1. Introduction

The software industry is calling for a growing number of practitioners capable of adapting to a variety of problem types and technologies. This objective can only be achieved by means of open and multidisciplinary training, acquainting future practitioners with a range of artifacts for software development and with criteria for choosing those best suited to the problem in question depending on each individual situation.

At the School of Computer Science, Polytechnic University of Madrid (FI-UPM) in Spain, we have addressed these problems and have restructured the masters in SE and KE to give joint instruction on both disciplines. This integration is practi-
cable thanks to the fact that, although SE and KE were originally two approaches taking different paths, as the problems they address and the software systems construction process have evolved, points of intersection have been found where these two disciplines can feed off each other [15]. These points of intersection involve the interchange of principles, methods and techniques between the two disciplines. This means that each discipline can benefit from the knowledge and experience of the other.

The objective of this joint instruction is to familiarize students with the techniques available for software development and support their application to a particular problem, irrespective of whether it is solved by means of a traditional or knowledge-based system.

This paper presents a particular approach to and the results of this integrated instruction. For this purpose, the paper is structured as follows. Section 2 describes the demands raised by the software industry concerning practitioners and how these can be met by means of joint SE and KE instruction. Section 3 presents the set of development process activities with respect to which the instruction of these two disciplines has been integrated. This integration was not addressed as an instantaneous process; it passed through several stages that are briefly described in Sec. 4. Before detailing the content of the master program in Sec. 6, Sec. 5 briefly describes the teaching/learning philosophy adopted. Then, Sec. 6 gives an itemized description of the master program, including a detailed justification of each module and the subjects they contain. Finally, Sec. 7 presents the quantitative results of this experience, affording an analysis of the impact of joint education on the industry, students and the development projects they undertake.

2. Motivations for Integrated SE and KE Education

A flexible attitude is an essential characteristic of good software systems engineers, as change and innovations will be a constant in their working life. Engineers with a rigid view of software development will be unwilling to include the latest developments in their projects. In 1991, the National Research Council Steering Committee on Human Resources in Computer Science and Technology organized a workshop [7] to discuss the computer science profession. One of the few points on which agreement was reached was that society is demanding more adaptable software practitioners with extensive expertise. Specifically, Joe Kubat of the New York Stock Exchange stated at this workshop that there was a need for malleable people who were open to change capable of moving with technology. People who do not expect to encounter change will soon be caught off-side, he said. Similarly, Barbara Wansley to the National Academy of Public Administration said that they were looking for someone adaptable enough to be taught the latest. Along these lines, Garlan [10] mentioned that the changing environment to which software engineers are exposed means that we not only have to teach the latest fashionable method, we have to provide the skills and analytical techniques required for future practitioners to be able to evaluate emerging technologies and adapt to new industry needs.
Typical examples of rigid practitioners are programmers who are “in love” with a language which they always use for any system without any prior evaluation, or those who want to sell us object orientation, now that it is in fashion, claiming that it can (and must) be used on all occasions regardless of the system type, problem, user and development team, . . . .

If they are to gain a flexible view of development, engineers must be educated in diversity as opposed to singularity. In our opinion, one of the most common and least beneficial attitudes among software developers is “dogmatism”. That is, each engineer “believes” in a particular paradigm, technique, tool, language, operating system, etc., which is seen as a remedy to all the problems of software construction.

Hence, the need arises to redefine instruction on the software development process so as to provide students with a broader and more flexible view of software systems construction. The first step towards this goal materialized as a result of the progressive deployment of integrated SE and KE instruction within the master course taught at the FI-UPM. Joint education seeks to prevent the above mentioned “reductionism”, defending a view in which there are, on the one hand, user needs and, on the other, a host of techniques and solutions, none of which is dominant and all of which have their own ecological niche and are best suited for particular situations or particular needs.

Although, as already mentioned, SE and KE originated as different software systems development disciplines, with time and as the problem types evolved, common ground has been sighted on which the two disciplines can complement each other [15]. This means that techniques traditionally applicable to one discipline can also be used, with only slight modifications, to the advantage of software systems developed using the other approach, thus filling the “voids” or gaps in the second discipline. Hence, integrated SE and KE instruction looks to teach students the techniques proper to both SE and to KE, qualifying them to apply, in practice, the techniques best suited to the type of problem to be solved at any time, irrespective of whether the response to the problem is a traditional system developed using SE or a knowledge-based system (KBS) developed by KE.

Joint SE and KE instruction as a master course is not a very common approach. Indeed, most universities that offer software development master courses provide separate SE courses (Master of Software Engineering at the Carnegie Mellon University, the University of Maryland, the University of Oregon, etc.) or KE courses (Master in Knowledge Engineering at University of Westminster, the University of Pittsburgh, etc.), but none integrate the instruction of both disciplines.

As mentioned above, the provision of joint education in SE and KE aims to prevent the rigid and uncompromising approaches to software development. Hence, the flexible and malleable approach to software development that we seek to convey in the master course involves:

- Software engineers seeing themselves as practitioners who should provide computational solutions to user problems, as opposed to imposing “their” type of software solution on the user, as they would if they took a rigid view.
Engineers dealing in a wide and ductile range of computable solutions, selecting the best suited after having understood the user need (including hybrid solutions that combine different fields of knowledge), as opposed to the rigid view in which developers have a preferred and pre-established solution, where they adapt the problem to the solution instead of adapting the solution to the problem.

Engineers knowing and mastering a variety of methods for building software systems, in such a manner that they look into their “toolbox” and pick the best suited technique in each case, as opposed to the rigid view whereby engineers are proficient in a single development paradigm which they apply to every development project.

An understanding of how the instruction was integrated is given by establishing what SE and KE issues have been integrated. Section 3 briefly describes these issues.

3. Scope of Integrated Education in SE and KE

SE and KE are both disciplines that encompass a host of topics. When we started to address the integration of the instruction, we needed a basis on which to carry out the integration. This foundation was provided by the construction process. That is, instead of separately analyzing which topics could and which could not be integrated, one by one, the software process was used as a general framework by means of which to analyze the individual integrations. So, we analyzed the construction processes in SE and KE, looking to identify what each phase of the two processes had in common and what differences there were with regard to the other engineering discipline. The results of the analysis are presented in [3]; however, a brief outline is given here, since they are the basis for understanding how the training was integrated.

The traditional software process, as per IEEE standard 1074-1991, is composed of four main processes (Software Life Cycle Model Process, Project Management Process, Software Development-oriented Process and Integral Process), each grouping a series of activities that are responsible for implementing their associated goals [14]. These processes are shown on the right side of Table 1.

As far as the KE software process is concerned, although it has not been formally defined by any organization or document, it can be considered to be composed primarily of the processes shown on the left side of Table 1, as all of these activities appear in some form or another in the most well-known KE development methodologies [2, 6, 8, 11, 26].

Table 2 shows the activities that can be exported from one discipline to the other. An arrow at the end indicates which of the techniques that are applicable to the source activity are likely to be exportable to the other discipline, whereas an arrow at the start and at the end means that the activities are complementary in both disciplines (by complementary we mean that both disciplines provide techniques
<table>
<thead>
<tr>
<th>Subprocess</th>
<th>Description</th>
<th>Subprocess</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Software Life Cycle Model Process</td>
<td>Identifies and selects a software life cycle.</td>
<td>• Feasibility</td>
<td>In particular, it evaluates whether the problem addressed can be solved using a KBS.</td>
</tr>
<tr>
<td>• Project Management Process</td>
<td>Creates the framework for the project and ensures the appropriate level of management throughout the entire product life cycle. It is composed of the Project Initiation, Project Monitoring and Control, and Software Quality Management subprocesses.</td>
<td>• Knowledge Acquisition</td>
<td>Extracts all the knowledge needed to understand the domain, the problem and the problem-solving process from any public sources (i.e., books, documents, etc., by means of document analysis, classification, etc.) and private sources (an expert in the domain, by means of protocol analysis repertory grid, etc.).</td>
</tr>
<tr>
<td>• Software Development-oriented Processes</td>
<td>Produce, install, operate and maintain the software and retire it from use. They are classed as Pre-development Processes (Concept Exploration and System Allocation), Development Processes (Analysis, Design and Implementation) and Post-development Processes (Installation, Operation, Support, Maintenance and Retirement).</td>
<td>• Conceptualization</td>
<td>Analyses and synthesizes the unstructured knowledge acquired in the knowledge acquisition phase which is organized in a conceptual model.</td>
</tr>
<tr>
<td>• Integral Processes</td>
<td>They are necessary to complete software project activities successfully. They ensure the completion and quality of project functions. They are carried out at the same time as software development-oriented processes and include activities that do not produce software but are absolutely necessary to obtain a successful system. They cover the processes of Verification and Validation, Software Configuration Management, Documentation Development and Training.</td>
<td>• Formalization</td>
<td>Selects a formalism (e.g., rules, frames, cases) to represent the knowledge conceptualized in the preceding stage in a formal model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Implementation</td>
<td>Codes the programs that make up the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Validation</td>
<td>Assesses the KBS and its components (i.e., KB, inference engine, user interface, etc.) against expected quality levels; for instance, by comparing KBS responses with expert responses.</td>
</tr>
</tbody>
</table>
that can be used in the other to improve the performance of the activity in question). Similar processes in the two disciplines are linked by a plain line.

A brief explanation of Table 2 is given below. Section 6 offers a deeper discussion, justifying in detail the content of the joint SE and KE program, which has been designed.

It follows that SE’s possibly biggest contribution to KE concerns issues related to management, such as estimation, planning, configuration management or project documentation. This is because, over the years, SE has defined a series of activities for building software that go beyond the merely technical (analysis, design, implementation). SE has added to these technical activities other tasks which bring software construction closer to being an engineering discipline. If KE is to realize its name, it will certainly have to consider all these activities. This is true as of the time when KBSs leave the research laboratory and go into a marketplace in which users are demanding quality, reliability, time and cost compliance, satisfaction and adaptation.

On the other hand, KE can make sizeable contributions to the traditional software development process with regard to the feasibility, knowledge acquisition and conceptualization fields. Related to the feasibility study, in KE, once the problem has been defined or the need has been identified, a feasibility study is carried out in order to decide whether the problem can be addressed by KE. This special feasibility study emerges as a result of the need to ascertain which problems can be processed by KBS and which by conventional software. In an integral software process that can be used by software and knowledge engineers, after exploring the domain and gaining an understanding of the problem posed, it is essential to include both the cost/benefit analysis of alternative approaches to a particular project and activities that make up the special feasibility study in order to decide on the type of software solution required by the problem.
Regarding knowledge acquisition and conceptualization, the trend in SE is towards the need to solve poorly specified, increasingly complex problems with shallow knowledge of the problem domain. Hence, SE will be dominated by open and dynamic requirements [5]. This situation brings the SE requirements process closer to the knowledge acquisition and conceptualization process employed in KE. Therefore, SE can make use of techniques traditionally applied in KE, for example, repertory grid analysis or protocol analysis for knowledge acquisition; or for example, task trees or decision tables to model particular points of the user world. Furthermore, techniques traditionally applied to traditional software development in SE, such as DFDs or scenarios, can be applied to particular problems in KE.

As far as design and formalization issues are concerned, although the fundamental conventional design principles (modularity, abstraction, etc) can be applied in both SE and KE, it is not feasible to export particular techniques for performing these activities, as they are very specific to the type of software to be built.

An evaluation framework was defined for the purpose of integrating issues related to verification and validation in SE and KE [16]. It establishes the types of evaluation to be conducted, when they are to be conducted and what techniques are to be used. Even though it is common to both types of software, the framework may contain techniques usable in both cases and techniques peculiar to one type of software.

These ideas on integration were used as a basis for designing a joint educational program in which the development process to be followed in the two disciplines promotes the use of activities proper to both SE and KE, regardless of the type of software system to be built. The design and detailed justification of this program is shown in Sec. 6. However, it was implemented as an incremental, rather than a one-off, process, looking to gradually integrate the instruction of the two disciplines. This gradual process is shown in the next section.

4. Progressive Approach to the Integrated SE & KE Instruction

The FI-UPM has offered master courses in information systems since 1978 and, more particularly, in SE or KE since 1987. The two SE and KE master programs were developed in parallel, each with a total of 600 teaching hours, intermixing theory and practice classes. Finally, students were expected to develop a practical project in the respective speciality. This original situation is shown in Fig. 1(a). That is, these were the two master courses in either SE or KE, much like many others taught at universities all over the world (some examples of such courses were mentioned in Sec. 2).

In 1993, we considered the possibility of transferring emerging ideas on SE and KE integration to the master course. In this respect, we sought to go further in the direction proposed by some authors, like Somerville [21], of supplementing methodologies with modeling techniques from outside to satisfactorily represent certain problem issues. First, we incorporated new techniques not only at the level of modeling, but, as we will see later, throughout the entire development process; second,
the above techniques were not taken from just one branch of software development, either SE or KE, we used techniques from KE that could be applied to supplement an SE methodology and vice versa.

In this manner, we sought to educate practitioners who were more adaptable to user and market needs (as mentioned in Sec. 2), and equally capable of developing a traditional software system, an expert system or a hybrid software solution. The manner of carrying out this integration was far from obvious, especially as there were no similar experiences to draw upon. We decided, therefore, to integrate by means of successive stages. Since then the two masters have been gradually integrated, as we came to understand how the issues could be related.

The first stage (1993 and 1994 academic years) consisted of introducing some basic subjects taken from one discipline into the master addressing the other area, and the two courses retained their own, separate identities. In this manner, students would gain an in-depth understanding of one discipline and would also have some notions of the other; that is, students were provided with knowledge of one discipline and merely information about the other. This situation is shown in Fig. 1(b).

Students were acquainted with basic notions of the other discipline in a few classes held at the end of the course, merely introducing the problems addressed by SE or KE, respectively, and giving a brief description of the development process to be used, and some of the most commonly used techniques in this process. The objective of this phase was to make students aware that there was another type of software system. Students taking the Master in KE could be assumed to know about traditional systems, as these systems are more common than KBS. However, the arrangement shown in Fig. 2 was maintained to standardize the structure of the two master courses.

In the second stage (1995 and 1996 academic years), the masters were unified at lesson level as shown in Fig. 1(c); that is, the curriculum was the same and was taught to students of both masters at the same time. This curriculum was composed
of specific subjects concerning the construction of traditional systems, followed by subjects addressing KBS construction. That is, the construction process of each software type was presented in detail in each part of the course, describing the techniques and methods proper to each one separately.

In this manner, we assured that the level of teaching was the same for both disciplines. However, the level of learning attained by the students was not the same, as the practical project completed at the end of the course addressed only one discipline. In other words, the course provided knowledge about both disciplines, but students attained practical knowledge of only one, whereas their knowledge of the other was merely theoretical. (This, however, would allow them to put their theoretical knowledge into practice at any point during their professional careers.)

The teaching of SE and KE could not be said to be integrated at this point. It was really a sequential education in both; that is, first all the SE and then all the KE was taught. It was up to students to form an integrated view, once they had acquired knowledge of both. Even this very primitive integration produced students with a more flexible approach to development than the traditional system of separate teaching. This was indicated by the fact that, for example, some students spontaneously used techniques from the other discipline in their practical projects, as and when they were needed. That is, students had somehow internally related SE and KE knowledge, as they resorted to one or the other as they required. The main areas where this integration took place were: (1) use of knowledge acquisition

Fig. 2. Third approach to teaching integrated SE and KE.
techniques for gathering information from the user during the requirements phase for traditional software development; (2) use of SE structured analysis representations (DFDs for example) to sometimes represent high-level strategic knowledge in KBS development; and (3) use of project management techniques (for example, resource planning) during KBS development.

We have now (1997, 1998, 1999 and 2000) reached the third stage of integration where we have tried to apply the ideas detailed in [2] and summed up in Sec. 3. This stage is shown in Fig. 2, where:

- The references to SE or KE have been removed from the name of the modules making up the master, which now go by general names of the content: Project Management, User Problem Analysis, Software System Design, etc.
- The management module possesses some contents taken from SE that are directly applicable to KE, for example, Configuration Management and Quality Management. Other contents are similar; however, there are no equivalent techniques for KE. For example, the project estimation techniques, Function Points or COCOMO, are applicable only to traditional software projects.
- The problem analysis module is the most integrated module in which we have managed to include subjects following the logical order of software development (elicitation, conceptual modeling, requirements specification) and in which the content of some subjects is taken from SE and that of others from KE. The contents have, however, had to be generalized in such a manner that the explanations omit the source of the techniques or ideas explained and are applicable to any system type. In this manner, the border between SE and KE has been erased.
- The design module is the least integrated, and the two disciplines are still taught sequentially. This is logical today as there are actually a series of different approaches to design and coding, which cannot be integrated: structured design, object-oriented design or KBS design. The most integrated subject in this module is testing. As discussed in Sec. 3, instruction on this subject involves a general-purpose framework valid for evaluating both traditional systems and KBS, as well as the different types of potentially applicable tools.
- The methodology module presents methodologies that guide engineers in the construction of traditional and knowledge-based systems, respectively. There is no development methodology applicable to both system types as yet, which means that two different methodologies have to be taught. However, the teaching process stresses the fact that these methodologies, whether applicable to SE or to KE, can be supplemented with tasks proper to the other discipline.
- The practical projects are still completed in the specialty in which the degree is awarded, but students are now encouraged to include techniques or activities from other disciplines, such as specific techniques for knowledge acquisition, or the activities of project management or software configuration.
The present make-up, and the reasoning followed to determine the content of each Master in SE and KE module, according to the third stage explained above, is presented in detail in Sec. 6. However, before addressing this point, a brief summary of the teaching/learning philosophy used in this configuration is given in Sec. 5.

5. Teaching/Learning Strategy

As a result of the master program, students should understand software systems construction as a process in which the real-world problems, information technology, people and organizations interact. That is, we seek to go beyond the traditional pedagogical method of teaching a series of techniques, methods and tools. With this objective in mind, the master program was structured based on the skills learning process proposed by Anderson [4]. Anderson specifies that, generally, learning has to pass through three stages:

1. Cognitive stage, in which a description of a procedure is learnt.
2. Associative stage, working with a method to develop the skills learnt in the preceding stage.
3. Autonomous stage, where the skills learnt become quicker and automatic.

In order to address these three stages in the software construction process, a gradual approach to learning is required. For this purpose, learning was scheduled as follows:

- Firstly, future practitioners have to learn the basic principles of software development to then be able to practice the techniques that are applied in each activity of this development process. At this level, students practice the application of each technique separately, completing simple exercises by means of which they are familiarized with the concepts to be employed. This process can be matched with the cognitive stage of skill acquisition.
- Once students have understood how the different techniques can be used in software development separately, they then need to interiorize their interrelation within the construction process. Under the master programs, this is done by learning a particular development methodology. Methodology means a complete set of phases, techniques and products that cover the entire software development process. An example of methodologies are: the Spanish Administration’s Métrica v. 2.0 [17], SSADM used by the British Administration [20], the French Administration’s Merise [24], or the methodology used by the US Department of Defense [18]. Teaching a methodology allows students to gain an integrated and consistent view of the development process. For students to be able to interiorize this knowledge, they need to address practical cases that are equivalent in complexity to a simple project in professional practice. Therefore, the methodology is taught on the basis of case studies solved jointly by the tutor and students in the classroom. In this
manner, students, led by the tutor, become versed in carrying out the entire development process using a host of interconnected techniques. This learning process corresponds with the associative stage of skills learning.

- Finally, in order to address the autonomous stage and reinforce the knowledge acquired in the preceding phases, students should apply the above knowledge to a real case of average or high complexity. For this purpose, students must be able not only to apply one of the methodologies taught and their respective techniques, as learnt in the preceding stage, but should employ the above methodology bringing in additional techniques applicable to the problem addressed, taken either from SE or KE. In order to implement this integration process, future practitioners have to be able to evaluate the techniques available and select the best suited in each case. Note that this is the culmination of teaching integrated SE and KE, where it is the students who, on the basis of the knowledge and guidelines with which they were acquainted in earlier stages and assisted by a tutor, perceive the need and usefulness of applying artifacts proper to one branch of software to another. When the techniques are taught, the tutor explains their usefulness for different types of software; however, these concepts are really learnt by students in this last stage, as a result of their practical application.

6. Contents of the Integrated Master in SE and KE

This section details the contents of each Master in SE and KE module, as established by the third stage of integrated SE and KE instruction (described in Sec. 4). These are justified taking into account the intersection of the different development process activities of both disciplines described in Sec. 3. These contents are structured according to the philosophy described in Sec. 5.

The master has been structured according to the modules shown in Fig. 3. The first three modules describe the techniques to be applied in software development, and corresponds to the cognitive stage described in the preceding section.

Module IV describes methodologies which set out the criteria and times for using the techniques presented in earlier modules, depending on whether a system is developed by means of SE or KE. In this manner, we aim to take up the concept of methodology as providing a series of procedures which guide engineers throughout the construction of software systems. This module would address the associative stage of skills learning.

Corresponding with the autonomous stage of learning, students are expected to develop a real conventional system or KBS project in Module V, applying the knowledge acquired in earlier modules.

As a complement to the preceding modules, Module VI seeks to go deeper into various issues related to software construction not traditionally addressed as part of the development. These issues include questions such as: usability, process maturity,
Fig. 3. Content of the Master in SE and KE.

reeengineering, knowledge management, etc. This module is divided into a series of seminars held simultaneously with Module V. This module is associated with the cognitive stage, as the instruction is given by means of theory and practice classes and short practical exercises.
6.1. **Project and product management**

The aim of the Project and Product Management Module is to ensure that students are able to apply project management procedures, which are an aid for completing projects on time and within budget, while satisfying customer requirements. Students are also acquainted with configuration management techniques for managing the products output during the development process and procedures for assuring that products are of the required quality.

As shown in Table 3, the techniques to be used during this module have their root in SE. For example, the selection of a life cycle model for a project enables software engineers to organize, plan, support, budget, program and manage a software project, and prescribes what documents are to be produced. It is reasonable to assume that KBS development also needs to be organized, planned, supported, budgeted, programmed and managed, and this process should, therefore, be exported to KE. This means that the software life cycle for the SE construction process can be useful for the process of building KBS. Obviously, as KBS have their own particular characteristics, the range of life cycle models from which to choose to develop this type of systems would differ from those used for conventional software. Indeed, traditionally, the life cycle models most used in KE are incremental development and prototyping.

Development projects have to be estimated, planned, monitored and controlled in order to commercialize software systems, that is, controlled resource consumption and performance of the commitments entered into with the customer is also of interest in KBS projects. Therefore, project management activities should be carried out to construct software as part of both SE and KE projects. Although there are no estimation techniques, like SE’s Function Points, proper to KE, the philosophy and the importance of management processes is applicable to both SE and KE. This is another point on which SE can make an important contribution to KE.

<table>
<thead>
<tr>
<th>Table 3. Project management module subjects.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Module I: Project and Product Management</strong></td>
</tr>
<tr>
<td>Strategic Information Systems Planning (3 hours)</td>
</tr>
<tr>
<td>Software Process (9 hours)</td>
</tr>
<tr>
<td>Quality Control (9 hours)</td>
</tr>
<tr>
<td>Configuration Management (9 hours)</td>
</tr>
<tr>
<td>Project Estimation and Planning (12 hours)</td>
</tr>
<tr>
<td>Team Management (3 hours)</td>
</tr>
</tbody>
</table>

Configuration management and quality management are a response to the problems that arise when large systems are built. These problems can be expected to recur in SE and KE when systems pass from programming-in-the-small to
programming-in-the-large [23]. Traditionally, these tasks have emerged in SE; however, their application in KE is also important, and these techniques are directly applicable to KE projects. So, these activities could also improve the results of KBS construction.

Most software development methodologies center on technical processes [20, 24], relegating management processes of both the project and the products output to a second plane. This is highlighted in this module to make students aware of this problem and ensure that they do not underestimate their importance. Indeed, students are obliged to include these activities in the practical project which they develop in the final module, even if they are not explicitly laid down by the chosen methodology.

The hours specified in the tables for the modules related to the cognitive stage (Tables 4, 5 and 6) reflect teaching hours. Students have to spend about as many hours again on the practical exercises that are set during the theory classes. These exercises involve students practicing each technique separately, irrespective of the others.

6.2. Problem analysis

The Problem Analysis Module provides a series of techniques for capturing, understanding and representing the user needs, that is the problem that is to be solved by the software and how it is usually solved today. As shown in Table 4, the subjects come from both disciplines.

As specified above, KE can make a significant contribution to SE in the fields of information and knowledge acquisition and conceptualization. The main reason for this is the idiosyncrasy of the problems that KE has traditionally faced: problems in complex domains, with non-algorithmic and non-explicit problem-solving processes. Compared with a traditional software engineer, a knowledge engineer would need a lot more time and effort to understand the domain and the expert problem-solving process. As the expert has internalized this process and it is not explicit, elicitation is a demanding and difficult task. Additionally, once the knowledge engineer gains an insight into the expert’s picture of the world and the pieces

<table>
<thead>
<tr>
<th>Table 4. Problem analysis module subjects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module II: Problem Analysis</td>
</tr>
<tr>
<td>Information and Knowledge Acquisition (12 hours)</td>
</tr>
<tr>
<td>Feasibility Analysis (6 hours)</td>
</tr>
<tr>
<td>Conceptual Modeling in Information Systems (12 hours)</td>
</tr>
<tr>
<td>Conceptual Modeling of Expert Knowledge (12 hours)</td>
</tr>
<tr>
<td>Software Requirements Specification (6 hours)</td>
</tr>
<tr>
<td>Formal Specifications (12 hours)</td>
</tr>
</tbody>
</table>
start to fit together, he/she must start to conceptualize all of the knowledge being elicited and make it explicit in a problem model. That is why this discipline has traditionally developed specific techniques for interacting with the user and conceptual models of the problem that are independent of the software solution that will solve such problems.

In view of the increasing complexity of the problems addressed by SE, there is a need to go deeper into domain and problem exploration and comprehension and, as a result, the software engineer must interact closely with the current problem solver to understand the problem-solving process and the application environment. System analysis requires intense communication between the customer and the analyst. The customer must understand the system objectives and be able to explain them clearly. The analyst must know what questions to ask, what advice to give and what investigations to undertake; in short, which is the best way of extracting information. If communication breaks down, the success of the entire project is jeopardized. Although the domains traditionally addressed by SE are better known, the problems are easier to understand and the problem-solving processes are often explicit and known algorithms, now that SE is faced with the challenge of solving more complex, albeit algorithmic, problems, the time spent interacting with the user as opposed to interacting with the computer is growing. Indeed, the activities of information or knowledge acquisition are becoming increasingly important within the analysis phase.

Once the problem has been defined or the need has been identified, a feasibility study is carried out in order to decide whether it is possible to provide a solution and, if so, whether commercial software exists that can solve the problem (or part of it), whether existing software can be reused or whether the software has to be developed from scratch, in which case we have to determine whether the problem can be addressed by KE or by SE, or by both in the case of an integrated system.

Having identified the approach to be taken to the problem (KE or SE), analysts must start to conceptualize all the knowledge being elicited and make it explicit in a conceptual model. This module describes both modeling techniques proper to SE (DFDs, STDs, OOA, etc.) and techniques proper to KE (task trees, knowledge maps, etc.). It aims for students to acquire an overview of the techniques to be employed and to which problem type they are best suited. This is another important point in the integration between SE and KE. KE can make a significant contribution in this field, as it has a diversity of techniques for representing the problem domain in a manner closer to the user world. The techniques employed by SE to analyze the problem are traditionally linked more with the machine world, as discussed by Hoyldavisk [12]. (In a future, fourth integration stage, we plan to apply a problem understanding framework, on which we are now working, based on describing the basic activities for the developer to understand the problem raised by the user independently of the development approach (structured, object oriented, KBS, etc.), which he/she will choose to solve it. Having understood the problem, the developer will be able to determine which is or are the best approaches to problem solving.)
One of the problems traditionally generated by the analysis process is a requirements specification document. This document contains a description of the external behavior expected of the system that is to meet the user needs. In SE, there are some standardized models such as the IEEE standard [13]. By contrast, the format of this document is less standardized in KE, and it is not even drafted at all in many KE projects. There is an increasing tendency to make distinctions between requirements that cannot be specified at the start of KBS development due to poor problem structuring, and other requirements that are typical of any software system and can be specified [22].

With regard to the formal specifications, they have their origin in SE; however, as they are logic-based, it is to be expected, and indeed there is experience in this respect, that at least part of the expert knowledge can be transcribed directly from the conceptual model to a formal specification [9].

6.3. **Software systems design**

After problem analysis, the aim of the development process is to determine the best means of satisfying user needs by means of a software system. Although the basic design principles used in SE are also applicable to KBS, the techniques used during this process are more specific to each type of approach, either SE and KE. Specifically, SE techniques applied to design and implement traditional software systems are described first, followed by the techniques provided by KE to design and implement KBS. Table 5 shows the subjects of this module.

With regard to validation and verification-related issues, as specified in Sec. 3, a common framework has been developed for integrating the techniques and procedures for SE and KE evaluation. This general-purpose framework establishes the principles governing a software evaluation, and divides the software system evaluation issues into four levels: correctness (software internal or form level), validity (semantic or software content level), usability (software relationship level) and use-

<table>
<thead>
<tr>
<th>Table 5. Traditional systems and intelligent systems design module subjects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module III: Software Design</td>
</tr>
<tr>
<td>Structured Software Design: Functions (15 hours)</td>
</tr>
<tr>
<td>Structured Software Design: Data (15 hours)</td>
</tr>
<tr>
<td>Development Support Tools: CASE (12 hours)</td>
</tr>
<tr>
<td>Object-Oriented Software (15 hours)</td>
</tr>
<tr>
<td>Validation and Verification (15 hours)</td>
</tr>
<tr>
<td>Programming Environments (12 hours)</td>
</tr>
<tr>
<td>Knowledge-Based Software Design: Control Reasoning (21 hours)</td>
</tr>
<tr>
<td>Knowledge-Based Software Design: Representation and Inference (21 hours)</td>
</tr>
<tr>
<td>Knowledge-Based Software Design: Uncertainty (6 hours)</td>
</tr>
<tr>
<td>KBS Construction Environments (12 hours)</td>
</tr>
</tbody>
</table>
fulness (organizational level). With regard to the type of techniques to be used to perform this verification, students are familiarized with different approaches (code testing, Turing Tests, Anomaly Detection, etc.) so that they can select the best suited depending on each situation, irrespective of whether the software to be tested is based on a traditional system or KBS.

The modules described above are part of the cognitive stage of learning, the function of which is for students to practice applying different techniques. For this purpose, apart from theory classes, exercises are set on fairly simple problems. However, the knowledge and skills acquired at the technical level are insufficient for future practitioners to exercise their profession. There is a higher level at which students must be able to gain an overview of the construction process, and how the different development process activities interrelate. This view is acquired in the following two modules.

6.4. **Software construction methodologies**

As specified above, students must learn and be able to use a series of techniques, and not until they are proficient in their separate use should they go on to learn how to conjugate and co-ordinate all the elements at a higher level (associative stage of learning). This higher level is what leads them to develop a full project. It is the Methodologies Module that lays the foundations for students to be able to move on from learning separate techniques to developing their own real projects.

The concept of methodology as an overview has been traditionally employed in European standards, for example, SSADM [20], MERISE [24] or MÉTRICA [17]. A methodology must be understood as an ordered series of phases, phase transition criteria, techniques to be used in and products to be output by each phase. Some of the approaches on the market considered methodologies would not be such according to the definition used here. This is the case of OMT [33], for example, or Yourdon’s methodology of analysis [25], which would be a more properly named method.

These master modules introduce students to two methodologies for traditional systems (MÉTRICA v. 2, as established by the Spanish administration, for structured development; and the method proposed by Larman [19] for object oriented systems) and another for KBS (IDEAL [1]; the KADS method [26] was not considered as it is merely a declarative method, which does not specify, step by step, the tasks and activities to be performed by the developer, that is it is not a procedural method). Both include a theory and practical side, in which students are expected to practice applying each methodology. The complexity of the problems set is higher than in the modules addressing techniques. Usually it represents a simple project in professional practice, having fairly clear and stable requirements. These cases are studied in the classroom, and students are advised by the tutor as project development progresses. In this manner, students are not left alone with the problem, they are supported and guided by the tutor at all times.
6.5. Practical SE and KE project
Students complete their master by developing a practical SE or KE project. Students are expected to develop a real project of medium to high complexity for real customers and users, using the above methodologies.

Students will work on their own on this project in such a manner as they apply and build on their previous learning. Students are now encouraged to use the techniques that they consider most appropriate, whether from SE or KE, even if they are not explicitly used in the respective methodology. Thus, they put into practice the process of integration that has been outlined during the cognitive stage modules. This integration is possible thanks to the flexible and malleable view of software development that we tried to teach students during the master course.

In this manner, we seek to familiarize students with a wide repertoire of techniques and assure they know the best scope of application for each one. This is what we referred to as the “toolbox” in Sec. 2. So, for each project, the student will use an instantiation of the available techniques, depending on the problem addressed and irrespective of their origin. This involves evaluating the different techniques, for which purpose students must have a sound knowledge of the characteristics of the problem to be solved and the features of the technique under evaluation. This process encourages students to develop their critical judgment and decision-making abilities.

6.6. Advanced SE and KE concepts
As Module V advances, a series of seminars are held which address various points of software systems construction that are difficult to bring into the instruction on development proper. These seminars are what make up Module VI. This module seeks to familiarize students with some of these concepts. Table 6 outlines the subjects taught in this module. However, this module is highly variable as it encompasses seminars and talks on new issues emerging every year.

<table>
<thead>
<tr>
<th>Module VI: SE and KE Advanced Subjects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance (6 hours)</td>
</tr>
<tr>
<td>Reuse (3 hours)</td>
</tr>
<tr>
<td>Reengineering (3 hours)</td>
</tr>
<tr>
<td>Usability (3 hours)</td>
</tr>
<tr>
<td>Process Maturity (6 hours)</td>
</tr>
<tr>
<td>Knowledge Managements (3 hours)</td>
</tr>
</tbody>
</table>

7. Evaluation of the Experiences
Data have been collected over the years, allowing us to visualize the changes taking place in student learning. However, it is extremely difficult to quantify student
learning and, especially, quantify the usefulness of such learning for job performance. In this case, we opted for assessing a series of factors that would suggest: the level of satisfaction of industry, the level of satisfaction of students, the level of integration (SE and KE) in the practical projects.

Industry satisfaction was evaluated by means of questionnaires addressed to the managers of organizations at which the students were placed (students usually work as software developers, where they supposedly have to apply the knowledge acquired during the master course). Figure 4(a) shows the decomposition of the satisfaction issue into more specific criteria. The questionnaire corresponding to the criteria is shown in Table 7.

Students evaluate different points of the course throughout its duration. However, Fig. 4(b) shows only the points affecting the subject addressed in this paper. Table 8 shows the questionnaire corresponding to the criteria for evaluation.

---

**INDUSTRY SATISFACTION**

- **Knowledge**
  - Sufficient for job performance
  - Insufficient to user problems
- **Flexibility**
  - With respect to job performance
- **Opinion**
  - With respect to user problems

---

**STUDENT SATISFACTION**

- **Course**
  - Instruction appears to be integrated or disintegrated
  - Logical coherence in subject order
- **Knowledge**
  - With respect to quantity
  - With respect to knowledge of the other discipline
- **Results**
  - With respect to user problems
  - With respect to specialization

---

**Fig. 4.** Criteria used in the industry and student evaluations.
Table 7. Manager questionnaire.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you think that your employee is skilled enough to perform the job?</td>
<td>No Yes</td>
</tr>
<tr>
<td>Do you think that your employee is sometimes lacking the knowledge</td>
<td>Very often</td>
</tr>
<tr>
<td>to perform the job?</td>
<td>Never</td>
</tr>
<tr>
<td>Do you think that your employee’s training is flexible enough for</td>
<td>Unable</td>
</tr>
<tr>
<td>him/her to address different user problem types?</td>
<td>Able</td>
</tr>
<tr>
<td>Does your employee’s training allow him/her to move from job to another</td>
<td>Finds it</td>
</tr>
<tr>
<td>job easily?</td>
<td>easy</td>
</tr>
<tr>
<td>What is your overall opinion on your employee’s training?</td>
<td>Very poor</td>
</tr>
<tr>
<td></td>
<td>Very good</td>
</tr>
</tbody>
</table>

Table 8. Student satisfaction questionnaire.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you think that the instruction on SE and KE is integrated enough?</td>
<td>Very disintegrated Very integrated</td>
</tr>
<tr>
<td>Do you think the logical sequence in which the subjects of the modules</td>
<td>Very disordered Very logical</td>
</tr>
<tr>
<td>are taught is adequate?</td>
<td>Insufficient Sufficient</td>
</tr>
<tr>
<td>Do you think that the instruction is sufficient for undertaking a</td>
<td>Yes</td>
</tr>
<tr>
<td>software system construction process?</td>
<td>No</td>
</tr>
<tr>
<td>Would you have preferred instruction on only SE or KE for reasons of</td>
<td>One type only Both</td>
</tr>
<tr>
<td>“clarity”? (for stages 2 and 3)</td>
<td></td>
</tr>
<tr>
<td>Would you have liked to have learnt about the other discipline? (for</td>
<td></td>
</tr>
<tr>
<td>stage 1)</td>
<td></td>
</tr>
<tr>
<td>Do you feel able to address both traditional systems and KBS?</td>
<td>One type only Both</td>
</tr>
<tr>
<td>Do you feel you have specialized in either discipline?</td>
<td>SE or KE No</td>
</tr>
</tbody>
</table>

Finally, the issue of students applying integration in practice is evaluated by observing their master dissertations and checking whether or not they address the different points that we sought to integrate. Table 9 shows the issues to be checked in each dissertation, according to whether this is classed as a SE or KE development project.

This type of evaluation has been performed since 1992. Figure 5 shows the results of the evaluations over the years. Both the years and the master integration stage (as discussed in Sec. 4) are shown along the x-axis, and the results of the evaluations are shown along the y-axis. As none of the evaluation issues (industry, students, dissertations) are weighted and the maximum value for each point is 10, the maximum values of the different evaluations are:
- Highest industry satisfaction value = 50
- Highest student satisfaction value = 60
- Highest dissertation satisfaction value = 40

The reflections on the results of the evaluations are as follows. With regard to industry, managers did not start to observe more flexibility among students until the instruction was integrated. Acquainting students with the basic notions of the other discipline is useful only for purposes of information and not as training. That

<table>
<thead>
<tr>
<th>Issue</th>
<th>Traditional Software Development (SE)</th>
<th>KBS Development (KE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Were knowledge acquisition techniques used to interact with the user?</td>
<td>Yes (10) No (0)</td>
<td>Was the project estimated and planned?</td>
</tr>
<tr>
<td>Was the problem environment, the problem-solving information, etc., conceptually modeled?</td>
<td>Yes (10) No (0)</td>
<td>Was configuration management applied?</td>
</tr>
<tr>
<td>Does the system have integrated KBS and SE modules?</td>
<td>Yes (10) No (0)</td>
<td>Does the system have integrated KBS and SE modules?</td>
</tr>
<tr>
<td>Was the general software evaluation framework used?</td>
<td>Yes (10) No (0)</td>
<td>Was the general software evaluation framework used?</td>
</tr>
</tbody>
</table>

Fig. 5. Result of the evaluations.
is, the knowledge taught was not applied in practice, since students viewed these classes as informative seminars, because they were not integrated into the central program of the course.

With regard to student satisfaction, the provision of instruction on basic notions worsened the situation, as it opened up new horizons for students, who felt more specialized and discovered that their knowledge was incomplete. On the other hand, although student satisfaction increased during the third stage, especially with regard to the question of whether they felt specialized, the response to the question regarding instruction worsened. That is, this third stage turned out, as we expected,
to be more difficult with regard to the contents, since the alternation of SE and KE sometimes blocked the attainment of a gradual view. This can be mainly put down to the teaching staff. Many of the professors find it somewhat difficult to teach techniques proper to one of the disciplines with a view to their application to the other. This means that the contents of the subjects also need to be reworked so as to remove, as far as possible, the flavor of the origins of the techniques.

Finally, with regard to the software projects developed by the students to complete the course, again mere information (first stage) is not sufficient for improvements to be observed in the practical projects. The only improvement from the joint, albeit sequential, instruction of the two disciplines (second stage) was that there were more projects containing SE and KE modules; however, save a few exceptions, these modules were built according to the methods and techniques proper to each engineering discipline. Students are only able to reproduce the patterns of integration taught in their practical projects after receiving instruction on techniques integrated into a generic process (third stage).

Quantitatively, the results of the integration at project level are as follows. No management techniques were applied in the practical projects developed using KE in the first stage, whereas they were included in only 5% of these projects in the second stage. On the contrary, they were addressed in all the stage 3 projects, regardless of whether the systems were built using SE or KE. Also, analysis techniques (mainly acquisition and conceptualization) are applied as a complement to the techniques supplied by the different methodologies used in 75% of the projects developed during the third stage of integration, whereas these techniques were used only in 10% of the projects developed in the second stage and in no first stage projects.

With regard to the application of the common verification framework, this is used in 85% of the projects developed in the third stage (it was not taught in the earlier stages, so it was not applicable).

It follows from the above data that the proposed integrated instruction is workable, and we trust that integration will increase progressively as it is internalized by the professors.

8. Conclusions

It is important to note that training and education shape a particular "structure of thought" which marks the world view that students will have from then onwards. Teaching molds students' reasoning. Rigid teaching generates a rigid view of problems and their solutions; monochromatic teaching leads to a permanent one-color view of the world. If we want to generate flexible practitioners who are prepared for change (new techniques, new problems, new computers, etc.), we must provide flexible and moldable instruction, allowing students to understand that other problem-solving methods can and, indeed, do emerge, and these will partially cover the user problem space.

This paper presented a possible approach towards instruction. This approach is based on an integrated instruction of SE and KE, which has been applied in the
Master in SE and KE at the FI-UPM. The scope of the integration applied centers mainly on project management, problem analysis and testing.

This integration was carried out gradually by means of three stages described in Sec. 4. We do not think that the third stage of integration, as detailed in this paper, is the last along this road, and we have plans to continue integrating new concepts as the software development process evolves and we understand how to implement this process of integration. This is a continuous process, by means of which we can adapt the instruction taught to future practitioners dynamically as software development advances.

A three-level evaluation (industry, students and projects) was conducted for the purpose of quantitatively analyzing the results of this experience. The main results at the industry level show that managers have perceived an improvement in the flexibility of the students who have taken the stage 3 master. However, with regard to student satisfaction with the instruction, although they are generally more satisfied than in earlier stages, they remark on the bigger effort required to learn and assimilate the contents. Finally, the integration concepts taught have been successfully applied in both the SE and KE practical projects developed. In this manner, the main conclusion drawn from this experience can be considered to be that students take a very positive attitude and are able to understand and support the philosophy of integration.

Some of the students have even contributed significantly to the integration. For example, as a result of this program, research is being conducted into setting up an integrated methodology for building both KBS and conventional systems. The definition of the development process is now complete [3]. Once the methodology has been empirically tested, there are plans to replace Module IV: Software Construction Methodology, in which two methodologies are now taught, one for conventional systems and another for KBS, by a module in which instruction is given on the guidelines and procedures proposed by this new integrated methodology.

This experience shows that the integration of SE and KE advocated by some researchers can be carried out in practice, if a flexible and open training is provided addressing the techniques of both disciplines. This paper presents one possible approach for teaching integrated SE and KE. There are many approaches to integration. The important point is that by teaching methods and techniques from other disciplines, we are giving future engineers an open and flexible view of software development. This extensive and malleable approach enables them to better address the increasingly more complex and innovative problems and needs raised by users.

References


