22nd Conference on Software Engineering Education and Training

A Requirements Engineering and Management Training Course for Software Development Professionals

João M. Fernandes
Dept. Informática
Universidade do Minho
Braga, Portugal
jmf@di.uminho.pt

Ricardo J. Machado
Dept. Sistemas de Informação
Universidade do Minho
Guimarães, Portugal
rmac@dsi.uminho.pt

Stephen B. Seidman
College of Natural Sciences and Mathematics
University of Central Arkansas
Comway AR, USA
sseidman@uca.edu

Abstract

Devising a course for software professionals working in industry depends on several factors. In order to create a course that fulfils professionals' expectations, it is important to take account of the skills of the participants, the time available, and the specific topics to be covered. This paper presents the curriculum of a course in requirements engineering and management intended for software developers with a first-level academic degree in computing and experience in developing real software solutions. This context requires the course to concentrate on topics that were not taught in the participants' previous education and that can have a positive impact on their daily practices.

1. Introduction

Typically, after three to five years of employment in software development, professionals realize that they need structured insights into specific software engineering topics. Until this point, topics such as requirements engineering, project management, and quality are typically not seen as important, because early-career software development professionals are essentially technology-driven.

In particular, software requirements is one of the eleven Knowledge Areas covered by the IEEE Computer Society's CSDP (Certified Software Development Professional) certification examination. The CSDP exam is designed to assess candidates with a minimum of 9 000 hours of software engineering experience.

This paper presents the curriculum of a course in requirements engineering and management intended for software development professionals with a first degree in computing (computer science, computer engineering, software engineering, information systems, or information technology) and experience in developing software solutions in real-world (industrial) contexts. The course must therefore concentrate on topics that were not taught during the first degree in computing and that can have a positive impact on the professionals' daily practice. We also discuss how software development professionals can attend the course for the following purposes:

- to obtain academic recognition of the education that they have received beyond the first-cycle degree by embedding it within a life-long learning scheme of cumulative credits to be used in a second-level academic degree (equivalent to a U.S. Master's degree).
- to get preparation for the software requirements section of the IEEE's CSDP certification examination.

This paper is organized as follows. Section 2 gives an overview of the structure and learning objectives of the course. In section 3, the seven training units that constitute the course are described. Section 4 discusses some issues related to academic and professional certification. Conclusions and some ideas for future work are presented in section 5.

2. Training Course

Most of our students have first-level academic degrees in computing. They work on the development of software applications, either as programmers, testers, or project managers. Typically, people conducting requirements activities also perform work in programming or project management; they sometimes call themselves analysts/programmers to reflect the fact that they assume several technical roles when developing software solutions.

Since requirements activities tend to be less technical than other software development activities, computing students and even faculty often perceive them as less important than, for example, programming, testing, or design. This clearly suggests that software developers, after leaving university, are generally not prepared to perform requirements activities in industrial contexts [14]. This reality is consistent with the report in [16], which indicates that there is a general weakness in requirements engineering knowledge in industry. Furthermore, requirements engineering tasks in industry are often performed in a non-systematic manner. For example, requirements elicitation tasks tend to be handled in a completely ad-hoc manner.

978-0-7695-3539-5/09 $25.00 © 2009 IEEE
DOI 10.1109/CSEET.2009.24

The lack of requirements engineering expertise in the software development industry suggests a serious need for further education [3]. Creating an appropriate course requires careful attention to the skills of the target audience and the number of hours available. We assume that our students have a first-level academic degree in computing, which means that they have solid basic skills in computing and that they are familiar with a computational approach to problem-solving. This typically implies that the students will have great strength in software construction but may lack other software engineering competencies, such as software process, software method, project management, and organizational issues.

Due to market pressure, companies are sometimes not willing to allow their employees to spend much time in educational activities. It is also the case that employees are not interested in further education in software engineering, either because they do not recognize the usefulness of further education, or because they rate further education as less important to their careers than other activities. These observations imply that a program of further education courses needs to be very attractive to potential students and provide them with skills and knowledge that can be rapidly and directly applied to their daily tasks. Requirements engineering is highly suited for this purpose. As a discipline, it encompasses a large set of competences and knowledge areas, which include elicitation, analysis, documentation, review, modeling, conflict resolution, prioritisation, team communication, and problem identification. All of these are directly applicable in the daily life of a software development professional.

The course described here totals 48 contact hours. Given the limited amount of time available in the course, it is not possible to cover the entire scope of requirements engineering and management. Therefore, one of the main challenges in devising the course was to select topics that give the students a general idea of the main concepts, approaches, theories, and challenges associated with requirements engineering and management, as well as some practical and useful methods, guidelines and hints for applying these concepts, approaches, theories, and challenges to real-world software projects.

The course begins with an introduction, followed by six units that address key topics in requirements engineering and management:

1) Introduction to requirements engineering;
2) Requirements elicitation;
3) Requirements prioritization;
4) Requirements negotiation;
5) Guidelines for writing requirements;
6) Requirements modeling and specification;
7) Interdependencies and impact analysis.

Each unit consists of 6 contact hours, so that the units can be completed in two sessions of 3 hours each. In total, the course consists of 48 contact hours: 42 are used for the sessions, and 6 hours are used for intermediate and final assessments.

Wherever possible, each unit begins with a small exercise that the students are asked to solve, either individually or in small groups. In this way, the students get a feeling for the problems addressed by the unit. Some of the exercises are games or invented situations that the students must play or analyse. After the students spend 30 minutes working on the exercise, the instructor makes a presentation and encourages discussion of the relevant topics. This typically takes approximately 4 hours and 30 minutes. Before the instructor's presentation, the students are asked to identify some of the topics that they expect to be discussed. For example, in the unit on requirements elicitation, the students are asked to describe the requirements elicitation process that they follow and the elicitation techniques that they use or know about. In the remainder of the unit, students spend about one hour working again on the initial exercise, but they now solve it using the techniques presented in the unit. The students' solutions are then discussed by all of the participants in order to promote discussion and to allow the instructors to get immediate feedback from the students.

The learning objectives for the course state that students that successfully complete the course should be able:

- To define the intervention that the requirements engineers must execute during the software life cycle and to identify the expected involvement of all the stakeholders.
- To decide how requirements should be captured, specifically by identifying all sources of requirements and the techniques that should be used.
- To detect and solve conflicts among the candidate requirements, based on agreed and negotiated solutions.
- To handle the requirements document from structure, quality and verifiability perspectives. This document should address both the user and the system requirements.
- To analyze and evaluate the requirements document to make sure that it describes the system under consideration, by inspecting (or formally revising) the document, or by constructing prototypes.
- To construct models of user and system requirements based on information obtained during the elicitation activities.
- To manage requirements change during the software life cycle by adopting traceability techniques.

3. Course units

In this section, all the seven training units that are part of the devised training course are described. We explain which topics are covered by each training unit and show how it is organized.

3.1. Introduction to Requirements Engineering

In this unit, the instructor focuses on the software requirements knowledge area as a critical domain of software engineering, as outlined in the IEEE Computer Society's Software Engineering Body of Knowledge (SWEBOK) [1]. This unit focuses on defining what a requirement is, on the distinction among different types of requirements (user requirements vs. system requirements; functional requirements vs. non-functional requirements), and on discussing the requirements process and its associated activities. Despite the fact that this unit is primarily devoted to the instructor's presentation of the material, students are encouraged to participate by asking questions or making comments, and also to reflect on the specific requirements problems they face daily. The material for this course unit is mainly based on [1], [2], [17].

3.2. Requirements Elicitation

In this unit, the instructors make a short presentation about requirements elicitation. They define the concept and emphasize its communicative nature. Subsequently, the students are asked to list all the requirements elicitation techniques they know about. All the techniques mentioned are written on the blackboard. When a student names a technique, he or she is asked to describe it succinctly. This initial exercise is used to draw the attention of the trainees to the full range of techniques used for requirements elicitation: interviews, questionnaires, task analysis, domain analysis, introspection, group work, brainstorming, joint application development, ethnography, prototypes, goal modelling, scenarios, viewpoints, persona, video and photographs, wikis and blogs, mind maps, creativity workshops, family therapy.

All of these techniques are presented to the class by the instructor and then discussed. The instructor focuses on the techniques that the students seem to use or enjoy the most, but also covers other techniques that he or she feels might improve the practices of the students. The unit ends by showing students some project contexts and asking them to identify the techniques that seem most appropriate for eliciting the project requirements. This promotes discussion and gives the instructors immediate feedback on the students' comprehension and retention of the material taught. The material for this unit is mainly based on [17], [22].

3.3. Requirements Prioritization

The unit begins by asking each student to prioritize a set of requirements for a software system. The students are given no criteria or prioritization techniques to be used in this exercise. However, this exercise provides background for the rest of the unit, where the following issues are presented and discussed:

1) the need for requirements to be prioritized,
2) the aspects of requirements prioritization (importance, penalty, cost, time, risk, volatility),
3) the prioritization techniques (AHP, 100-unit test, numerical assignment, ranking, top-10 requirements), and
4) the stakeholders involved in the prioritisation process.

The unit ends by asking the students to redo the initial exercise, but this time to do so by selecting a set of criteria and a particular technique. The material for this unit is mainly based on [6].

3.4. Requirements Negotiation

The unit begins by asking each student to choose six people (from a list of twelve) who are to be put in a bunker in case of a bombing attack. For this exercise, the students are expected to apply some of the prioritization techniques addressed in the previous unit. Students are later asked to form groups. Each group is asked for a list of the people who will go to the bunker. The preparation of group lists requires some sort of negotiation. Students are later asked to play a small collective game, that follows some of the characteristics of the Prisoner's Dilemma, a well-known problem in game theory. The experience acquired in these two exercises provides background for the rest of the unit, which presents and discusses:

1) the need for requirements to be negotiated,
2) the negotiation process, and finally
3) the dimensions of requirements negotiation (conflict resolution strategy, collaboration situation of stakeholders, and degree of tool support).

The material for this unit is essentially based on [8], [9].

3.5. Guidelines for Writing Requirements

In this unit, students are initially challenged with two small requirements documents: one document is very difficult to understand, while the second has a well-structured set of requirements. The students' experience with trying to understand the requirements for the two systems spurs discussion about the advantages of having requirements written in a clear, methodical, and unambiguous way. The unit focuses on describing a set of practical guidelines for writing good requirements (e.g., using simple sentences, using a limited vocabulary, and defining verifiable criteria) and on analyzing the structure of a requirements specification template. The material for this course unit is mainly based on [2], [17].

3.6. Requirements Modeling and Specification

This unit addresses modeling issues in software engineering, focusing on the models that are most relevant for the

activities of the requirements engineering process. At the beginning of the unit, students are asked to identify different types of models that they typically use in their software development projects.

Later, the instructor introduces the notions of abstraction, refinement, and complexity, and then presents the key characteristics of a useful model: abstract, understandable, accurate, predictive, and inexpensive [18]. Several types of models particularly relevant to requirements engineering are also introduced and described: state-oriented models (finite-state machines and high-level Petri nets), activity oriented models (use case models, data-flow diagrams and flowcharts), structure-oriented models (UML deployment and component diagrams), and data-oriented models (entity-relationship diagrams and Jackson structured diagrams). At the end of the unit, students are given a small software problem and are asked to model it with the techniques introduced in the unit. The models built by the students are presented and discussed in the session. The material for this training unit is mainly based on [13].

3.7. Interdependencies and Impact Analysis

This unit deals with questions related to requirements traceability. The instructors introduce horizontal and vertical traceability and requirement interdependencies (structural, constraint, cost/value). Issues related to impact analysis, including automatable and manual strategies and metrics are also discussed. At the end of the unit, students are given a set of requirements related to a software system and are requested to use the concepts introduced in the unit to describe their dependencies. Later, changes to the requirements are progressively introduced; students are then asked to use appropriate techniques to deal with these changes. The solutions obtained by the students are presented and discussed in the session. The material for this course unit is mostly based on [4], [11], [7].

4. Academic and Professional Certification

Although the course described above was explicitly created for the immediate use of software development professionals, we believe that the life-long learning efforts of these professionals can be formally recognized both for academic and professional credentials.

Since the requirements engineering course is designed for software developers with first-level (i.e., equivalent to U.S. Baccalaureate) academic degrees in computing, the course content, the level of detail and the approach towards problem solving justifies considering the course as a second-level (equivalent to U.S. Master's level) education activity. The learning outcomes of the course are compatible with those defined for second-level academic degrees by the EU Dublin Descriptors that were adopted in the Bologna Process [12]. These general descriptors are concerned with the following student competences: (1) knowledge and understanding, (2) applying knowledge and understanding, (3) making judgments, (4) communication skills, and (5) learning skills. The course was therefore designed with explicit attention to the principles stated in the EU Bologna Declaration.

The course is given over a period of 8 weeks; each week consists of 2 sessions of 3 hours. These 16 sessions correspond to a total of 48 contact hours. In addition to attending the course sessions, each student must devote up to 92 hours to self-study, project development and writing an essay. The global effort of the training course therefore consists of 140 hours, which corresponds to 5 ECTS (European Credit Transfer and Accumulation System) units.

At the end of the training course, each student who accomplishes the course's learning outcomes is credited with 5 ECTS units toward one of Universidade do Minho's Master's programs that include a major in software engineering. This mechanism is used for each of the five software engineering courses (requirements engineering and management, software architecture and design, software process and maturity, software project planning and control, software costs and management) that Universidade do Minho has been offering to the local software industry. Each student who has completed the five courses is granted the equivalent of the first year of a Master's program and is allowed to start the Master's dissertation.

The instructors of the requirements engineering course are faculty members with research interests in requirements engineering as well as experience with real-world industrial problems. These faculty members have a strong background in requirements engineering, as well as an understanding of the specific needs and motivation of the students. They can therefore discuss the way in which the methods and techniques presented in the course are used to solve real-world problems.

When applying for the CSDP certification offered by the IEEE Computer Society (www.computer.org/csdp), a software developer must possess a minimum of 9 000 hours of software engineering experience within at least six of the eleven knowledge areas covered by the CSDP program: (I) business practices and engineering economics, (II) software requirements, (III) software design, (IV) software construction, (V) software testing, (VI) software maintenance, (VII) software configuration management, (VIII) software engineering management, (IX) software engineering process, (X) software engineering tools and methods, and (XI) software quality. The IEEE Computer Society offers an online course to prepare software professionals for the CSDP examination. However, this course is intended as a review for professionals who have already acquired the necessary knowledge. For software professionals who have not had much exposure to particular topics in software engineering, additional training is generally needed. The course in re-

quirements engineering and management described in this paper is intended to provide the necessary training in this area. It covers all the material included in CSDP knowledge area II (software requirements) and its fundamental bibliographic resources are the three volumes referenced by the CSDP program [20], [21], [15].

We feel that our effort to assure a complete alignment of our industrial courses with the first academic year of the Master's program at Universidade do Minho, as well as with the software requirements knowledge area of the CSDP examination, is a key feature that is very beneficial to students, whether they only seek additional professional training or intend to continue their formal education with the eventual goal of acquiring a Master's degree.

5. Conclusions and Related Work

In our opinion, the major challenge is how to introduce a practical component into the course in the eight weeks available. From student feedback, it seems clear that most of the students from industry enjoy the material presented in the course units, because it usually presents topics that the students were not taught in their undergraduate studies. However, the students also expect a more practical flavor in the course, so that they can see how to rapidly put into practice some of the concepts, techniques, and approaches.

It might be a good idea to use some of the ideas proposed in [5]. In this paper, Beatty and Agouridas suggest the use of extended exercises to help students practice requirements engineeriing techniques and become aware of the associated limitations and potentials.

We would like also to put into practice the tag-team lecturing approach, as proposed in the context of requirements engineering by Svahnberg and Gorschek [19]. This approach uses two instructors in the classroom. One is responsible for the lecture in the traditional way. The second instructor gives supplementary information, whenever appropriate. We have already used tag-team lecturing in workshops on requirements engineering and management for industry, and feedback indicates that this style of lecturing is seen as making lectures more dynamic. In particular, we believe that tag-team lecturing facilitates students' participation in class discussions. Once students have seen that multiple perspectives are acceptable, they are more willing to share and discuss their own perspectives with the class.

The experiences in teaching WinWin described in [10, sect. 5.2] are also interesting, since the authors discuss some practical issues related to teaching requirements negotiation topics. We expect to incorporate some of those ideas in future delivers of the course.

It has proved to be difficult to get students' full attention to courses that are offered at their place of employment, due to the proximity of daily work demands. We highly recommend that software engineering courses directed to professionals be delivered outside of the daily work environment. Ideally, this should take place in university facilities.

In summary, we feel that the requirements engineering course has been very successful, and we regard it as a template for courses in the other SWEBOK/CSDP areas.

As future work, we plan to similarly analyze and describe courses in other software engineering knowledge areas (software architecture and design, software process and maturity, software project planning and control, software costs and management) that we have been offering to professionals working in the software industry in our region. The sharing of our experiences with these courses will enable the assessment of the adequacy of this set of topics as components of a life-long learning program for experienced software professionals who want to proceed towards second-level academic qualifications and/or professional certification.

References

[1] Alain Abran, James W. Moore, Pierre Bourque, and Robert Dupuis. *Guide to the Software Engineering Body of Knowledge: 2004 Edition — SWEBOK*. IEEE CS Press, New York, United States, 2004. ISBN 0-7695-2330-7.

[2] Ian F. Alexander and Richard Stevens. *Writing Better Requirements*. Addison-Wesley, 2002. ISBN 978-0-321-13163-8.

[3] Jocelyn Armarego. *Educating Requirements Engineers in Australia: effective learning for professional practice*. Ph.D. thesis in Information Technology, School of Computer and Information Science, University of South Australia, February 2007.

[4] Robert Arnold and Shawn Bohner. *Software Change Impact Analysis*. Wiley-IEEE Computer Society Press, June 1996. ISBN 978-0-8186-7384-9.

[5] Joy Beatty and Vassilis Agouridas. Developing Requirements Engineering Skills: A Case Study in Training Practitioners. In Didar Zowghi and Jane Cleland-Huang, editors, *Proceedings of the First and Second International Workshop on Requirements Engineering and Training (REET2005 and REET2007)*, pages 6–17. Aardvark Global Publishing, March 2008. ISBN 978-1-4276-3012-4.

[6] Patrik Berander and Anneliese Andrews. Requirements Prioritization. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 69–94. Springer, 2005. DOI 10.1007/3-540-28244-0_4.

[7] Åsa G. Dahlstedt and Anne Persson. Requirements Interdependencies: State of the Art and Future Challenges. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 95–116. Springer, 2005. DOI 10.1007/3-540-28244-0_5.

[8] Roger Fisher and William L. Ury. *Getting to Yes: Negotiation Agreement Without Giving*. Penguin Books, 1983. ISBN 0-1400-6534-2.

[9] Paul Grünbacher and Norbert Seyff. Requirements Negotiation. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 143–162. Springer, 2005. DOI 10.1007/3-540-28244-0_7.

[10] Paul Grünbacher, Norbert Seyff, Robert O. Briggs, Hoh Peter In, Hasan Kitapci, and Daniel Port. Making Every Student a Winner: The WinWin Approach in Software Engineering Education. *Journal of Systems and Software*, 80(8):1191–200, August 2007. DOI 10.1016/j.jss.2006.09.049.

[11] Per Jönsson and Mikael Lindvall. Impact Analysis. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 117–142. Springer, 2005. DOI 10.1007/3-540-28244-0_6.

[12] JQI. Shared 'Dublin' descriptors for Short Cycle, First Cycle, Second Cycle and Third Cycle Awards. Technical report, Joint Quality Initiative Informal Group, 2004. Available at www.jointquality.nl. Accessed 22.09.2008.

[13] Ricardo Machado, Isabel Ramos, and João M. Fernandes. Specification of Requirements Models. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 47–68. Springer, 2005. DOI 10.1007/3-540-28244-0_3.

[14] Oliver Minor and Jocelyn Armarego. Requirements Engineering: A Close Look at Industry Needs and a Model Curricula. *Australasian Journal of Information Systems*, 13(1):192–208, September 2005.

[15] J. Fernando Naveda and Stephen B. Seidman. *IEEE Computer Society Real-World Software Engineering Problems: A Self-Study Guide for Today's Software Professional*. Wiley-IEEE Computer Society Press, 2006. ISBN 978-0-471-71051-6.

[16] Uolevi Nikula, Jorma Sajaniemi, and Heikki Kälviäinen. A State-of-the-practice Survey on Requirements Engineering in Small- and Medium-sized Enterprises. Technical report, Telecom Business Research Center, Lappeenranta University of Technology, Lappeenranta, Finland, 2000.

[17] Suzanne Robertson and James C. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 2006. ISBN 978-0-321-46797-3.

[18] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25, September 2003. DOI 10.1109/MS.2003.1231146.

[19] Mikael Svahnberg and Tony Gorschek. Multi-perspective Requirements Engineering Education with focus on Industry Relevance. In Didar Zowghi and Jane Cleland-Huang, editors, *Proceedings of the First and Second International Workshop on Requirements Engineering and Training (REET2005 and REET2007)*, pages 88–97. Aardvark Global Publishing, March 2008. ISBN 978-1-4276-3012-4.

[20] Richard H. Thayer and Mark J. Christensen. *Software Engineering: Volume 1, The Development Process*. Wiley-IEEE Computer Society Press, 3rd edition, December 2005. ISBN 978-0-471-68417-6.

[21] Richard H. Thayer and Merlin Dorfman. *Software Engineering: Volume 2, The Supporting Processes*. Wiley-IEEE Computer Society Press, 3rd edition, September 2005. ISBN 978-0-471-68418-3.

[22] Didar Zowghi and Chad Coulin. Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 19–46. Springer, 2005. DOI 10.1007/3-540-28244-0_2.