployment diagrams. In the projects with a web based architecture students also used Conallen's extensions [9].

The largest set of artefacts is represented by the test cases. On average students produced more than 100 test cases. In particular, the number of test cases produced in 2009 and 2010 is higher than in the previous years, ranging from 79 to 284 in the different projects (on average 150). Indeed, in the last two years category partition [30] was introduced in the SE course. This technique resulted much more usable and effective than the combination of other black-box techniques, such as equivalence class partitioning and boundary value analysis [1]. Another reason for this increment is that besides the XUnit framework, in the last two years we also presented the Selenium framework to support regression testing. The use of such a tool improved the performances of regression testing, thus allowing students to test more functionalities and consequently increase the number of test cases.

Figure 4 shows the average size (in terms of LOC) of the projects for each year. The lines of code produced are generally higher than 10 KLOC. In particular, for the projects developed in 2007-2009 the lines of code produced is higher than the other years (15 KLOC on average). In 2007, the team composed of 14 Bachelor's students and coordinated by two Master's students developed eTour, a tourist electronic guide of 40 KLOC. This is the largest project developed in all the seven years. Another project that is worth mentioning is SMOS, a system developed by 7 Bachelor's students and one Master's student in 2008. In this case, the students developed the entire system producing 35 KLOC. However, there are also few cases where the number of lines of code is very small (around 5 KLOC). In these cases Master's students allocated much more time on the

Table III
AVERAGE NUMBER OF DOCUMENTATION PAGES GROUPED BY TYPE.

Year	Analysis	Design	Testing
2006	204	109	107
2007	141	208	166
2008	349	164	287
2009	251	137	219
2010	142	96	229

Table IV
TASKS AND ROLL-OVERS: AVERAGE DATA.

Year	# of Projects	Tasks	Roll-overs	% of Roll-overs
2006	3	53	17	32.5
2007	2	90	25	27.2
2008	2	199	30	14.8
2009	6	57	10	16.6
2010	5	63	22	34.1

documentation than on the source code. Basically, due to the short time available, in the latest phases of the projects the managers preferred to pay more attention to activities such as document review, code inspection, and testing, rather than coding. Figure 4 shows the average number of lines of code including and excluding comments. On average, the percentage of comments in the source code is acceptable, ranging from 15% to 25%.

Other than analysing the size of projects in terms of artefacts and lines of code produced, we also report the average number of documentation pages produced by the students (see Table III), organised in three groups: (i) *analysis*, (ii) *design*, and (iii) *testing*. In particular, the average number of pages of the Requirements Analysis Documents range from 142 to 349 in the different years. The testing documentation is also notable, ranging on average from 107 to 287 pages across the different years.

Once again, testing activities produced the highest number of documentation pages. Very often, the average number of pages of the testing documents is higher than 200. In 12 projects the number of pages is lower than 200 and only in 6 cases it is lower than 100. It is worth noting that testing is related to only the implemented subsystems. Therefore, the number of testing artefacts is very important and highlights the effort devoted by students to such an activity and the attention paid by the managers to quality management.

Generally, also the RAD has more than 200 pages (only in 2 projects it has less than 100 pages). The RAD of the system ELEION developed in 2008 by a team composed of 8 Bachelor's students and one Master's student is notable. ELEION aimed at managing the e-voting process and it was developed by using J2EE[14] and Ajax technologies[15]. The system includes 98 use cases and a considerable number of non-functional requirements. The RAD produced by the students was composed of 525 pages. In this project, students also used OCL (Object Constraint Language) to describe the electronic voting functionality. Also, even if students only implemented 40 use cases and tested 20 functionalities of the developed system, they produced a testing document of 246 pages (including 86 test case specifications).

[14]http://java.sun.com/j2ee/overview.html
[15]http://developers.sun.com/scripting/ajax/index.jsp

B. Project and Quality Management

As said before, each Master's student had to schedule the tasks of the Bachelor's students taking into account that they could use up to 80 hours for each team member. Master's students tried to keep low the effort for the Bachelor's students. Indeed, the average effort is usually lower than 70 hours. However, there were cases (only 4 out of 169 students) where the work load was higher than 80 hours. The reason was that the managers asked for an extra work load to compensate the abandonment of the project by a team member. We registered only two abandonments, one in 2006 and one in 2009. Also, even if the Master's students had the possibility to fire Bachelor's students, they never used such a weapon and tried to reduce the risk of abandonment.

Interesting is the data on the number of roll-overs, i.e., late tasks that were re-scheduled. Table IV reports the data we collected from some projects where these data were available. As we can see, the number of roll-overs is generally low as compared to the number of assigned tasks. However, the percentage is quite low in 2008 and 2009 and higher in 2006, 2007, and 2010. In 2010, we also observed the highest percentage of roll-overs. However, the year 2010 has to be considered as a special case. Indeed, in 2009 the Faculty changed the organisation of the Bachelor program in Computer Science and the Software Engineering Course moved from the second to the third year. For this reason, in 2010 there were no second-year students requiring to attend this course. We decided anyway to organise a remedial course for late students. The number of these students was rather low (17), while the number of Master's students (10) was higher than in the previous years. We decided that the participation to the coordinated project had to be mandatory for all Bachelor's students (including students with a scarce motivation). In addition, even allocating all the students we were not able to achieve a minimum number of students to build five teams composed of two Master's students and at least five Bachelor's students.

For this reason, we created distributed teams recruiting students from the University of Sannio (Italy) and the University of Molise (Italy). We believe that the lack of motivation of some Bachelor's students as well as the geographical distribution of some teams are the causes of the higher number of roll-overs as compared to the previous years. However, allocating two Master's students to these projects enabled to effectively and timely manage project risks and complete the projects in time and with an overall good quality.

Students usually scheduled at least one meeting per week. In 5 projects meetings were scheduled twice a week. The average duration of the meetings is around 25 minutes. In each meeting students first discussed on the progress of the assigned tasks and rolled-over some late action items. Then, managers presented the new tasks and possibly discussed about problems encountered by the team members. Meetings were usually held in the software engineering laboratory at the University of Salerno. Due to logistic issues, in 4 projects (including the 2 distributed projects in 2010), students organised virtual meetings using Skype[16].

Regarding the communication among team members, the primary channel is email. To facilitate the exchange of email usually managers defined a project mailing list. Besides emails, team members also used other communication channels, such as Internet Relay Chat (IRC). In some projects, managers also analysed the emails exchanged by students noting that the number of exchanged emails is usually stable during the development process with some peaks just before the milestones.

Turning to risk management, we observed several risks that where overestimated by students. We derived such information comparing the risk assessment reported in the first and final project reports. The abandonment of one or more team members was overestimated in almost half of the projects, while the risk related to unskilled members and members' training were overestimated in about 30% and 20% of the projects. The risk related to an optimistic schedule was also overestimated in about 20% of the projects. Besides overestimated risks we also observed underestimated risks. The unavailability or poor availability of one or more team members during a critical phase was underestimated in about 20% of the projects as well as the risk related to the delayed delivery of documents or software system components.

Concerning quality planning and control, students used the quality model of the ISO/IEC 9126 standard [25] and different direct and indirect product and process metrics to analyse the developed systems. Concerning the characteristic functionality, in almost all projects (91%) students analysed the suitability considering the functional coverage (i.e., the ratio between implemented use cases and documented use cases). Students also payed attention to the analysis of the security, performing a deeper testing on the access control subsystems. In almost all the projects (82%) students also analysed the reliability by measuring its error tolerance (and in some cases also the error recovery). For the efficiency, students focused their attention on the time performances measuring the turn-around time. Only in a few number of projects, students analysed the usage of the system resources. Learnability and operability were the two characteristics analysed by the students to assess the usability of the developed system. In addition, in several projects, students also analysed the comprehensibility of the system by verifying the presence of useful help messages. In one system, the manager also applied the Nielsen's heuristics [29] to analyse the usability of the developed system. Modifiability and analysability were the primary characteristics analysed by the students to assess the maintainability of the system. However, in several projects also testability was analysed. Finally, adaptability and installability were the main characteristics analysed related to the portability of the system.

Table V
AVERAGE NUMBER OF VERSIONS (VER) AND REVIEWS (REV) FOR THE RAD, SDD, ODD, AND TESTING DOCUMENTS.

Year	RAD		SDD		ODD		Testing	
	Ver.	Rev.	Ver.	Rev.	Ver.	Rev.	Ver.	Rev.
2006	7	3	6	3	4	1	13	3
2007	6	2	5	1	2	1	6	3
2008	4	3	5	2	2	2	5	3
2009	5	2	3	1	3	1	10	3
2010	5	2	3	2	3	1	9	3

As for to the review process, Table V reports the average number of versions and reviews for the RAD, System Design Document (SDD), Object Design Document (ODD), and testing documents. As we can see, testing documents are those with the higher number of versions and reviews. This again highlights the effort devoted by students to testing activities. The RAD and the SDD had generally lower number of versions as compared to testing documents. However, the number of reviews for the RAD is comparable to the number of reviews of the testing documents while for the SDD the number of reviews is only slightly lower than those of the testing documents. The ODD was the documents with generally the lowest number of versions and reviews. This document is mainly focused on the specifications of the module interfaces, while UML diagrams (typically class and sequence diagrams) are obtained by reverse engineering the produced code with the adopted CASE tool.

V. THE STUDENTS' POINT OF VIEW

Other than analysing project data, we conducted a survey to get the feedbacks from the students about the course and the project they participated. Since this survey was not made at the end of the projects, but years after the students attended the courses, we had the possibility to ask questions related to whether the course satisfied their expectations as well as industrial needs, in addition to questions related to the difficulty and the organisation of the course. We also asked students to evaluate how much the project participation enriched and/or complemented the knowledge acquired during the lectures of the course. In particular, we explicitly asked Bachelor's students to specify how much they learnt about project and quality concepts from Master's students

[16]http://www.skype.com

through the project activities. The questionnaire has been completed by 38 Master's students and 61 Bachelor's students. Among them 6 students participated to the project as team member (during they Bachelor program) and as project manager (during their Master program). Such students filled-in both the questionnaires.

Students generally considered adequate the topics covered by the courses. However, for the SE course some students suggested to give more details on design pattern driven development and refactoring. An interesting suggestion given by one of the respondent is to schedule some laboratory lessons to train students on the use of JMeter[17], a tool to measure the performances of a Java programs. As for the SPM course, some students suggested to expand the part of the course concerned with people and risk management. Also in this case, interesting topics were suggested by the respondents, e.g., strategies usually employed to launch a new software product on the market.

Concerning the project experience, both Master's and Bachelor's students were generally satisfied and appreciated its organisation. While Master's students particularly appreciated project management activities (i.e., scheduling and people management), generally Bachelor's students considered the collaborative work and the presence of deadlines the strength of the project organisation. These two perspectives are synthesised in two respondents' comment. One of the Master's students wrote "*In this course (SPM) you have the possibility to measure your organisation ability. But, more important, you are able to know yourself, your skills and attitudes*", while, one of the Bachelor's students summarised the SE course and the project organisation as follows: "*In this course you start understanding what is the difference between academic and industrial environments. It test your ability to work in team and pressed by strict deadlines that usually reign supreme in the software industry. Very important is also the final presentation of the developed systems, since you start to train yourself to publicly present your work. One of the most useful courses, one that gave me something, that contributed to my professional training. When I started to work in industry I realised that this course and the project I participated were extremely important.*"

Besides technical aspects, such as developed methodologies and tools, the project was extremely useful for the students to enrich their communication skills and understand the real life of software engineers and project managers in industrial environments. In addition, Master's students declared that the participation to the project contributed to sensibly increase the knowledge on risk and people management, as well as on quality management. The project organisation also facilitated the transfer of key project and quality management concepts from Master's students to Bachelor's students through the project activities and documents. In particular, Bachelor's students enriched their knowledge on key project management concepts, such as schedule and planning as well as people management. Part of the Bachelor's students also acquired key quality management concepts through the project. The reason why quality concepts were acquired only by a sub-population of Bachelor's students is that generally Master's students allocated only few team members on quality control activities. Thus, only a few number of Bachelor's students deeply read the quality management documents provided by Master's students. The other students did not pay attention to such documents and they were not able to properly catch concepts related to quality management.

Finally, we asked the students whether they could recommend the course to younger students and to write a slogan to convince (or discourage) them. All the respondents recommended the course to younger students. As for the slogan, we received some serious slogans, such as "*This course gives you a sense of responsibility. You will learn how to do things faster, by organising and planning your work better*", or "*A good opportunity to get in touch with the labour market*", or "*The SE course opens your mind and it is so satisfying to see that your software system – yes, the software system that you believed impossible to develop some months before – works (and works well) and has hundreds of documentation pages.*", as well as some playful slogans, such as (for the SPM course) "*This could be your only chance to fire someone in your life!*".

VI. Conclusion

In this paper we presented an integrated and practical approach to teach Software Engineering (SE) and Software Project Management (SPM) courses. The approach is based on mixed project teams composed of Bachelor's students (with development roles) and Master's students (with management roles). The experience was very successful. The success was demonstrated by the higher quality of the documentation and source code produced within these projects (with respect to non coordinated projects), balanced by an accurate distribution of the effort to the different activities as well as to the different team members.

The success of this experience was also perceived in the enthusiasm of both Bachelor's and Master's students, in the positive feedbacks and in the great improvement of the communication skills of the students. Most of the students that responded to our survey questionnaire declared that after the course and the project they were more mature and better prepared for the workforce, having at least some idea of some industrial scenarios. Most of the respondents are now employed in industry and some of them declared that during the interviews to get the job they used what they learnt within their project experience. In some cases they showed to the employers part of the documentation produced during the project to reinforce their position.

[17]http://jakarta.apache.org/jmeter/

References

[1] V. Arnicane. Complexity of equivalence class and boundary value testing methods. In *Scientific Papers, University of Latvia*, pages 80-101, 2009.

[2] J. Beidler. Teaching project management. In *Proc. of SIGCPR Conf.*, pages 20–24, 1979.

[3] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, K. Horowitz, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[4] A. Bollin, E. Hochmuller, and R. Mittermeir. Teaching software project management using simulations. In *Proc. of CSEET*, pages 81–90, 2011.

[5] P. Brazier. Process and product in a software engineering course: simulating the real world. In *Proc. of Frontiers in Education*, volume 3, pages 1292–1297, 1998.

[6] F. Brooks. *The mythical man month*. Addison-Wesley, 1995.

[7] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering: Using UML, Patterns, and Java*. Prentice Hall, 2003.

[8] A. T. Chamillard and K. A. Braun. The software engineering capstone: structure and tradeoffs. In *Proc. of Technical Symposium on Computer Science Education*, pages 227–231, 2002.

[9] J. Conallen. *Building Web applications with UML*. Pearson Education, 2002.

[10] D. Dahiya. Teaching software engineering: a practical approach. *SIGSOFT Software Engineering Notes*, 35:1–5, 2010.

[11] P. Doerschuk. Incorporating team software development and quality assurance in software engineering education. In *Proc. of Frontiers in Education*, pages 7–12, 2004.

[12] M. Feldgen and O. Clua. An integrated software engineering workshops program. In *Proc. of Frontiers in Education*, volume 3, pages 1016–1021, 1998.

[13] N. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company, 2nd edition, 1996.

[14] Z. Gao and C. Xie. The study of content simulation using in the software project management teaching. In *Proc. of Int'l Workshop on Education Technology and Computer Science*, volume 3, pages 576–578, 2010.

[15] M. Gnatz, L. Kof, F. Prilmeier, and T. Seifert. A practical approach of teaching software engineering. In *Proc. of CSEET*, pages 120–128, 2003.

[16] L. Hai. The four ps in an undergraduate software engineering course. In *Proc. of Frontiers in Education*, pages S4E-7-S4E-12, 2007.

[17] L. Huang, L. Dai, B. Guo, and G. Lei. Project-driven teaching model for software project management course. In *Proc. of ICCSSE*, volume 5, pages 503 –506, 2008.

[18] B. Hughes and M. Cotterell. *Software project management*. McGraw Hill, 4th edition, 2006.

[19] E. P. Katz. Software engineering practicum course experience. *Proc. of CSEET*, pages 169–172, 2010.

[20] E. West. *The big book of icebreakers: quick, fun activities for energizing meetings and workshops*. McGraw Hill, 1999.

[21] P. Kruchten. Experience teaching software project management in both industrial and academic settings. In *Proc. of CSEET*, pages 199 –208, 2011.

[22] L. Leventhal and B. Mynatt. Components of typical undergraduate software engineering courses: Results from a survey. *TSE*, 13(11):1193 – 1198, 1987.

[23] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. Fine-grained management of software artefacts: the adams system. *SPE*, 40(11):1007–1034, 2010.

[24] IEEE. Guide to the Software Engineering Body of Knowledge (SWEBOK). Angela Burgess, 2004.

[25] ISO/IEC 9126. Software engineering – Product quality – Part 1: Quality model. 2001.

[26] P. Mandl-Striegnitz. How to successfully use software project simulation for educating software project managers. In *Proc. of Frontiers in Education*, pages 19–24, 2001.

[27] J. McDonald. Teaching software project management in industrial and academic environments. In *Proc. of CSEET*, pages 151 – 160, 2000.

[28] M. Murphy. Teaching software project management: a response-interaction approach. In *Proc. of CSEET*, pages 26 –31, 1999.

[29] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Proc. of Human Factors in Computing Systems Conference*, pages 249–25, 1990.

[30] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Comm. of the ACM*, 31(6):676–686, 1988.

[31] W. Pádua. Measuring complexity, effectiveness and efficiency in software course projects. In *Proc. of ICSE*, pages 545–554, 2010.

[32] PMI. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, 2008.

[33] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, 2004.

[34] I. Sommerville. *Software Engineering*. Pearson Education, 8th edition, 2007.

[35] L. Werth. Software process improvement for student projects. In *Proc. of Frontiers in Education*, pages 2b1.1 –2b1.4 vol.1, 1995.

[36] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering - An Introduction*. Kluwer, 2000.