A Two-Year Software Engineering M.Sc. Degree designed under the Bologna Declaration Principles

João M. Fernandes
Departamento de Informática
Universidade do Minho
4710-057 Braga, Portugal
jmf@di.uminho.pt

Ricardo J. Machado
Departamento de Sistemas de Informação
Universidade do Minho
4800-058 Guimarães, Portugal
rmac@dsi.uminho.pt

ABSTRACT
This paper presents and discusses the syllabus of a second cycle degree on Software Engineering in which any student that finishes any undergraduate Computing degree (Computer Engineering, Computer Science, Information Systems, Information Technology, and Software Engineering) can enroll. In the first year, the degree is composed of two 30-ECTS modules, one dedicated to software analysis and design and the other devoted to software quality and management. Each module is composed of five curricular units, being one of them dedicated to the experimental integration of the module's topics. The second year allows two different paths to be followed by the students. The professional path includes a 30-ECTS industrial project, while in the scientific path students must write a 45-ECTS master dissertation. The degree is mainly structured to consider the Bologna Declaration that is now being used in Europe to recast all university degrees. Additionally, we also considered the Software Engineering 2004 Curriculum Guidelines and the Knowledge Areas described in the SWEBOK.

Keywords
Software engineering, Bologna declaration, master degree, project-led engineering education.

1. INTRODUCTION
Across Europe, a great number of Computing departments are concerned with the process of restructuring their curricula, according to the Bologna declaration, announced in June 1999, by the ministers of education of several European countries [8]. At Universidade do Minho (UMinho), located in the cities of Braga and Guimarães (Portugal), all five-year undergraduate degrees are being restructured to take into consideration the principles of the Bologna Declaration. Among those degrees, three belong to the Computing field: Informatics and Systems Engineering, Mathematics and Computing Science, Management Informatics. These degrees are to be restructured in two cycles: a three-year first cycle and a two-year second cycle. This paper explains the syllabus of a second cycle degree (Master Course) on Software Engineering which can be enrolled by any student that finishes any undergraduate first cycle Computing degree at UMinho (or at any other institution that offers degrees with the similar learning outcomes).

The degree was designed to consider the principles stated in the Bologna Declaration. It runs in 4 semesters and consists of a total of 120 ECTS (European Credit Transfer and Accumulation System). The first year of the course is divided in two modules of 30-ECTS and the second year includes either two 15-ECTS modules and an industrial project, or one 15-ECTS module and a research-oriented master dissertation. Each 30-ECTS module is composed of five curricular units, being one of them dedicated to the experimental integration of the module's specific curricular topics. These project-oriented curricular units are included to make the degree follow the project-led engineering education (PLEE) paradigm [10].

Due to several causes (like, lack of resources and facility of grading), many software engineering courses/degrees compromise the project experience by reducing the team sizes, project scope, and risk. The solution relies usually in dividing the students among small groups (two to four persons), solving all the groups the same software engineering project. In the proposed degree, we suggest to introduce two two-semester software engineering experimental curricular units, in which all students work together to develop a new software system, or to diagnose and analyze the process or the artifacts of a software system developed by third parties. This approach provides a more realistic project experience for the students and facilitates the participation of software houses in the learning activities. This constitutes an important factor to ensure that students tackle modern industrial problems, shortening the links between students and their potential future employers. This also solves, at least partially, the fact that educators rarely have the time required to manage real software projects in addition to their normal pedagogical duties [3]. In fact, an important part of education in an academic setting is the practical application of concepts. The application of software engineering in the academic environment is quite different from software engineering in a professional context [4]. In any case, it is the task of teachers to provide realistic experience to better prepare students for work in professional settings.

Team-based projects are a well-known mechanism to motivate students by giving them the chance to participate in the type of work and environment that can be found in a software house [2, 5]. When such learning mechanism is adopted, it is reported that students can learn more about "real" software engineering than with less ambitious frameworks [7, 9]. However, the usage of a format that resembles the context of a software house is more difficult to implement, which unfortunately makes it quite uncommon in software engineering education.

For project-like curricular units, the proposed degree adopts a company-based framework, in which students collaborate in the accomplishment of a software product- or process-related activity (analysis, design, implementation, test, maintenance, evolution, management) as if they were employees of a software house. Within the project, students are organized in a simulated software development environment, with each student being responsible for specific individual and group tasks. In [13], Way has introduced several attributes that are not typical in a software engineering course, such as class-wide product brainstorming sessions, overlapping subgroups of students, distributed group working,

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

weekly engineering meetings, and business and marketing strategic planning aimed at releasing the finished product to the outside world.

Nowadays, being a software engineer requires not only mastering a programming language, but also know-how in several technical and managerial topics. The proposed degree must include modern topics on software engineering, such as open-source software development, service–oriented architectures, integration with legacy systems, usage of components and libraries, model-driven development, software methods for embedded and pervasive systems, and construction of web-based applications [11].

Additionally, we also considered the recommendations existent in the Software Engineering 2004 Curriculum Guidelines [12] and the Knowledge Areas described in the SWEBOK [1].

2. BOLOGNA DECLARATION AND ECTS

In June 1999, 29 European ministers in charge of higher education met in Bologna (Italy) to lay the basis for establishing a European Higher Education Area by 2010 and promoting the European system of higher education worldwide. In the Bologna Declaration, the ministers affirmed their intention:

- to adopt a system of easily readable and comparable degrees;
- to adopt a system with two main cycles (undergraduate and graduate);
- to establish a system of credits (such as ECTS);
- to promote mobility by overcoming obstacles;
- to promote European co-operation in quality assurance;
- to promote European dimensions in higher education.

In short, ECTS is considered to be a common framework for Academic Recognition. ECTS is a way of ensuring equivalent academic recognition between colleges and universities and its adoption in Europe is becoming a reality. The system is based on transparency and mutual trust. ECTS is a student-centered system based on the student workload required to achieve the objectives of a program, objectives preferably specified in terms of learning outcomes and competences to be acquired. ECTS was introduced in 1989, within the framework of Erasmus, now part of the Socrates program. ECTS is the only credit system, which has been successfully tested and used across Europe. ECTS was set up initially for credit transfer. The system facilitated the recognition of periods of study abroad and thus enhanced the quality and volume of student mobility in Europe.

ECTS is based on the principle that 60 credits measure the workload of a full-time student during one academic year. The student workload of a full-time study program in Europe amounts in most cases to around 1500–1800 hours per year and in those cases one credit stands for around 25 to 30 working hours.

Credits in ECTS can only be obtained after successful completion of the work required and appropriate assessment of the learning outcomes achieved. Learning outcomes are sets of competences, expressing what the student will know, understand or be able to do after completion of a process of learning, long or short.

Student workload in ECTS corresponds to the time required to complete all planned learning activities like attending lectures and seminars, studying, and preparing projects and exams.

Credits are allocated to all educational components of a study program (such as curricular units, courses, placements, dissertation work, etc.) and reflect the quantity of work each component requires to achieve its specific objectives or learning outcomes in relation to the total quantity of work necessary to complete a full year of study successfully.

3. DEGREE SYLLABUS

3.1 Pre-Requisites

To enroll in this degree, students must have, at least, competencies and skills in the following SEEK (Software Engineering Educational Knowledge) Areas [12]:

- CMP — Computing Essentials (Computer Science foundations, Construction technologies, Construction tools, Formal construction methods)
- FND — Mathematical & Engineering Fundamentals (Mathematical foundations, Engineering foundations for software, Engineering economics for software)

These skills are obtained in any first cycle computing degree at UMinho, and typically in any 3-year degree in Computer Engineering, Computer Science, Information Systems, Information Technology, and Software Engineering. More specifically, we expect students to be able to adopt rigorous methods of analysis and specification to build programs, and to use proper tools (editors, compilers, interpreters, debuggers, and integrated development environments - IDEs) to execute them.

3.2 General Structure

The degree runs in 4 semesters and consists of a total of 120 ECTS. Its general structure is presented in table 1.

<table>
<tr><td rowspan="2">first year</td><td colspan="2">first semester</td><td colspan="2">second semester</td></tr>
<tr><td colspan="4">Module on Analysis and Design (30 ECTS)</td></tr>
<tr><td></td><td colspan="4">Module on Quality and Management (30 ECTS)</td></tr>
<tr><td rowspan="4">second year</td><td colspan="2">Professional path</td><td colspan="2">Scientific path</td></tr>
<tr><td>first sem.</td><td>second sem.</td><td>first sem.</td><td>second sem.</td></tr>
<tr><td>first module (15 ECTS)</td><td rowspan="2">Industrial Project (30 ECTS)</td><td>Module (15 ECTS)</td><td rowspan="2"></td></tr>
<tr><td>second module (15 ECTS)</td><td colspan="2">Master Dissertation (45 ECTS)</td></tr>
</table>

Table 1 – Generic structure for the degree

The course's first year is divided in two one-year modules of 30-ECTS. Each 30-ECTS module of 5 curricular units, being one of them devoted to integrate the module's topics, by adopting a set of laboratorial experiments. The second year includes either a professional or a scientific path. In the former, students enroll, during the first semester, in two 15-ECTS modules and, in the second semester, develop an industrial project. In the latter, students enroll, during the first semester, in just one 15-ECTS module and start a research-oriented master dissertation, which they continue in full-time in the second semester. Each 15-ECTS module is composed of 3 5-ECTS curricular units.

Each 15- or 30-ECTS module must obey the following issues:

- There is a unique assessment (i.e. it exists only one unique mark for the module, even though it is composed of 5 internal curricular units).
- The module is supported by a set of teachers, being one of them the coordinator.
- Each laboratory must include all the subjects of the other internal curricular units. It must be accompanied by all teachers of the module, so that they can adjust their materials to better support the execution of the project.
- The project must develop not only the technical competences of the students, but also planning and management skills.

The 15-ECTS modules must address one application area of the main topics of the master course. Each 15-ECTS module must incorporate the methodological approaches covered by the first year modules. Examples of these 15-ECTS modules are: (1) data-oriented enterprise applications; (2) pervasive software and ubiquitous services; (3) real-time, embedded and critical systems; (4) industrial informatics and automation. Students must choose either one or two 15 ECTS modules, depending on the chosen path (professional or scientific).

3.3 Module on Analysis and Design

Analysis and design are activities included in the software development process. The development refers to the software lifecycle phases responsible for conceiving, designing, building, and testing a software artifact. Within the development, one finds the implementation phase (also designated construction). Excluded from the development are, for example, the economic viability studies, the maintenance tasks, and the effective usage of the software system. These issues are included in the module on quality and management.

The main aim of this 30-ECTS module on analysis and design of software is to provide students with the methodic skills to tackle any software development project. More specifically, we expect students to be able to analyze and design highly-complex software, to obtain correct and reliable solutions based on sound engineering principles.

This module is organized around 5 Knowledge Areas (KAs), as defined in [1], whose internal curricular units that support them sum up 30 ECTS. The assessment and grading is made through a unique global exam that may touch all the topics covered within the module.

Curricular unit	ECTS
Analysis and Modeling of Requirements (AMR)	5
Software Architectures (SA)	5
Formal Methods (FM)	5
Usability and Interaction (UI)	5
Laboratory of Software Development (LSD)	10

Table 2 – Curricular units for the 30-ECTS module on (Software) Analysis and Design

3.3.1 Analysis and Modeling of Requirements

The area of software requirements deals with the acquisition, analysis, specification, validation, and maintenance of software requirements. Requirements are the properties that a given system (still in project) will exhibit when its development is finished. This area (KA #1 "Software Requirements" [1], and MAA "Software Modeling & Analysis" [12]) is recognized as being extremely important for industry, since its activities have a great impact on the development process. Upon successful completion of this curricular unit, students should be able:

- to define what type of procedures the requirements engineering team is supposed to execute at the development process, by identifying the formal involvement of the stakeholders, along requirements engineering process;
- to define the way requirements are to be elicited and the techniques to use to correctly gather requirements from all the (human and non-human) sources;
- to detect and solve the conflicts among the captured requirements, to define the boundaries of the system under project and the way it interacts with its environment, and to transform user requirements into software requirements;
- to handle the structure, quality and verifiability of the requirements document, which normally includes: (1) the definition of the user requirements; and (2) the specification of the software requirements that establishes the agreement between clients and developers of the software;
- to examine the requirements document to guarantee that it correctly describes the needed system, through inspections or formal revisions, and rapid prototyping of its interfaces;
- to manage the change of requirements to ensure a well-defined semantics for them, and their traceability during the development process.

3.3.2 Software Architectures

Software design (KA2 "Software Design" [1], and DES "Software Design" [12]) is both the process of defining the architecture, its components, the interfaces and other characteristics of the system (or component), but also the result of that process. From the process perspective, software design is the activity where requirements are analyzed to produce the description of the internal organization of the system. From the product perspective, software design (the result of the process) must describe the system's architecture (i.e., how the system is decomposed and organized in components), the interfaces between components, and also the components themselves, with a given level of detail that permits their construction. Upon successful completion of this curricular unit, students should be able:

- to evaluate the importance of a software architecture for the effective availability of a solution;
- to define the different types of architectural models for describing the system's structure, the decomposition with layers or by control elements;
- to analyze the existing solutions and its level of flexibility and reuse;
- to analyze and evaluate the qualities of an architecture (conceptual integrity, correctness and completion, and constructability), the attributes that reveal in design time (changeability, portability, reuse, integrity and testability), and the attributes that appear in execution time (performance, safety, dependability, usability, and functionality);
- to reuse pre-defined, efficient and already-tested solutions within the problem domain, using, for example, architectural structures and viewpoints, architectural styles, design patterns, program families, and frameworks.

3.3.3 Formal Methods

The formal methods area (KA9 "Software Engineering Tools and Methods"; "software engineering methods" sub-area [1]) includes all usages of discrete mathematics to tackle software engineering problems. The application of formal methods to the development of software uses formal languages to describe software artifacts

(specifications, architectures and code), which allows the formal proof of properties and their reification to support the implementation. This curricular unit covers the development of software based on Formal Methods, with an emphasis on specification techniques based on discrete mathematics that allow the synthesis of the system's implementation, assuring its correctness by construction and by verification. Upon successful completion of this curricular unit, students should be able:

- to identify the role and importance of formal methods within the development of complex software systems;
- to constructively model the requirements of a software system with a formal specification language;
- to reason with rigorous models, by identifying the proof properties;
- to use an environment to prototype formal specifications.

3.3.4 Usability and Interaction

Usability (KA2 "Software Design" (sub-area Key Issues in Software Design) [1], DES – Software Design / VAV – Software V&V [12]) is a property of the interaction of a given system and its users defined by the ISO 9241-11 standard as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Ensuring the usability of a system requires its development in such a way that its users achieve their goals efficiently and satisfactorily. The definition and evaluation of the usability aims, the task analysis, or the definition of the most appropriate interaction paradigms to a given context are examples of issues that have a great impact in the development process. The Usability Engineering gathers a set of methods, techniques and tools, derived from the Human-Computer Interaction area, that make usability a central issue in software development. Upon successful completion of this curricular unit, students should be able:

- to identify the different types of models relevant in the usability engineering process;
- to apply usability analysis techniques;
- to develop an interface with toolkits and existing components, creating for that purpose a software architecture to support the interactive system.

3.3.5 Laboratory on Software Development

This unit aims to integrate in a single implementation project, all the competencies acquired in the other curricular units of the module. The implementation phase (KA3 "Software Construction" and KA9 "Software Engineering Tools and Methods", software engineering tools sub-area) [12], PRF – Professional Practice [1]) is a fundamental act in software engineering, since it consists on the construction of software in accordance with the user requirements to accomplish the non-functional requirements.

The implementation activities are directly related with the design, since the former must transform into code the architectures conceived and decided within the execution of the latter. This transformation is becoming each time more automatic, which demands the intensive usage of software IDEs. This unit must follow a PLEE (Project-Led Engineering Education) approach. Thus, the topics included in the other curricular units of the module on analysis and design must directly contribute to the effective execution of the laboratorial experiments of this unit. A careful coordination among the teachers must be assured, so that this curricular unit effectively integrates a broad set of skills and competences within the software development. Upon successful completion of this curricular unit, students should be able:

- to construct, integrated in a team, correct complex software systems, in accordance with the requirements, by combining analysis, design, coding and test activities;
- to apply in practical situations, as typically found in the software industry, the skills acquired in the module;
- to use tools (meta-CASE, frameworks, IDEs) to support all the activities inherent to software development;
- to adopt standards to ensure an uniformity both in process and notation and to promote the interoperability and portability of the designed solutions.

3.4 Module on Quality and Management

Software Management refers to activities responsible for planning, coordinating, measuring, monitoring, controlling, and documenting all the development tasks, either during the project or after it (where, one may include, for example, maintenance tasks). Software management is considerably distinct from the one followed in other engineering fields, due to the specific characteristics of the software and its underlying process.

The main aim of this 30-ECTS module on quality and management of software is to provide students with the methodic skills to control and plan the software development activities, to obtain with a systematic, disciplined and quantifiable approach solutions that are economically viable.

This module is also organized around 5 SWEBOK KAs, and their internal curricular units sum up 30 ECTS. The assessment and grading is made through a unique global exam that may touch all the topics covered within the module

Curricular unit	ECTS
Project Management (PM)	5
Software Process and Maturity (SPM)	5
Quality and Test (QT)	5
Maintenance and Evolution (ME)	5
Laboratory of Software Engineering Management (LSM)	10

Table 3 – Curricular units for the 30-ECTS module on (Software) Management and Quality

3.4.1 Project Management

In the management of software engineering projects (KA7 "Software Engineering Management", software project management sub-area [1], MGT – Software Management [12]), one applies knowledge and tools to the project activities, to fulfill the defined requirements. Upon successful completion of this curricular unit, students should be able:

- to establish the connection between the organizational and the software engineering activities, by dealing with the management of strategic, human-resources, communication, clients, supplying and outsourcing issues;
- to apply techniques and tools to the project activities for fulfilling the established requirements;
- to formally authorize the beginning of the project, by proceeding viability studies, establishing the requirements revision, and defining the requirements revision policy;
- to define the best way to finish a project to achieve the proposed aims, through process and project planning definition, deliverables characterization, effort/cost

estimation, resources allocation, risk management, and quality management.
- to coordinate persons and other resources to fulfill the plan.
- to analyze, in some critical points of the project, the fulfillment of the system's requirements.

3.4.2 Software Process and Maturity

The software engineering process can be seen at two different levels. The first level focus on the activities executed during the acquisition, development and maintenance of the software artifacts. The second level is related to the definition, implementation, measurement, management, improvement, and change of the software engineering process itself. Upon successful completion of this curricular unit, students should be able:

- to promote the assessment of software process and to monitor, in collaboration with software engineers, the software process improvement efforts;
- to identify the positive and negative aspects of the software process, through the acquisition, analysis, and interpretation of quantitative data;
- to define the software process for facilitating the human communication and comprehension in the execution of the software engineering tasks;
- to propose changes to a software process, so that the needs of the developers are fulfilled.

3.4.3 Quality and Test

The quality of a product is normally described as the set of characteristics (of the product or the service) that must exist to satisfy the required needs. Within software engineering, quality can be understood as the efficient, effective and comfortable usage, by part of a group of users, of a software system for a set of valid functionalities and under some given conditions. All these restrictions justify the huge dependency among software quality and the fields of requirements engineering and software metrics.

Software quality is an activity that spans over all the software process and that may require the explicit treatment of non-functional issues like, for example, the dependability, the reliability, the portability, the maintainability, and the availability. Other quality aspects directly related with the software engineering process are the code style, the reusability of objects, the requirements traceability, the code modularity, and independence among curricular units.

Software testing constitutes a mandatory phase of the software development, and simultaneously a technique to evaluate and to improve the product quality, through the identification of its defects and problems. Test consists on the dynamic verification of the software systems' behavior with respect to the expected behavior, using a set of finite test cases, specially chosen to cover the most critical cases. Upon successful completion of this curricular unit, students should be able to:

- to characterize the quality attributes of the software artifact and to plan the needed processes to obtain it;
- to define the procedures to be executed for ensuring that the software product fulfils its requirements and that it reaches the highest possible level of quality, taking into account the project restrictions;
- to indicate how quality plans are being implemented, and also how adequately are the initially specified requirements being incorporated into the software products under development;
- to use metrics for obtaining indicators as a tool to support quality analysis and decision taking in software engineering management;
- to use different kinds of software tests, either related with the artifact being tested (unit, integration, or system testing) or the test aims (acceptance, installation, alpha and beta, conformance, regression, performance, recovery, configuration, and usability testing);
- to organize tests according to the way they are generated (ad hoc, specification-based, code–based, fault-based testing) or the implementation knowledge (black-box or white box testing);
- to plan and execute the testing process, using metrics related to programs (accounting failures and faults) and related to the kind of tests (assessing how adequate are the tests being used);
- to organize the test activities, taking into account the human resources, the tools, and the metrics.

3.4.4 Maintenance and Evolution

The maintenance phase starts as soon as a software product is deployed, even though some tasks are supposed to start earlier (during development), since to deal with deficiencies, technological changes, and modifications of user requirements it is necessary to be prepared as earlier as possible. Upon successful completion of this curricular unit, students should be able to:

- to define the maintenance process that best fits the software under consideration;
- to identify the technical, managerial, and financial factors that affect maintenance;
- to adopt the most adequate maintenance techniques and to effectively execute them;
- to assess in software audits the conformance of software processes and products to regulations, standards, plans and applicable procedures;
- to assess the importance of software portability and to identify the restrictions to its achievement;
- to use component-based programming models and to assess the impact that functional modifications (new releases or products) may bring to the software architecture;
- to use metrics for assessing the quality of the produced software architectures and its level of maturity and evolvability.

3.4.5 Laboratory on Software Engineering Management

This unit aims to integrate all the competencies acquired in the other curricular units of the module, by analyzing and diagnosing real software cases (both at the process– and product–level). Since managing engineering activities requires indicators, the definition of software-related metrics is of great importance. Metrology is a topic typically offered in traditional engineering curricula. In what concerns software engineering, although physical measures do not exist, the need for metrics is crucial to effectively conceive software engineering as a real branch of engineering. Under these circumstances, metrology techniques must be use to systematically quantify software artifacts and their corresponding processes to obtain numeric representations of their properties.

It is recommended that industrial partners may collaborate in this experimental curricular unit by making available real cases and

reports that can be studied and analyzed in laboratory. The topics included in the other curricular units of the module on quality and management must directly contribute to the effective execution of the laboratorial experiments of this unit. Upon successful completion of this curricular unit, students should be able:

- to construct, integrated in a team, correct complex software systems, in accordance with the requirements, by combining analysis, design, coding and test activities;
- to apply in practical situations, as typically found in the software industry, the skills acquired in the module;
- to use tools (meta-CASE, frameworks, IDEs) to support all the activities inherent to software development;
- to adopt standards to ensure an uniformity both in process and notation and to promote the interoperability and portability of the designed solutions.

3.5 Project or Dissertation

The last part of the degree is composed either by a 30-ECTS industrial project or a 45-ECTS research–oriented master dissertation. Internally, this dissertation is divided in a 15-ECTS part, during the first semester, to write the thesis proposal and a 30-ECTS part, during the second semester, to develop the thesis and write the thesis. Both the project and the dissertation must be developed within the topics associated with the first year modules and supervised by teachers who are experts in software engineering.

The industrial project is the best choice for students that plan to become professional software engineers, because it provides an environment where they participate in a real software project within a development team (typically, a software house). This participation is however controlled by the university tutor, which is responsible for escorting his activities and for helping him in some specific points.

The master dissertation provides the right framework for students who wish to follow a career more oriented towards research. This alternative requires students to define a research problem, to devise a solution for it, to write a document describing the conducted research, and to present and defend publicly the thesis.

4. SUMMARY

In this paper we have discussed a two-year degree in software engineering, designed to cope with the principles of the Bologna Declaration. This degree is also strongly influenced by SWEBOK and the SE2004.

This ensures that the students will receive education on a broad set of areas related to software engineering. The authors believe that this degree does not present the deficiencies noted by [6]. We believe that the proposed degree:

(1) puts emphasis on the principles of software development;
(2) includes important topics such as specification and testing, which are to be taught in great detail.
(3) covers the major issues of software development in such a way that students understand what choices are available and how to use them.
(4) exposes students to software design;
(5) includes important areas of the discipline, such as real-time and embedded systems.

5. ACKNOWLEDGMENTS

The main ideas and the general architecture of the master courses in the Computing field at U.Minho, according to the Bologna Declaration, belong to José M. Valença. The authors would like also to thank fruitful discussions with João A. Carvalho, Pedro Ribeiro, Luís S. Barbosa, António N. Ribeiro, and José F. Campos that somehow are reflected in the proposed degree.

6. REFERENCES

[1] Abran A., Moore J.W. Bourque P., Dupuis R. (eds.), *Guide to the Software Engineering Body of Knowledge: 2004 Edition – SWEBOK*, IEEE CS Press, 2004.

[2] Adams, E. J. *A Project-Intensive Software Design Course*. 25th ACM SIGCSE Technical Symposium on Computer Science Education, Indianapolis, Indiana, pp. 112–6, March 1993.

[3] Coppit D., Haddox-Schatz J.M., *Large Team Projects in Software Engineering Courses*, 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, Missouri, USA, pp. 137-41, ACM Press, 2005.

[4] Fenwick Jr. J.B., and Barry L. Kurtz, *Intra-curriculum Software Engineering Education*, 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, Missouri, USA, pp. 540-4, ACM Press, 2005.

[5] Habra N., Dubois E., *Putting into Practice Advanced Software Engineering Techniques through Students Project*. 7th Conference on Software Engineering Education, San Antonio, Texas, LNCS 750, pp. 303–16, Springer-Verlag, January 1994.

[6] Knight J.C., Leveson N.G., *Software and Higher Education*, Communications of the ACM 49(1):160, January 2006.

[7] McCauley, R. A., Adams, E. J., Gotterbarn, D. J., Northrop,L. M., Saiedian, H. and Zweben, S.. Organizational issues in teaching project-oriented software engineering courses. ACM SIGCSE Bulletin , Fifth SIGCSE Symposium on Computer Science Education, Volume 26, Issue 1, 392-393, 1994.

[8] Meyer B., Zwaenepoel W., *Europe's Computer Scientists Take Fate into Their Own Hands*, Communications of the ACM 49(2):21–4, March 2006.

[9] Northrop, L.M., *Success with the Project-Intensive Model for an Undergraduate Software Engineering Course*. SIGCSE Bulletin 21(1):151–5, February 1989.

[10] Powell P.C., Weenk G.W.H., *Project-Led Engineering Education*, Lemma, Utrecht, The Netherlands, 2003.

[11] Shaw M., *Software Engineering Education: A Roadmap*, In "The Future of Software Engineering", Finkelstein A. (ed.), ACM Press, 2000.

[12] Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE CS & ACM, 2004.

[13] Way T.P., *A Company-Based Framework for a Software Engineering Course*, 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, Missouri, USA, pp. 132-6, ACM Press, 2005.